Edge-Minimum Walk of Modular Length in Polynomial Time

Antoine Amarilli
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
LTCI, Télécom Paris, Institut polytechnique de Paris, France

Benoît Groz
Paris-Saclay University, CNRS, LISN, France

Nicole Wein
University of Michigan, Ann Arbor, MI, USA

Abstract

We study the problem of finding, in a directed graph, an *st*-walk of length r mod q which is edge-minimum, i.e., uses the smallest number of *distinct* edges. Despite the vast literature on paths and cycles with modularity constraints, to the best of our knowledge we are the first to study this problem. Our main result is a polynomial-time algorithm that solves this task when r and q are constants.

We also show how our proof technique gives an algorithm to solve a generalization of the well-known Directed Steiner Network problem, in which connections between endpoint pairs are required to satisfy modularity constraints on their length. Our algorithm is polynomial when the number of endpoint pairs and the modularity constraints on the pairs are constants.

In this version of the article, proofs and examples are omitted because of space constraints. Detailed proofs are available in the full version [3].

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Shortest paths

Keywords and phrases Directed Steiner Network, Modularity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.5

Related Version *Full Version*: `https://arxiv.org/abs/2412.01614` [3]

Funding *Antoine Amarilli*: Amarilli was partially supported by the ANR project EQUUS ANR-19-CE48-0019, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758, and by the ANR project ANR-18-CE23-0003-02 ("CQFD").

Acknowledgements We are grateful to the reviewers of the conference version for their helpful feedback. We also would like to thank the Simons Institute Fall 2023 programs "Logic and Algorithms in Database Theory and AI" and "Data Structures and Optimization for Fast Algorithms" for the initiation of this work. Last, we are grateful to Xiao Hu and Mikaël Monet for early discussions.

1 Introduction

We begin with a simple question: Given an n-vertex, m-edge directed graph G and terminals s,t, can we efficiently find an odd-length *st*-walk that is "edge-minimum", i.e., has the minimum number of *distinct* edges? This question may appear similar to classical problems from the vast literature on paths and cycles with parity constraints. For instance, one may think of the classical "shortest odd *st*-path" problem, which is well-known to be NP-hard (this is via a simple reduction from the 2-disjoint paths problem of [19], and can be proved in the same way as [37, Proposition 2.1]). However, this hardness result only applies to *simple paths*, whereas the edge-minimum odd *st*-walk may not be a simple path. One may also think of the "shortest odd *st*-walk" problem (i.e., minimizing the length), which is well-known to

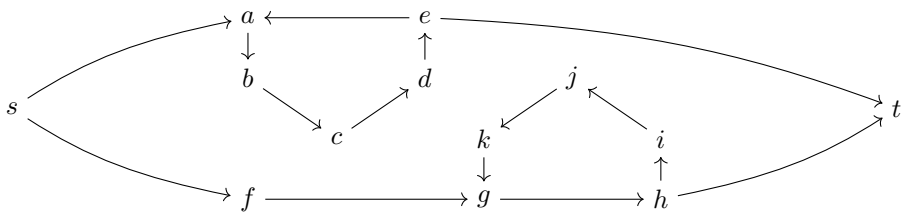

Figure 1 An example of a graph with different answers to the problems of finding the edge-minimum odd st-walk, the shortest odd st-walk, and the shortest odd simple st-path. First, every simple path from s to t is of even length, so there is no shortest odd st-path. Second, the *shortest* odd st-walk uses the bottom cycle $s, f, g, h, i, j, k, g, h, t$: it has length 9 and uses 8 distinct edges. Third, the *edge-minimum* odd st-walk uses the top cycle: $s, a, b, c, d, e, a, b, c, d, e, t$. It has length 11 but uses only 7 distinct edges.

be in polynomial time[1]. However, the *edge-minimum* odd st-walk may be *longer* than the *shortest* odd st-walk while using *fewer distinct edges*. See Figure 1 for an example of a graph in which the solutions to all three of these problems is different.

The focus of this paper is the *Edge-Minimum Walk of Modular Length* problem, which is the above problem generalized to an arbitrary modularity q and remainder r. Let us define it formally:

Edge-Minimum Walk of Modular Length (EWM):
Input: An unweighted directed[2] graph G, terminals s,t, and non-negative integers $r < q$.
Output: An st-walk of length r mod q which is edge-minimum, i.e., uses the minimum number of distinct edges (or $\emptyset$ if no such walk exists).

We stress that the modularity constraint does not apply to the number of distinct edges, but only to the length of the walk. We are most interested in the regime where q, and hence r, are constants; our algorithms will work without this assumption, but they achieve worse complexities.

Despite the vast literature on paths and cycles of given modularities [31, 26, 37, 36, 39, 27, 4, 38, 10, 41, 34, 32, 29, 40, 24, 1, 35, 6, 25, 14, 16, 7, 23] (see [2] for a survey), to the best of our knowledge we are the first to study the EWM problem.

EWM can also be viewed as a problem in the field of *network design*. Network design problems ask questions of the form "find an edge-minimum subgraph with a certain property". Famous network design problems include, for instance, Minimum Spanning Tree, Traveling Salesperson, and st-Shortest Path. We can give an equivalent rephrasing of EWM in this way: find an edge-minimum subgraph that contains an st-walk of length r mod q. The network design problem most related to EWM is the Directed Steiner Network problem (DSN) [17, 18, 8, 20, 15, 9, 33, 21, 13]. In DSN, the input is a directed graph G and a set of k terminal pairs $(s_1,t_1),\ldots,(s_k,t_k)$; the goal is to find an edge-minimum subgraph that

[1] Make two copies of the vertex set, and for every edge $u \to v$ in the original graph, add the edges $u_1 \to v_2$ and $u_2 \to v_1$. Then find the shortest walk from s_1 to t_2. This also trivially generalizes to modularities which are polynomial in n.

[2] One could also ask this question on an undirected graph. However, in this case, one unnatural aspect of the problem is that one can always traverse the same edge back and forth over and over to add any multiple of 2 to the path length. We use this observation to show in the full version [3] that EWM on undirected graphs reduces to EWM on directed graphs.

contains a path $s_i \to t_i$ for all terminal pairs. When k is arbitrary, DSN generalizes Directed Steiner Tree and is therefore NP-hard. When k is constant, Feldman and Ruhl [17] showed that DSN can be solved in polynomial time. In fact, we show in Section 8 that, in the special case where the set of terminal pairs is strongly connected, the DSN problem with constant k can be directly expressed as a special case of EWM with constant q. As a consequence, our main result will imply, as a byproduct, the known result that this special case of DSN is in polynomial time for constant k [17] (albeit with a larger exponent). We also study a problem generalizing both EWM and DSN and give an algorithm that subsumes the tractability of both problems, as we will explain later in the introduction.

EWM is also related to problems from database theory, in particular the evaluation of *regular path queries* (RPQs) on graph databases. More precisely, a graph database is a graph whose edges are labeled with symbols from a fixed alphabet, and an RPQ is a regular expression e over the alphabet. The results of the RPQ are vertex pairs (s, t) such that there is a walk from s to t (or sometimes a simple path or a trail, depending on the variant) whose edge labels from a word that matches e (see e.g. [12, 11, 5, 28]). In particular, RPQs of the form $a^r(a^q)^*$ express walks of length r mod q for constant r and q. The EWM problem then corresponds to the *smallest witness problem* [30, 22] for such queries, which is the problem of finding a sub-database of minimum size that satisfies the query. However, to the best of our knowledge there is no prior work on the smallest witness problem for RPQs enforcing modularity conditions, or indeed for RPQs in general: our work can be seen as a first step towards addressing this problem.

Our results. What is the complexity of EWM for constant q? It is unclear a priori; EWM is related to both polynomial-time solvable problems such as Shortest Modular-Length st-Walk and Directed Steiner Network, as well as NP-hard problems such as Shortest Modular-Length st-Path. Our main result is that EWM is in polynomial time for constant q:

▶ **Theorem 1.1.** *There is an algorithm solving EWM in time:* $n^{O(\log q)} \cdot 2^{O(q \log^2 q)}$.

Specifically, the exponent of n in Theorem 1.1 has reasonable constants: $7 + 3\log_2 q$, though we did not focus on optimizing them.

Generalizations. A natural generalization of EWM is to consider weighted graphs. EWM admits two natural notions of weights: (1) each edge has a *length*, and the length of a walk is the sum of the lengths of the edges, and (2) each edge (or vertex) has a *cost* and the edge-minimum walk is measured in terms of the sum of the costs of the edges (or vertices) that it traverses. We show in Section 7 that our algorithm extends to accommodate costs, and that it can also accommodate lengths provided that their values are polynomial. In the case where lengths and the value q are super-polynomial, we show that the problem is NP-complete by reduction from Subset Sum.

Inspired by Directed Steiner Network, we also extend our results in Section 8 from one walk to multiple walks. The input is a directed graph G, and k endpoint pairs along with modularity constraints: $(s_1, t_1, q_1, r_1), \ldots, (s_k, t_k, q_k, r_k)$, and the goal is to find an edge-minimum subgraph that contains a walk from s_i to t_i of length r_i mod q_i, for all i. Generalizing our main result, we show (Theorem 8.4) that this problem, which subsumes the DSN and EWM problems, also admits a polynomial-time algorithm for constant q_i values and a constant number k of endpoint pairs. Our approach to show the result focuses on the case of strongly connected solutions (in line with the connection to DSN mentioned earlier), before extending the proof to arbitrary solutions by re-using lemmas from [18].

2 Technical Overview

Our work is related to the Directed Steiner Network (DSN) problem studied by Feldman and Ruhl [17], where we must find an edge-minimum subgraph of the input directed graph which satisfies connectivity requirements. Their work shows that the DSN problem is tractable when the number of endpoint pairs in the connectivity requirements is constant. Following their work, Feldmann and Marx [18] have investigated the parameterized complexity of the DSN problem. One key idea of their approach is to show bounds on the *cutwidth* of solutions to the DSN problem. Informally, cutwidth is a parameter that limits how many edges are "cut" when arranging vertices in a well-chosen order $<$. Bounding the cutwidth of a graph ensures that we can go over vertices in the order of $<$ and that, at any cut along $<$, all edges except a constant number will be entirely to one side of the cut. (See Section 3 for the formal definition of cutwidth.)

Feldmann and Marx show that when the number of endpoint pairs is bounded by a constant k, then DSN instances always have a solution of constant *undirected* cutwidth (depending only on k). They further show that bounded-cutwidth solutions can be computed in polynomial time by dynamic programming. We follow the same framework and show that solutions to EWM also have bounded cutwidth. Then we can find the solution by an approach similar to the dynamic programming algorithm of [18], or to the token game of [17].

Thus, our main goal is to solve the purely structural problem of bounding the cutwidth of edge-minimum st-walks of length $r \bmod q$ in arbitrary directed graphs. We only focus on this goal in the rest of the technical overview. To provide intuition, we will begin by considering the case of $q = 2$, $r = 1$ (i.e. odd walks), and then outline the obstacles that arise when generalizing to arbitrary q, r.

Odd walks. Let w denote an edge-minimum odd-length st-walk, and let G_w be the subgraph of G consisting of the union of edges in the walk w. We first observe that, without loss of generality, w does not contain any even cycles: If w contained an even cycle, we could simply delete it, and w would still be of odd length, and still be edge-minimum. (To clarify, we are not referring to even cycles in G_w, but rather even cycles in the walk w itself. After deleting a cycle C from w, edges from C can still appear in G_w if they are traversed at another point of the walk w.)

Now suppose w contains two odd cycles in succession. Again, we could simply delete both cycles, and w would still be of odd length, and still be edge-minimum. So, we can suppose without loss of generality that w has either no cycles (and trivially has small cutwidth), or consists of a simple path P_1, followed by an odd cycle C, followed by a simple path P_2. We assume this latter case in the rest of the discussion, and we define C by looking at first time that the walk re-visits a previously visited vertex: this ensures that, by construction, P_1 is vertex-disjoint from C. However, the graph does not necessarily have small cutwidth, because P_2 could intersect P_1 and C in intricate ways forming an unbounded number of nested cycles. It is a priori conceivable that, e.g., a grid-like structure with high cutwidth could emerge. However, we can show that this does not happen.

First, we can observe that once P_2 leaves C, without loss of generality it does not return. This is because if P_2 leaves C at vertex a and then returns at vertex b, we can delete the subpath of P_2 from a to b, and instead traverse only the edges of C to get from a to b. This way, we can still ensure that w is of odd length since every traversal of C changes the parity of w, and we can traverse C for "free" without adding any extra edges.

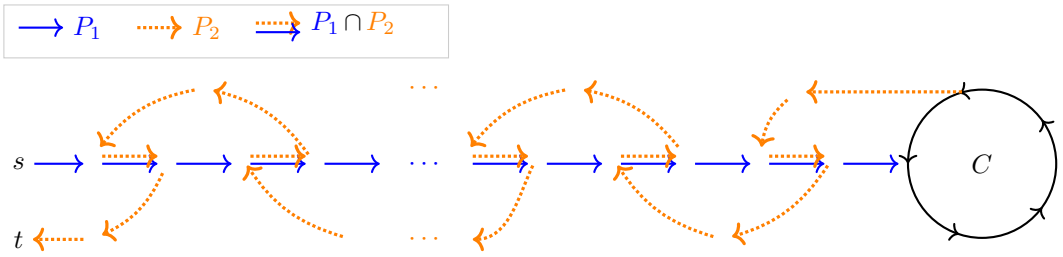

Figure 2 Specific shape of solutions to EWM for $q = 2$. Each "edge" in the figure represents a subpath, not necessarily a single edge. The cycle C is of odd length.

Finally, we consider how P_1 and P_2 can interact. We can observe that without loss of generality P_2 cannot hit a vertex a on P_1, then leave P_1, then return to a *later* vertex b on P_1. In this case we could delete the subpath of P_2 from a to b, and instead traverse P_1 to get from a to b. Again, we can still ensure that w is of odd length since we can always traverse C for free to change the parity of w. We stress that this argument only holds when P_2 re-enters P_1 at a *later* vertex, and in fact, P_2 can re-enter P_1 at an *earlier* vertex an unbounded number of times.

With all of these observations in mind, we conclude that if w has a cycle, then w must have the very specific structure depicted in Figure 2. Then, it is visually clear that any vertical cut only has a constant number of crossing edges, and thus w has constant cutwidth.

This concludes the outline for the case of odd walks. The same argument holds for even walks, and more generally an argument for a particular choice of constants q, r is easily extendable by reduction to other constant values r' with the same q, simply by adding a new source connected to the original source by a path of constant length $r' - r \bmod q$. The challenging part is extending to arbitrary q. Our approach is to define a *segment decomposition* of the walk and prove that each successive segment allows us to expand the reachable set of remainder values, so that the number of segments is bounded as a function of q (Lemma 4.3). We then show that the cutwidth of a walk can be bounded linearly in its number of segments (Proposition 5.1): we do this by defining a suitable ordering of the vertices following the *chunks* of the walk, which refine the segments in the decomposition. These two results imply that solutions to EWM have bounded cutwidth and hence that they can be found in polynomial time.

3 Preliminaries

In this section, we present preliminary notions used throughout the proof of our main result (Theorem 1.1).

Basic definitions. All graphs in the paper are finite and directed. Graphs are not necessarily simple; they may contain self-loops, but they do not contain multi-edges. Given a directed graph $G = (V, E)$, a *walk* w in G is a sequence of edges $w[1], \ldots, w[\ell]$ such that the target vertex of $w[i]$ is equal to the source vertex of $w[i+1]$ for each $1 \leq i < \ell$. Note that an edge e of G may occur at several positions in G: considering a position i for which $w[i] = e$, we write $w[i]$ to refer to that occurrence of e at the i-th position of w. However, when convenient we abuse notation and identify $w[i]$ with e itself; e.g., we talk about the source and target vertices of $w[i]$ to mean those of e. For $1 \leq i \leq \ell$ we call $w[i]$ a *first-visited* edge occurrence if it is the first occurrence of some edge e, i.e., if there is no $j < i$ such that $w[j] = w[i]$. Otherwise, $w[i]$ is a *revisited* edge. Of course, each edge that occurs in w is first-visited at exactly one position.

The *source vertex* s of w is that of $w[1]$, and its *target vertex* t is that of $w[\ell]$: we then call w an *st-walk*. The *length* $|w|$ of w is ℓ. The *set of vertices used by* w, denoted V_w, is simply the set of vertices that occur in w, i.e., occur in some edge of w; in particular the source and target vertices of w are in V_w. The *set of edges used by* w is $E_w := \{w[i] \mid 1 \leq i < \ell\}$. The *subgraph spanned by* w is $G_w = (V_w, E_w)$. Note that G_w is *not* the subgraph induced by V_w, as it only contains the edges that actually occur on the walk. A *subwalk* of w is a contiguous subsequence of w. We denote subwalks of w with the slicing convention, i.e., $w[: i] = w[1] \dots w[i]$ and $w[i : j] = w[i] \dots w[j]$. Note that the right endpoint is included so that, e.g., $w[i : j]$ is empty precisely when $j < i$, and $w[: i]$ is empty precisely when $i \leq 0$. When u and v are walks and when the target vertex of u is the source vertex of v, we write uv to mean the walk obtained by concatenating u and v: it admits u and v as subwalks, and of course $|uv| = |u| + |v|$.

For convenience, throughout the paper we denote by $G_{w,i}$ the graph $G_{w[:i]}$ spanned by the subwalk $w[: i]$ (up to i included); by convention $G_{w,0}$ is always the graph with the single vertex s and no edges.

Strongly connected components. A *strongly connected component* (SCC) of a graph G is a maximal set of vertices C of G such that, for any two distinct vertices $u, v \in C$, there is a directed path from u to v in G. As usual, belonging to the same SCC is an equivalence relation, so that SCCs form a partition of the vertices of G. We say that an SCC C of G is *non-trivial* if the subgraph of G induced by C contains at least one edge. This is the case if and only if C contains more than one vertex, or contains a single vertex on which there is a self-loop.

EWM and solutions. We study the *Edge-Minimum Walk of Modular Length* problem (EWM), as defined in the introduction: given a directed graph $G = (V, E)$, terminals s,t, and non-negative integers $r < q$, we wish to compute a subset E' of E such that (V, E') has an st-walk of $r \bmod q$ and E' is edge-minimum (i.e., the number of edges of G' is minimum). This phrasing is somewhat different from the one given in the introduction, in which we wanted to compute an edge-minimum walk. However, we can easily compute an edge-minimum walk from E' by a product construction, in time $O(|E'| \times q)$.

In the sequel, we only consider the computation of edge-minimum subsets of edges (rather than walks). When fixing inputs $G = (V, E)$, s, t, q, and r, we talk about a *candidate solution* to mean a subset E' of E such that (V, E') has an st-walk of length $r \bmod q$ but E' is not necessarily edge-minimum, and of an *optimal solution* to mean a candidate solution which is additionally edge-minimum.

Cutwidth. The *cutwidth* is a structural parameter of graphs. We show that there always exists an optimal solution to EWM with bounded cutwidth, which suffices to ensure that such solutions can be found with an efficient algorithm. We first recall the definition of cutwidth for undirected graphs. Let $G = (V, E)$ be an undirected graph. An *ordering* of G is a total order over the vertex set V of G. A *cut* (V_-, V_+) of G that respects the ordering $<$ is a partition $V_- \uplus V_+ = V$ such that $v_- < v_+$ for all $(v_-, v_+) \in V_- \times V_+$. We say that an edge e of G *crosses the cut* if it has one endpoint in V_- and one in V_+, i.e., $e \cap V_-$ and $e \cap V_+$ are both nonempty. Note that self-loops never cross cuts. The *cutwidth* of a cut (V_-, V_+) that respects $<$ is the number of edges that cross this cut, and the *cutwidth* of $<$ is the maximum cutwidth of a cut that respects $<$. The *cutwidth* of G is then the minimum cutwidth of an ordering of V.

We now define cutwidth for directed graphs. Let $G = (V, E)$ be a directed graph. Its *underlying undirected graph* is $G' = (V, E')$ where $E' = \{\{u, v\} \mid (u, v) \in E, u \neq v\}$: note that self-loops are not reflected in G'. We then define the *cutwidth* of G to be that of G'. We underscore that our definition of cutwidth for directed graphs is always the undirected cutwidth: we do not consider the directed cutwidth in this paper. We then define the *cutwidth* of a walk w of G as the cutwidth of the directed graph G_w.

4 Segment Decomposition of a Walk

Our proof of our main theorem (Theorem 1.1) consists of three steps, spanning this section and the next two sections. First, in this section we introduce the notion of the *segment decomposition* of a walk, and we show that any walk w can be transformed into a walk w' that satisfies the same modularity conditions (i.e., has the same remainder modulo q), uses a subset of the edges (i.e., $E_{w'} \subseteq E_w$), and has a segment decomposition with $O(\log q)$ segments. Second, in the next section, we will show how the cutwidth of the graph $G_{w'}$ spanned by w' can be bounded linearly in this number of segments, implying that optimal solutions to EWM have bounded cutwidth. Third, in Section 6, we show that this cutwidth bound makes it possible to solve EWM in polynomial time.

Segment decomposition. Let w be a walk. Its *segment decomposition* is a sequence of walks $s_1, \ldots, s_\xi$ such that $w = s_1 \cdots s_\xi$; the number ξ is the *number of segments* of w. The segment decomposition is defined by processing the walk w from left to right as follows. Initially, the segment decomposition is the empty sequence, and the entire walk w remains to be decomposed. At any point of the decomposition, we have already computed the first σ segments $s_1, \ldots, s_\sigma$ for some number $\sigma \geq 0$ of segments, and can write $w = s_1 \cdots s_\sigma w'$. If w' is non-empty, we compute the next segment as a prefix of w'.

Informally, we read w' and terminate the segment whenever we have a first-visited edge $w'[j] = (u, v)$ followed (not necessarily immediately) by an edge $w'[k] = (x, y)$ where y can be reached by a path from u using only edges of w up to $w'[j]$ excluded. The intuition of segments is that they capture a moment where the walk revisits a vertex y from a vertex u via a path that starts with an edge $e = (u, v)$ which had not been previously visited in w, even while there was already a path from u to y in the graph spanned by the walk before e was visited. Formally, we will look for the *segment end* inside the suffix w' of w, according to the following definition:

▶ **Definition 4.1** (Segment end, Segment detour). *Having fixed $w = s_1 \cdots s_\sigma w'$, let $\lambda_\sigma = |s_1 \cdots s_\sigma| = |w| - |w'|$ be the total length of the already-computed segments. For $\lambda_\sigma < k \leq |w|$, we say that segment $s_{\sigma+1}$* ends at *k if k is minimum such that the following hold:*

- *There is $\lambda_\sigma < j \leq k$ such that $w[j]$ is first-visited in w; we write $w[j]$ as (u, v);*
- *Writing the edge $w[k]$ as (x, y), then there is a path from u to y in the graph $G_{w,j-1}$ which contains the edges of the walk w up to $w[j]$ excluded. Note that, in particular, if $u = y$ then this requirement is always satisfied using the empty path from u to y.*

We stress that the path from u to y required by the second item of the definition is a path in the graph spanned *by a certain prefix of w; it may be the case that the path does not occur as a subwalk of w. As for the subwalk $w[j : k]$ from u to y in w, which is a subwalk of w', we call it the* segment detour *of $s_{\sigma+1}$, denoted $\det_{\sigma+1}$. Note that, having chosen the minimal k, there could be multiple valid choices for j, and we pick one arbitrarily.*

Given the definition of a segment end above, the next segment of the decomposition after s_σ is simply $s_{\sigma+1} := w[\lambda_\sigma + 1 : k]$. This means that we terminate the $(\sigma+1)$-th segment right after the edge $w[k]$, and continue the decomposition if $s_1 \cdots s_{\sigma+1}$ does not yet cover the entire walk w. If there is no segment end at any k, then we take all the rest of the walk w' to be the last segment of the decomposition and we finish. Note that for this reason, unlike other segments, the last segment of the decomposition may not feature a segment detour.

Let us formalize which paths are known to exist at the moment when a segment ends:

▶ **Lemma 4.2.** *Let w be a walk and let $w[i:k] = s_\sigma$ be the σ-th segment of the walk. Let y be the ending vertex of $w[i:k]$, and assume that $k < |w|$, i.e., the segment finishes before the end of the walk. Let $w[j] = (u, v)$ in $w[i:k]$ be the first edge of the segment detour $\det_\sigma$. There is:*

- *the segment detour $\det_\sigma$ from u to y in $G_{w,k}$*
- *a path p_σ from u to y in $G_{w,j-1}$.*
- *a path $\overline{p_\sigma}$ from y to u in $G_{w,j-1}$.*

Notice that we may have $y = u$, and then p_σ and $\overline{p_\sigma}$ may be empty; but $\det_\sigma$ is never empty because it contains at least $w[j]$.

In the sequel for each $1 \leq \sigma < \xi$ we denote by p_σ and $\overline{p_\sigma}$ a pair of paths obtained by applying Lemma 4.2 (if there are multiple valid choices for p_σ and $\overline{p_\sigma}$, we choose arbitrarily).

Achievable differences. For a walk w and segment σ, we denote by $\Delta_q(w, \sigma)$ the value $(|\det_\sigma| - |p_\sigma|) \bmod q$ and call this the[3] *difference achievable by σ*. We often drop the subscript q when clear from context. Intuitively, we know that we can modify the walk w to replace the subwalk $\det_\sigma$ from u to y by the existing path p_σ that goes from u to y, and this change subtracts $\Delta_q(w, \sigma)$ from the length of the walk modulo q; we will make this formal in the sequel. It should also be intuitively clear that achievable differences can be assumed to be non-zero in an edge-minimum walk: intuitively, an achievable difference of 0 means that we can replace the detour $\det_\sigma$ by the existing path p_σ that has the same endpoints, without changing the length of the walk modulo q. We know that this change does not harm the edge-minimality of w because p_σ consists of already visited edges. What is more, having too many segments with the same achievable difference is useless, because we can intuitively replace $\det_\sigma$ by p_σ and compensate the effect of this change by doing similar substitutions in earlier segments. Then, thanks to Lagrange's theorem on the additive group $\mathbb{Z}/q\mathbb{Z}$, the number of segments can in fact be bounded by $1 + \log_2 q$. We formalize this in the following lemma, which is the main result of this section:

▶ **Lemma 4.3** (Segment Decomposition Lemma)**.** *For every st-walk w with length $r \bmod q$, there is another st-walk of length $r \bmod q$ using only edges from w whose number of segments is at most $1 + \log_2 q$.*

The rest of this section is devoted to proving the Segment Decomposition Lemma; we will then use it in the next section to bound the cutwidth of some edge-minimum solution.

Recall that a *preorder* over a set is a binary relation that is both reflexive and transitive (this is a weaker requirement than non-strict partial orders which are additionally required to be antisymmetric). To prove the Segment Decomposition Lemma (Lemma 4.3), we define

[3] Note that there could be several choices for the difference achievable by σ depending on the arbitrary choices made earlier, e.g., depending on the first edge j for the segment detour, and on the choice of paths p_σ and $\overline{p_\sigma}$; nevertheless, we fix one single value $\Delta_q(w, \sigma)$ according to the choices made.

a preorder over the edge-minimum walks that are using a specific set of edges. Intuitively, the preorder favors walks which do their first visit of edges as late as possible relative to the end of the walk. More precisely, the preorder criterion asks us to minimize the number of remaining steps of the walk at the moment where the last first-visited edge is visited; then break ties by minimizing the number of remaining edges at the moment where the second-to-last first-visited edge is visited; and so on. This is designed to ensure that replacing a detour $\det_\sigma$ by p_σ improves the walk according to the preorder.

Let us fix from now on the graph $G = (V, E)$, the source s and target t, and the integers r and q. For any subset of edges $E' \subseteq E$, we define an *E'-walk* to mean an st-walk of length r mod q in G which uses precisely the edges of E'. Let us then define the order:

▶ **Definition 4.4.** *Let $E' \subseteq E$ be a subset of edges and let $m := |E'|$. Let w be an E'-walk, and let $i_1, \ldots, i_m$ be the indexes of the first-visited edges in ascending order, i.e., $w[i_1], \ldots, w[i_m]$ are the first-visited edges of w and $i_1 < \cdots < i_m$. The* first-visited timestamp *of w is then the m-tuple $(|w| - i_m, \ldots, |w| - i_1)$.*

We define as follows a preorder relationship $\trianglelefteq$ called the timestamp preorder *on E'-walks. Let w and w' be two E'-walks, and let t_w and t'_w be their first-visited timestamps, respectively. Then we have $w \trianglelefteq w'$ if we have $t_w \leq t'_w$ in lexicographic ordering.*

In other words, the timestamp preorder compares two walks by the number of remaining edges to traverse when visiting the last first-visited edge, then the second-to-last, and so on. Note that the timestamp preorder is not antisymmetric because two different walks on the same set of edges may happen to have the same first-visited timestamp (i.e., they visit first-visited edges at the same positions from the end, even though the identity of these edges and the revisits may be different). We then define:

▶ **Definition 4.5.** *If w is an E'-walk, we say w is* timestamp-minimum *if, for every E'-walk w', we have $w \trianglelefteq w'$.*

Note that, whenever there is an E'-walk, then there is a timestamp-minimum E'-walk, but it is not necessarily unique because the timestamp preorder is not antisymmetric. We are now ready to conclude the section by proving the Segment Decomposition Lemma (Lemma 4.3):

Proof sketch. We show that a timestamp-minimal walk has at most $1 + \log_2 q$ segments. We consider the successive subgroups of $\mathbb{Z}/q\mathbb{Z}$ generated by the achievable differences of the successive segments, and show that these must be a strictly increasing subsequence, so that the bound follows by Lagrange's theorem. To do this, we show as an intermediate claim that walks can be modified to augment their length by any combination of achievable differences of previous segments, while still using the same edges. Thanks to this claim, assuming by contradiction that there is a segment s_σ whose achievable difference is already achievable using the preceding segments, then we rewrite the segment to replace its detour $\det_\sigma$ by the path p_σ from Lemma 4.2, and use the claim to modify the walk and fix its length. We then show that this modification yields a walk which is smaller in the timestamp preorder, contradicting the minimality of w. See the full version [3] for the full proof. ◀

5 Number of Segments and Cutwidth Bounds

In this section, we continue our proof of Theorem 1.1 by showing that, for any walk w, the number of segments in the decomposition of the previous section gives a bound on the cutwidth of G_w up to a constant factor. This result is completely independent from the definition of the EWM problem. Formally, we show:

▶ **Proposition 5.1** (Segment Cutwidth Bound). *For any walk w, letting ξ be the number of segments in its segment decomposition, the cutwidth of G_w is at most 3ξ.*

We remark that there is no converse of this result: a walk formed of a succession of cycles connected by single edges will have an unbounded number of segments but has cutwidth 2. Combining Proposition 5.1 with Lemma 4.3 establishes the following:

▶ **Corollary 5.2.** *For every st-walk w' with length $r \bmod q$, there is another st-walk w of length $r \bmod q$ using only edges from w' such that the cutwidth of G_w is at most $3 + 3\log_2 q$.*

This implies the following bound on the cutwidth of optimal solutions to the EWM problem, on which our algorithm relies:

▶ **Corollary 5.3.** *For any graph $G = (V, E)$, terminals s,t, and non-negative integers $r < q$, every optimal solution to the EWM problem on G, s, t, r, and q has cutwidth at most $3 + 3\log_2 q$.*

To prove Proposition 5.1, we need several intermediate steps. First, we define the notion of *chunk* of a walk, which is a contiguous sequence of first-visited edges whose intermediate vertices are also first-visited. Second, we define the ordering $\prec$ on the vertices of G_w along which the cutwidth bound will be shown: this order is defined using the notion of chunks, intuitively because all vertices of a chunk will be ordered relative to the already-visited vertices that are the endpoints of the chunk (also distinguishing special cases like *cycle chunks* and *tadpole chunks*). Third, we show that the cutwidth along the order $\prec$ is bounded by 3ξ, by showing for each segment that the number of times its first-visited edges can cross the cut is at most 3. Throughout this section, we fix an arbitrary walk w in a graph G, and call s and t its source and target vertices.

Chunks. We define first-visited vertices similarly to first-visited edges: for every vertex $v \in V_w$ occurring in w, we say v is *first-visited at $w[i]$* if $w[i]$ is the first edge in which v occurs. Note that a vertex v which is first-visited at $w[i]$ always occurs as the target vertex of $w[i]$, except in the specific case of the first edge $w[1]$ where both the source s and the target vertex are first-visited at $w[1]$. Of course, every vertex of V_w is first-visited at exactly one position. Further, if a vertex v is the target vertex of an edge $w[i]$ and is first-visited at $w[i]$, then both $w[i]$ and $w[i+1]$ (if it exists) must be first-visited edges.

We can now define *chunks*:

▶ **Definition 5.4.** *A* chunk *of w is a maximal subwalk $w[i : j]$ of w where all edges are first-visited, and where, for every $i \leq k < j$ (if any), the target vertex of $w[k]$ is first-visited at $w[k]$.*

In other words, a chunk is a maximal sequence of one or more consecutive first-visited edges such that the first and last vertex are not first-visited (unless they are extremities of the whole walk) but all intermediate vertices are first-visited. A chunk may consist of a single first-visited edge $w[i]$ between two already-visited vertices, in which case there are no intermediate vertices.

The *first* and *last* vertices of chunk $w[i : j]$ are the source vertex of $w[i]$, and the target vertex of $w[j]$, respectively. Note that two successive chunks in the walk need not be separated by revisited edges and can simply be separated by a revisited vertex: for instance, for the length-2 walk $(s, s), (s, t)$, all edges are first-visited, but there are two chunks of length 1 which are separated by the revisit of the vertex s.

It will be useful to distinguish two special kinds of chunks. First, *tadpole* chunks, which loop back on an intermediate vertex of the chunk:

▶ **Definition 5.5.** *A chunk is a* tadpole *if it consists of at least two edges and if, letting $u_1, \ldots, u_\ell$ be the successive vertices that it visits, with u_1 its first vertex and u_ℓ its last vertex, we have $u_\ell = u_{\ell'}$ for some $1 < \ell' < \ell$.*

In other words, a *tadpole* is a chunk that consists of a path (containing at least one edge), followed by a cycle (possibly a self-loop).

Second, *cycle* chunks, which loop back on the vertex from which they started:

▶ **Definition 5.6.** *A chunk is a* cycle *if its first vertex and last vertex are identical.*

Note that these two cases are mutually exclusive. A chunk which is neither a tadpole nor a cycle, and thus is a simple path, is a *normal chunk*.

The notion of chunk must be distinguished from the notion of segments used in the segment decomposition of the previous section, but we can notice the following connections between the two notions:

- Segment ends never happen within a chunk: indeed, the intermediate vertices y reached within a chunk are first-visited by definition, so there is no way to reach them except by the edge that precedes them, and thus no earlier path can reach y in the walk.
- At the end of a tadpole chunk or cycle chunk, the current segment always ends. Indeed, the last edge $e' = (x, y)$ of the chunk reaches a vertex y which already has an outgoing edge in the chunk: this edge is a first-visited edge $e = (y, v)$, and the empty path from y to itself allows us to finish the segment at the end of the chunk, i.e., just after e'.

In summary, chunks are contiguous subsequences of the walk that never straddle segment boundaries, and the end of a tadpole chunk or cycle chunk always triggers the end of a segment. Also note that, by definition, chunks form a partition of the first-visited edges. This implies that, except possibly for the last segment, every segment must contain at least one chunk, because they contain at least one first-visited edge.

Defining the ordering. Having defined the notion of chunks of the walk w, we now define the ordering $\prec$ along which we will show that the cutwidth is bounded. This definition only depends on chunks; it does not depend on the segment decomposition. We see the total order $\prec$ as a sequence of vertices. Initially, the order is the empty order on the single vertex s. Then, for every chunk $w[i : j]$ successively, we consider the (possibly empty) sequence σ of the intermediate vertices of $w[i : j]$. Recall that the first vertex of $w[i : j]$ is already in the domain of $\prec$: either it is s (for the first chunk), or it is a vertex which is already visited. Then there are four cases:

- Tadpole: If $w[i : j]$ is a tadpole, then we insert all its intermediate vertices at the end of the current ordering $\prec$, in the order in which they were first-visited.
- Cycle: Otherwise, if $w[i : j]$ is a cycle, then, letting v be its first and last vertex, we know that v already occurs in $\prec$, and we insert all its intermediate vertices right after the vertex v, in the order in which they were visited.
- Normal: We consider two subcases:
  - First, suppose that the last vertex t of $w[i : j]$ is first-visited at $w[j]$. This is only possible with $j = |w|$, so that $w[i : j]$ is the last chunk. Then we do the same as in the tadpole case: insert the intermediate vertices and t at the end of the current ordering $\prec$, in the order in which they were first-visited.
  - Otherwise, $w[i : j]$ is a chunk whose last vertex v is first-visited at $w[k]$ with $k < i$ so v already occurs in $\prec$, and as we explained the first vertex u of the chunk also already occurs in $\prec$. Further, we have $v \neq u$ because the chunk is not a cycle. Then, we insert

the intermediate vertices so that the vertices along the chunk are ordered in a monotone fashion in the ordering. In other words, let $x_1, \ldots, x_\ell$ be the successive intermediate vertices of the chunk, so that its edges are $(u, x_1), (x_1, x_2), \ldots, (x_{\ell-1}, x_\ell), (x_\ell, v)$. If $u \prec v$ then we insert $x_1, \ldots, x_\ell$ in order between u and v, and if $v \prec u$ we insert $x_\ell, \ldots, x_1$ in order between v and u. The order $\prec$ between the newly inserted elements and the existing elements between u and v is arbitrary, for instance we can arbitrarily say that we insert the new vertices just after the smallest of u and v in $\prec$.

Note that, in all cases above, the (intermediate) vertices of a chunk are always ordered as a monotone sequence.

SCCs of the segment decomposition. We have defined, from our walk w, the order $\prec$ along which we will bound the cutwidth. To show the cutwidth bound, we establish two results describing the SCCs of the graph generated by the walk and its close relationship with our ordering $\prec$:

▶ **Lemma 5.7.** *At each step $0 \leq i \leq |w|$, the successive SCCs $C^i_1, \ldots, C^i_{\kappa_i}$ of $G_{w,i}$ are linearly ordered by the reachability relationship (i.e., every vertex in C^i_b is reachable from every vertex in C^i_a iff $a < b$), and the target vertex of the last edge $w[i]$ is in the last SCC $C^i_{\kappa_i}$ of $G_{w,i}$.*

▶ **Lemma 5.8.** *For every prefix $w[:i]$ of the walk such that a chunk of w ends at $w[i]$, the ordering $\prec$ induced by $w[:i]$ is consistent with the topological order of the SCCs of $w[:i]$.*

Bounding the cutwidth from the number of segments. We can now turn back to the segment decomposition introduced in the previous section for the walk w, and show how the number of segments of w can be used as a bound on the cutwidth of the graph G_w spanned by w, following the order $\prec$ that we defined. For this, we will consider an arbitrary cut $V_- \uplus V_+$ of V_w following $\prec$, and count how many edges of w cross the cut. As each edge is first-visited once, and first-visited in exactly one segment, it suffices to bound how many edges each segment contributes to the cut, and to count only the first-visited edges of each segment. We want to show:

▶ **Lemma 5.9.** *For each segment $w[i:j]$ of the walk w, the number of first-visited edges in $w[i:j]$ that cross the cut is at most 3.*

This immediately implies the Segment Cutwidth Bound (Proposition 5.1) stated at the beginning of the section. This bound of 3 edges crossing the cut can be proved by a case analysis summarized in the following lemmas:

▶ **Lemma 5.10.** *Let $w[i:j]$ be a segment of w and assume that the last chunk of $w[i:j]$ is a cycle $w[k:j]$. Then $w[k:j]$ crosses the cut at most twice. Further, if it does cross the cut twice then it must be the case that the first and last vertex of $w[k:j]$ is in V_- and that $w[k:j]$ contains some intermediate vertex in V_+.*

▶ **Lemma 5.11.** *Let $w[i:j]$ be a segment of w and assume that the last chunk of $w[i:j]$ is a tadpole $w[k:j]$. Then $w[k:j]$ crosses the cut at most twice. Further, if it crosses the cut twice then it has an intermediate vertex in V_- and an intermediate vertex in V_+.*

▶ **Lemma 5.12.** *Let $w[i:j]$ be a normal chunk of w, then it crosses the cut at most once.*

▶ **Lemma 5.13.** *Let $w[i:j]$ be a normal chunk of w which crosses the cut forwards, i.e., the starting vertex of the chunk is in V_- and its ending vertex is in V_+. Then the segment containing $w[i:j]$ ends at j.*

All that remains is to bound the contribution to the cut of the normal chunks that cross the cut backwards. To this end, let us distinguish the last chunk of a segment which we call the *final* chunk, and the remaining chunks in that segment which we call *non-final* chunks. Non-final chunks must be normal (because the segment ends right after cycle chunks or tadpole chunks), and they cannot cross the cut forward by the previous lemma.

We are ready to conclude the proof of Lemma 5.9:

Proof sketch. We consider a segment and consider the SCC decomposition of the graph of the walk when the segment starts, using Lemma 5.7. By Lemma 5.8 the segment begins in the rightmost SCC according to the order. Now the intuition is the following: if all chunks but the final one stay on the same side of the cut, then only the final chunk can cross the cut and the preceding lemmas (Lemma 5.10, Lemma 5.11 and Lemma 5.12) show that the bound is satisfied.

Otherwise, there is a first non-final chunk which ends on some SCC C_ϕ that contains nodes to the left of the cut and this chunk thus potentially crosses the cut (backwards, by Lemma 5.13). If the next chunk is final, the bound is satisfied, so we may assume the next chunk is not final. This next chunk may also cross the cut, but in any case it has for target an SCC C_ψ which is further left than C_ϕ, a property we exploit to show that the walk cannot return to the right of the cut without terminating the segment. This implies that the cut is crossed at most 4 times in total: once for each of the two non-final chunks considered, and potentially twice for the final chunk. We lower the bound to 3 by showing that in fact the final chunk can only cross the cut once if it is preceded by two chunks that already crossed the cut. See the full version [3] for the full proof. ◀

6 Computing an Edge-Minimum Walk of Bounded Cutwidth

Up to now, we have shown Corollary 5.3: optimal solutions to the EWM problem have cutwidth at most $3 + 3\log_2 q$. In this section, we conclude the proof of Theorem 1.1 by showing that optimal solutions of bounded cutwidth can be efficiently found.

At a high level, our algorithm proceeds by searching for a shortest path in a graph of configurations. The approach is similar to the approaches in [17] and [18]. The intuitive principle of our algorithm is the following. As we follow a path in the graph of configurations and move from one configuration to another, we choose which edges of the original graph to keep in the solution. To be more precise, the path will start on the empty configuration which denotes that no edges have been selected, and will iteratively add vertices and some of their incident edges to the solution.

To make sure that the selected edges do form a solution, we could try to record in the configurations the exact set of edges kept in the solution so far. However, the space of configurations would then be exponential. To avoid this, a configuration does not really store the entire set of chosen edges: instead it concisely represents the possible lengths modulo q of walks between a small subset of vertices of G, whose size is bounded as a function of the cutwidth.

The distance in the configuration graph from the initial configuration to any configuration Ξ will reflect the smallest cardinality of a set of edges that achieve the set of walks witnessed by Ξ. The algorithm therefore looks for a shortest path in the configuration graph from the initial configuration to any configuration which witnesses the st-walk of length r mod q required by the EWM problem.

Formally, fix the input graph $G = (V, E)$, the source $s \in V$ and the target $t \in V$, and the modularity requirement $r \bmod q$. Let $\omega \in \mathbb{N}$ be a *domain size bound*, which is related to the cutwidth, but will be precisely defined later. Let us define the notion of configurations formally:

▶ **Definition 6.1** (Configuration). *A V, ω, q-*configuration *$\Xi = (D, \rho)$ is a subset D of at most ω vertices of V, called the* domain *of the configuration, together with a function ρ mapping each ordered pair (u, v) for $u, v \in D$ to a subset of $\{0, \ldots, q-1\}$. Having fixed the vertex set V of the input graph G, we denote by $\Phi_{\omega,q}$ the set of all possible V, ω, q-configurations, and omit the subscript when clear from context.*

To compute transitions in the configuration graph, our algorithm will need to compute the *closure* of a configuration (D, ρ). The point of the closure is to ensure that, whenever we can achieve a walk from $u \in D$ to $v \in D$ via intermediate vertices of D and using remainders given by ρ, then that walk can be witnessed directly by a value in $\rho(u, v)$. Formally:

▶ **Definition 6.2** (Closure). *Given a configuration $\Xi = (D, \rho)$, an* internal walk *of Ξ is a sequence of vertices of D together with a choice of remainder for each step. Formally, it is a sequence $v_1, \ldots, v_\ell \in D$ and a choice of remainders $0 \leq r_1, \ldots, r_{\ell-1} < q$ such that we have $r_i \in \rho(v_i, v_{i+1})$ for each $1 \leq i < \ell$. The* total length *of the internal walk is the sum of the remainders modulo q, namely, $\sum_{1 \leq i < \ell} r_i \bmod q$.*

The closure *of (D, ρ) is the configuration (D, ρ') where, for each $u, v \in D$, for each $r' \in \{0, \ldots, q-1\}$, we have $r' \in \rho'(u, v)$ iff there is an internal walk of total length r' from u to v in (D, ρ). Note that we always have $\rho'(u, v) \supseteq \rho(u, v)$ because for each $r' \in \rho(u, v)$ we can take the single-edge internal walk u, v with remainder r'.*

We can easily compute the closure of a configuration in polynomial time in q and ω, for instance using a product construction. Specifically, create a graph on vertices $D \times \{0, \ldots, q-1\}$, then add the following edges: for each $u, v \in D$ and each $r' \in \rho(u, v)$, for each $i \in \{0, \ldots, q-1\}$ create an edge from (u, i) to $(v, i + r' \bmod q)$. Then compute the transitive closure and define $\rho'(u, v)$ to be the set of r' such that $(u, 0)$ has a path to (v, r').

We now define the graph over configurations, which we call $\Gamma_{\omega,q}$ and which also depends on the directed graph G given as input to EWM. We omit the subscripts when clear from context.

▶ **Definition 6.3.** *The* graph of configurations $\Gamma_{\omega,q}$ *is a weighted and labeled directed graph: each edge carries an integer cost and is labeled by a subset of edges of the original graph G. The vertex set of Γ is the set Φ of all V, ω, q-configurations. To define the edges, let us choose any configuration $\Xi \in \Phi$ and define the outgoing edges of Ξ. These edges are of two kinds:*

- Forget*: from $\Xi = (D, \rho)$ with D nonempty, for each $v \in D$, letting $D' := D \setminus \{v\}$ be the new domain, we have an edge leaving Ξ which has cost 0, is labeled with the empty set of edges, and leads to $(D', \rho_{|D'})$ where $\rho_{|D'}$ is the restriction of ρ to D'*
- Introduce*: from $\Xi = (D, \rho)$ with $|D| < \omega$, for each $v \in V \setminus D$, let $D' := D \uplus \{v\}$ be the new domain, and let $E_{v,D'}$ be the set of edges of G which are of the form (v, v') or (v', v) with $v' \in D'$; we also add to $E_{v,D'}$ the self-loop edge on v if it exists. Then for each $E' \subseteq E_{v,D'}$, we have an edge leaving Ξ which has cost $|E'|$, is labeled by E', and leads to the closure of the configuration (D', ρ') with ρ' intuitively defined from ρ by adding the edges of E', formally:*
  - *For all $u, u' \in D \cup \{v\}$, we initialize $\rho'(u, u') := \emptyset$.*
  - *For each $(u, u') \in D \times D$, we set $\rho'(u, u') := \rho(u, u')$.*
  - *For each edge $e \in E_{v,D'}$, we set $\rho'(e) := \{1\}$. Note that by definition of $E_{v,D'}$ this case is disjoint from the previous one.*

Note that we may have several Introduce edges with the same source and target configurations but labeled with different sets of edges and with different costs; in this case all of these edges exist in Γ *as defined above, although we could equivalently have decided to keep only one of these edges among those with minimum cost.*

We have now defined the graph Γ of configurations. We will look for shortest paths in this graph from the *initial configuration* to a *final configuration*, namely:

▶ **Definition 6.4.** *The* initial configuration *is simply the configuration with empty domain.*

A configuration is final *if it contains* s *and* t *and features a walk with the requisite remainder, i.e., the configuration* (D, ρ) *is final if* $s, t \in D$ *and* $r \in \rho(s, t)$*.*

We can now define our algorithm to solve the EWM problem, which we call the *EWM algorithm*. We first set the value ω, following the cutwidth bound, to be: $\omega := 6 + 3\log_2 q$, i.e., three more than the cutwidth bound that follows from Corollary 5.3. (Intuitively, we add 2 to make sure that the sources s and t can always be part of the domain of configurations, and we add 1 extra to make sure that we can always perform Introduce steps before Forget steps.) The EWM algorithm then builds explicitly the graph $\Gamma_{\omega,q}$ and computes a shortest path π in Γ from the initial configuration to a final configuration. Once such a shortest path π is found[4], then the algorithm returns the subgraph of G formed of the edges of G obtained as the union of all edge labels in π.

Proving that the EWM algorithm correctly solves the EWM problem is relatively straightforward: we show that whenever the algorithm return a subgraph, this subgraph is indeed a candidate solution. Conversely, we show that for every optimal solution E' there is a path from in the configuration graph from some initial configuration to some final configuration, such that the union of labels along the path is E' without duplicates (hence the path has cost $|E'|$), which guarantees that the algorithm returns the optimal solution. Those properties can be proved by establishing the following invariant:

▷ Claim 6.5. Let $\pi = \Xi_1, \ldots, \Xi_\ell$ be a path in Γ from the initial configuration Ξ_1. Let $G_1 = (V, E_1)$, …, $G_\ell = (V, E_\ell)$ be the sequence of subgraphs of G defined in the following way: we have $E_1 = \emptyset$, and for each $1 < i \leq \ell$ we set $E_i = E_{i-1} \cup E'_i$ where E'_i is the edge set that labels the edge of Γ used to go from Ξ_{i-1} to Ξ_i in Γ. Then the following is true: for any $1 \leq i \leq \ell$, writing $\Xi_i = (D_i, \rho_i)$, for any $u_1, u_2 \in D_i$, for any $r' \in \{0, \ldots, q-1\}$, we have $r' \in \rho_i(u_1, u_2)$ iff there is a u_1u_2-walk in G_i whose length modulo q is r'.

Note that, in the definition of the graphs G_i, we may add the same edge of G multiple times because it occurs in the label of several edges of π; this can happen even if π is a simple path in Γ. As it turns out, this never happens when π is a shortest path (because we are then, in effect, paying twice for the same edge); but the invariant of Claim 6.5 applies to paths π even if they do traverse edges labeled with non-disjoint edge sets.

Altogether, we have explained why the algorithm correctly solves the EWM problem, and the algorithm can be shown to have complexity $n^{\omega+1} \cdot 2^{O(\omega^2 q)}$. Instantiated with our choice of $\omega = 3 + 3\log_2 q$, we get a final complexity of $n^{O(\log q)} \cdot 2^{O(q \log^2 q)}$ for the algorithm. See the full version ([3]) for details. This concludes the proof of our main result (Theorem 1.1).

[4] If there is no path in Γ from an initial configuration to a final configuration, then we return $\emptyset$ to indicate that there is no subgraph of G with an st-walk of length r mod q. Note that this case can be excluded from the outset, simply by checking whether G contains an st-walk of length r mod q: this can be done, e.g., with the product construction.

7 Extension to Weighted Graphs

Having shown our main result (Theorem 1.1), in this section we start exploring variants and extensions of the EWM problem. We first study two extensions of EWM in this section. First, we study the addition of *costs* on edges, meaning that we look for a walk where the total cost is minimum (instead of the number of edges). Second, we study the addition of integer *lengths* on edges.

In this section and the next, the problems that we define and study are always posed on a directed graph G and ask about the existence of a subgraph of G satisfying certain properties and which is optimal according to some criterion (usually, being edge-minimum). We use the same terminology as before: a *candidate solution* is a subgraph which satisfies the properties but is not necessarily edge-minimum, and an *optimal solution* is a candidate solution which is additionally edge-minimum.

Costs on edges and vertices. We first study the extension of the EWM problem with costs on edges. Specifically, the *EWM problem with costs on edges* takes as input a directed graph $G = (V, E)$, a cost function γ giving to each edge $e \in E$ a cost $\gamma(e)$, a pair of a source $s \in V$ and target $t \in V$, and a modularity requirement r mod q for integers q and r. The output to the EWM problem with costs on edges is an st-walk of length r mod q such that the cost $\gamma(E_w) := \sum_{e \in E_w} \gamma(e)$ is minimum. We assume that all costs given by γ are nonnegative. Indeed, in the presence of negative weights, we lose the correspondence explained in Section 3: computing a minimum-weight *subgraph* that contains a walk reduces to the case of positive weights (all negative-weight edges will always be included in the optimal solution subgraph); but computing a minimum-weight *walk* is NP-hard even without modularity constraints:

▶ **Proposition 7.1.** *The following problem is NP-complete: given a directed graph $G = (V, E)$, a cost function $\gamma \colon E \mapsto \mathbb{Z}$, and source and target $s, t \in V$, compute a minimum-weight subset of edges $E' \subseteq E$ such that there is an st-walk using precisely the edges in E'.*

Our tractability result for EWM (Theorem 1.1) can be easily extended to solve the EWM problem with costs on edges. Specifically, the Segment Decomposition Lemma (Lemma 4.3) shows that there must be an optimal solution to EWM with costs on edges that obeys the bound on the number of segments, because the γ function on subset of edges is monotone. Then the Segment Cutwidth Bound (Proposition 5.1) applies as is, and the algorithm to find an optimal solution of bounded cutwidth from the previous section can be easily extended: simply modify the definition of the graph of configurations (Definition 6.3) so that the cost of an edge labeled with a set E' of edges of G is no longer the cardinality $|E'|$ of E but the total cost $\gamma(E')$. Other than that, the algorithm proceeds in the same way, and computes a shortest path which now reflects the subgraph of G satisfying the requirements which has minimum cost instead of minimum cardinality.

We also mention that the EWM problem can be defined to have costs (unit costs or not) on vertices instead of edges. In this case, the cost of a candidate solution is the number of different vertices traversed by the walk (or more generally their total cost). However, this problem is interreducible to the EWM problem with costs on edges:

▶ **Lemma 7.2.** *The EWM problem (with unit costs on edges) reduces to the EWM problem with unit costs on vertices, and the EWM problem with costs on edges reduces to the EWM problem with costs on vertices. Conversely, the EWM problem with unit costs on vertices reduces to the EWM problem with unit costs on edges, and the EWM problem with costs on vertices reduces to the EWM problem with costs on edges.*

Lengths on edges. Having studied the use of costs on edges to change the optimization criterion, we turn to a different problem variant where we annotate edges with integer lengths. Specifically, the *EWM problem with lengths* takes as input a directed graph $G = (V, E)$, a length function $\delta\colon E \to \mathbb{N}$, and a pair of a source $s \in V$ and target $t \in V$ and a modularity requirement $r \bmod q$ for integers q and r. The answer to the EWM problem with lengths is an st-walk w that traverses a minimum number of distinct edges and whose total length is $r \bmod q$, i.e., $\delta(w) := \sum_{1 \leq i \leq |w|} \delta(w[i])$ is $r \bmod q$. (Note that the length of an edge is summed as many times as the edge is traversed.)

The impact of allowing lengths is different depending on the regime. In the case where r and q are constant, or given in unary, then we can rewrite the graph in polynomial time to eliminate edge lengths. Specifically, letting m be the number of edges in the graph, we replace each edge e of length $\delta(e)$ by a path on $(m+1)q + (\delta(e) \bmod q)$ edges (where m is the number of edges in the graph). This ensures that a walk in the new graph has the same modularity as the corresponding walk in the original graph, and the cost of a candidate solution in the rewritten graph is $(m+1)qM + \epsilon$, where M is the number of original edges traversed and $\epsilon \leq mq$. This ensures that an optimal solution in the original graph indeed minimizes the number of edges taken from the original graph.

One different regime is when r and q are written in binary and do not necessarily have polynomial value. In this case, the EWM problem with lengths written in binary can be shown to be NP-hard by an easy reduction from Subset Sum. In fact, just the problem of deciding the existence of a walk of length $q \bmod r$ is NP-hard:

▶ **Proposition 7.3.** *The problem of deciding whether an st-walk of length $q \bmod r$ exists, where q, r and the lengths of δ are written in binary, is NP-hard.*

We do not know whether an analogous hardness result holds for the original EWM problem (without edge lengths) in the setting where q and r can be written in binary and do not necessarily have polynomial value. Indeed, in the proof above, we crucially use the edge lengths to concisely write the numbers given in the Subset Sum instance; writing them in unary as paths of the corresponding length will not give a polynomial-time reduction and indeed the Subset Sum problem can be solved in pseudo-polynomial time.

We last address the question of showing an upper bound on the complexity of EWM with lengths, by showing an NP upper bound. Of course the bound is phrased on the decision version of EWM with lengths, i.e., given an instance of EWM with lengths and a threshold k, we wish to decide whether there exists a candidate solution having at most k different edges. We have:

▶ **Proposition 7.4.** *The following problem is in NP: given a directed graph $G = (V, E)$, a function $\gamma\colon E \to \mathbb{N}$ that assigns lengths (written in unary) to each edge, integers q, r and k (written in binary), source and target $s, t \in V$, decide if there is a subset $E' \subseteq E$ of at most k edges that contains an st-walk of length $r \bmod q$ according to γ.*

8 Connections to DSN and SCSS

We conclude the paper by exploring how EWM relates to the Directed Steiner Network (DSN) problem mentioned in the introduction, and more specifically the *Strongly Connected Steiner Subgraph (SCSS)* problem. We first define the SCSS problem and show that it is subsumed by EWM, in the sense that a polynomial algorithm for EWM (with fixed q and r) gives a polynomial algorithm for SCSS with a constant number of terminals. Then, we introduce a problem generalizing both SCSS and DSN on the one hand, and EWM in the other: we study

how to find the smallest subgraph satisfying connectivity requirements on specified endpoint pairs with specified modularities. We show that we can generalize our results to this problem, and provide a polynomial algorithm for the setting when the number of endpoint pairs and the modularity requirements are constants.

Reducing SCSS to EWM. Let us recall the definition of the SCSS problem [17, 18] before explaining how it can be reduced to EWM. In the SCSS problem, the input simply consists of a directed graph $G = (V, E)$ with an input set $T \subseteq V$ of terminals, and we want to find the smallest subgraph $E' \subseteq E$ that is strongly connected and contains edges incident to each terminal in T. In other words, SCSS is a special case of DSN where the connectivity requirements on the vertices of T require that they are strongly connected. The SCSS problem is NP-hard in general (by an easy reduction from the directed Steiner tree problem) but it can be solved in polynomial time provided that the size of T is bounded by a constant, as shown in [17]. Thus, for $k \in \mathbb{N}$, we write k-SCSS to refer to the SCSS problem where the set T of terminals is required to contain at most k vertices. Following our previous terminology in the paper, when we talk of *candidate solutions* we mean a subgraph satisfying the requirements of a problem (e.g., for k-SCSS, a strongly connected subgraph containing the requisite terminals), and by *optimal solution* we mean a candidate solution which is also edge-minimum.

Let us show that the SCSS problem can be reduced to EWM, which implies that Theorem 1.1 gives an alternative proof of the results of [17] (with worse bounds). More precisely, we show:

▶ **Lemma 8.1.** *For any constant $k > 0$, we can compute constants q and r in $2^{O(k \log k)}$ such that there is a linear-time reduction from k-SCSS to EWM with the constant values q and r.*

The previous reduction shows that our EWM problem is intuitively at least as complicated as SCSS, given that the tractability of EWM for constant q implies that of SCSS for constant k. (This being said, we do not claim that the polynomial algorithm given by our results is as efficient as that of earlier works [17, 18].) Alternatively, instead of this black-box reduction, we can also adapt our techniques to recapture the tractability of SCSS by showing a bound on the cutwidth of optimal solutions, using the segment decomposition. We exemplify below how this is done, and will revisit this afterwards when extending EWM to support multiple paths. Recall the notion of timestamp-minimum walks (Definition 4.5), and let us show the following result; note that it applies to *edge-minimal* solutions (not just optimal solutions).

▶ **Lemma 8.2.** *Let $G = (V, E)$ and $T = \{v_1, \ldots, v_k\}$ be an SCSS instance. Every edge-minimal solution H to the instance has cutwidth at most $3k$.*

What is more, for any terminal v_i, considering all H-walks that start and conclude in v_i and visit all terminals from T, then any timestamp-minimum H-walk among those H-walks has at most k segments.

We remark that the cutwidth bound of $3k$ given by this result is slightly better than the bound of $6k$ shown in [18].

Defining Directed Steiner Network with Modularity constraints. The reduction from SCSS to EWM leads to the natural question of whether there could be a problem which subsumes SCSS and EWM, or even DSN and EWM; and a polynomial-time algorithm that subsumes the tractability of all these problems. We now introduce such a problem, dubbed *Directed Steiner Network with Modularity* (DSNM), and show a polynomial-time algorithm to solve it.

▶ **Definition 8.3.** *For $k > 0$, the* k-Directed Steiner Network with Modularity *problem (k-DSNM) takes as input a graph (V, E) and a* k-requirement specification $\mathcal{R}$ *consisting of k tuples (s_i, t_i, r_i, q_i) for $1 \leq i \leq k$ such that $s_i, t_i \in V$ and $q_i > 0$ and $0 \leq r_i < q_i$. Our goal is to compute an optimal solution, i.e., an edge-minimum subgraph H of G such that H contains a walk from s_i to t_i of length $r_i \bmod q_i$ for every $i \leq k$.*

Our last result in this paper is to show that the k-DSNM problem is in PTIME when k and the modulo values q_i are constants. This result subsumes Theorem 1.1 and the tractability of k-DSN for constant k shown in [17, 18].

▶ **Theorem 8.4.** *We can compute in $n^{O(k+\log q)} \cdot 2^{O(q(k+\log q)^2)}$ time an optimal solution to k-DSNM, where q denotes the least common multiple (LCM) of the q_i in the input k-requirement specification.*

Note that our exponents are given up to constant factors, so we do not claim to recover the same exponents as earlier works on k-DSN without modularity constraints (i.e., for $q = 1$).

The overall strategy to prove Theorem 8.4 is to follow the methodology used for EWM, adjusting the constructions presented so far in the paper. Deviating somewhat from EWM, but following prior work about DSN [17, 18], in our study of DSNM we will focus on the SCCs of an optimal solution and study them separately, instead of studying the entire solution graph. Our main objective is to bound the cutwidth of SCCs in an optimal solution as a function of q and of the number k of endpoint pairs. We can then easily deduce like in [18] that the cutwidth of optimal solutions is bounded as a function of q and k. To compute the solutions of bounded cutwidth, we then modify slightly the algorithm of Section 6.

To study the SCCs of optimal solutions, it will be useful to introduce an analogue of the SCSS problem featuring modularities. Specifically, we call k-SCSSM the problem that takes the same input as k-DSNM and returns a smallest *strongly connected* subgraph H of G such that H contains a walk from s_i to t_i of length $r_i \bmod q_i$ for every $i \leq k$. Note that, unlike the SCSS problem which was simply defined by a set of terminals, in k-SCSSM we specify a set of tuples connecting pairs of vertices, because we need to specify which remainder must be achieved by which endpoint pair. The difference with k-DSNM is only that the solution graph is required to be strongly connected. Our study of k-SCSSM is motivated by the fact that the SCCs of optimal solutions to k-DSNM are optimal solutions to instances of k-SCSSM, as was already known in the setting without modularities [17]. Namely:

▷ Claim 8.5. Let $G = (V, E)$ be a directed graph, and $\mathcal{R}$ be a k-requirement specification. In any optimal solution $H = (V, E')$ to k-DSNM on G for $\mathcal{R}$, for any SCC C of H, there exists a k-requirement specification $\mathcal{R}'$ such that C (as a set of edges) is an optimal solution to k-SCSSM on G for $\mathcal{R}'$.

We next bound the cutwidth of optimal solutions to the k-SCSSM problem specifically. Then we will deduce a cutwidth bound on optimal solutions to k-DSNM using a result of [18]. Finally, we explain how to design an algorithm to compute these solutions.

Bounding the cutwidth of solutions to k-SCSSM Let us start by showing that solutions to k-SCSSM have a small segment number and hence a small cutwidth:

▶ **Lemma 8.6.** *Let G be a graph and $\mathcal{R}$ be a k-requirement specification, denote by $q_1, \ldots, q_k$ the respective modularities of its tuples, and denote by q the least common multiple (LCM) of $q_1, \ldots, q_k$. Then every optimal solution of the k-SCSSM problem on G and $\mathcal{R}$ can be covered with a walk of at most $O(k + \log q)$ segments and therefore has cutwidth at most $O(k + \log q)$.*

Note that this claim only works for the k-SCSSM problem, not the k-DSNM problem, because it assumes the solution can be covered by a single walk which is not true in general for k-DSNM. Let us show the result:

Proof sketch. The proof of this result can be decomposed into three steps. In step 1, we define the notion of a *legal covering walk*, which is a walk that covers the edges of a solution in a prescribed order: first visit all terminals to witness that they are strongly connected in a subwalk w_0 (similarly to Lemma 8.2), then visit all paths with the prescribed modularities in a subwalk w'. In step 2, we carefully impose a variant of timestamp-minimality on legal covering walks towards bounding their number of segments. In step 3, we use Lemma 8.2 (on the first part of the legal covering walk) and an argument adapted from Lemma 4.3 (for the second part) to show that the number of segments is bounded. The point of splitting the walk in two is that w_0 visits enough edges to allow us to move to arbitrary endpoints and ensure strong connectedness: this allows us to replay arbitrary detours in any subwalk no matter where they occur, intuitively "pooling" the achievable differences $\Delta(w, \dots)$ between all the subwalks. See the full version [3] for the full proof. ◀

From k-SCSSM to k-DSNM. We have shown that optimal solutions to the k-SCSSM problem can be assumed to have bounded cutwidth. We now wish to show that the same is true for optimal solutions to the k-DSNM problem. This is simple, using Lemma 8.6 above along with two lemmas from [18]:

▶ **Lemma 8.7.** *Fix a graph G and a k-requirement specification $\mathcal{R}$, and let q be the LCM of the q_i in $\mathcal{R}$. Let H be an optimal solution to k-DSNM on G and $\mathcal{R}$. Then H has cutwidth at most $O(k + \log q)$.*

Computing optimal solutions. We have shown in Lemma 8.7 that optimal solutions to the k-DSNM problem have cutwidth $O(k+\log q)$. The only remaining thing to prove Theorem 8.4 is to adapt the dynamic algorithm from Section 6 to compute solutions to that problem instead of EWM. We explain in the full version [3] how this is done.

9 Future work

We now outline possible directions for future work in light of our results:

Improving our bounds. One natural direction for further research would be to improve the complexity bounds that we show. Our algorithm for EWM runs in time $n^{O(\log q)} \cdot 2^{O(q \log^2 q)}$. Can the factor of q in the exponent be improved, e.g., by getting an algorithm in time $n^{O(\log q)}$? Or, on the other hand, is the problem NP-hard? (We have only shown that the variant of EWM with edge lengths is NP-hard, in Proposition 7.1.)

Intermediate problems between shortest walk and edge-minimum walk. It could be interesting to search for walks that are minimized according to some criterion that is "intermediate" between shortest walk and edge-minimum walk, e.g., if the cost of each edge is expressed as a function of how many times it is traversed by the walk. Such a general framework would capture the two problem variants that we contrast in the introduction: shortest walks are the case where we are charged ℓ when traversing an edge ℓ times, and edge-minimum walks are the case where we are charged 1 when traversing an edge $\ell > 0$ times and charged 0 when we do not traverse it.

Edge-minimum walks satisfying other constraints. Last, one other problem of interest would be to investigate the complexity of finding edge-minimum subgraphs guaranteeing the existence of st-walks satisfying other properties. One very natural example is the following: Given a number ℓ, and a directed graph G with specified vertices s, t, find an edge-minimum st-walk of length exactly ℓ. (This is the same as EWM except the length is exactly ℓ, instead of r mod q.) The trivial algorithm for this problem is in time $O(n^\ell)$, but can we do better? To our knowledge, this problem has not been studied.

Another example is looking for edge-minimum walks that achieve constraints expressed by a finite semigroup. For instance, assume that each edge of the graph is labeled by a semigroup element, and that we want a walk whose evaluation in the semigroup achieves a specific target element of the semigroup, where evaluating the walk means multiplying the labels of its edges in the order that they are traversed. This problem generalizes the EWM problem (with lengths on edges), which uses the semigroup $\mathbb{Z}/q\mathbb{Z}$. An alternative way to phrase this problem is in the language of regular path queries (RPQs) mentioned in the introduction: fixing a regular language L on an alphabet Σ, and given a directed graph G with terminals s and t and with edges labeled by letters of Σ, find an edge-minimum subgraph G with an st-walk which evaluates to a word that belongs to L. For which fixed regular languages L can this problem be solved in polynomial time in G?

References

1 Noga Alon and Michael Krivelevich. Divisible subdivisions. *J. Graph Theory*, 98(4), 2021. doi:10.1002/jgt.22716.

2 Antoine Amarilli. Survey of results on the ModPath and ModCycle problems, 2024. doi:10.48550/arXiv.2409.00770.

3 Antoine Amarilli, Benoît Groz, and Nicole Wein. Edge-minimum walk of modular length in polynomial time, 2024. arXiv:2412.01614.

4 Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *J. ACM*, 38(2), 1991. doi:10.1145/103516.103517.

5 Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. *J. Comput. Syst. Sci.*, 108, 2020. doi:10.1016/J.JCSS.2019.08.006.

6 Andreas Björklund, Thore Husfeldt, and Petteri Kaski. The shortest even cycle problem is tractable. In *STOC*, 2022. doi:10.1145/3519935.3520030.

7 Archit Chauhan, Samir Datta, Chetan Gupta, and Vimal Raj Sharma. The even-path problem in directed single-crossing-minor-free graphs, 2024. doi:10.48550/arXiv.2407.00237.

8 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed Steiner network problem. *ACM Trans. Algorithms*, 7(2), 2011. doi:10.1145/1921659.1921664.

9 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected Steiner network problems. *ACM Trans. Algorithms*, 17(2), 2021. doi:10.1145/3447584.

10 Fan R. K. Chung, Wayne Goddard, and Daniel J. Kleitman. Even cycles in directed graphs. *SIAM J. Discret. Math.*, 7(3), 1994. doi:10.1137/S0895480192225433.

11 Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *PODS*, 1990. doi:10.1145/298514.298591.

12 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *SIGMOD*, 1987. doi:10.1145/38713.38749.

13 Gianlorenzo D'Angelo and Esmaeil Delfaraz. Approximation algorithms for node-weighted directed Steiner problems. In *IWOCA*, 2024. doi:10.1007/978-3-031-63021-7_21.

14 Shagnik Das, Nemanja Draganić, and Raphael Steiner. Tight bounds for divisible subdivisions. *J. Combin. Theory Ser. B*, 165, 2024. doi:10.1016/j.jctb.2023.10.011.

15 Irit Dinur and Pasin Manurangsi. ETH-hardness of approximating 2-CSPs and directed Steiner network. In *ITCS*, 2018. doi:10.4230/LIPICS.ITCS.2018.36.

16 Ajit A Diwan. Cycles of weight divisible by k, 2024. doi:10.48550/arXiv.2407.01198.

17 Jon Feldman and Matthias Ruhl. The directed Steiner network problem is tractable for a constant number of terminals. *SIAM J. Comput.*, 36(2), 2006. doi:10.1137/S0097539704441241.

18 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed Steiner network problems. *ACM Trans. Comput. Theory*, 15(3–4), 2023. doi:10.1145/3580376.

19 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10, 1980. doi:10.1016/0304-3975(80)90009-2.

20 Esther Galby, Sándor Kisfaludi-Bak, Dániel Marx, and Roohani Sharma. Subexponential parameterized directed Steiner network problems on planar graphs: A complete classification. In *ICALP*, 2024. doi:10.4230/LIPICS.ICALP.2024.67.

21 Jiong Guo, Rolf Niedermeier, and Ondrej Suchý. Parameterized complexity of arc-weighted directed Steiner problems. *SIAM J. Discret. Math.*, 25(2), 2011. doi:10.1137/100794560.

22 Xiao Hu and Stavros Sintos. Finding smallest witnesses for conjunctive queries. In *ICDT*, 2024. doi:10.4230/LIPICS.ICDT.2024.24.

23 Alpár Jüttner, Csaba Király, Lydia Mirabel Mendoza-Cadena, Gyula Pap, Ildikó Schlotter, and Yutaro Yamaguchi. Shortest odd paths in undirected graphs with conservative weight functions. *Discrete Appl. Math.*, 357, 2024. doi:10.1016/j.dam.2024.05.044.

24 Naonori Kakimura and Ken-ichi Kawarabayashi. Packing cycles through prescribed vertices under modularity constraints. *Adv. in Appl. Math.*, 49(2), 2012. doi:10.1016/j.aam.2012.03.002.

25 Ken-ichi Kawarabayashi, Stephan Kreutzer, O-joung Kwon, and Qiqin Xie. A half-integral Erdos-Posa theorem for directed odd cycles. In *SODA*, 2023. doi:10.1137/1.9781611977554.ch118.

26 Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4), 1984. doi:10.1002/NET.3230140403.

27 Anna Lubiw. A note on odd/even cycles. *Discret. Appl. Math.*, 22(1), 1988. doi:10.1016/0166-218X(88)90125-4.

28 Wim Martens, Matthias Niewerth, and Tina Popp. A trichotomy for regular trail queries. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:20)2023.

29 William McCuaig. Pólya's permanent problem. *Electron. J. Comb.*, 11(1), 2004. doi:10.37236/1832.

30 Zhengjie Miao, Sudeepa Roy, and Jun Yang. Explaining wrong queries using small examples. In *SIGMOD*, 2019. doi:10.1145/3299869.3319866.

31 Burkhard Monien. The complexity of determining a shortest cycle of even length. *Computing*, 31(4), 1983. doi:10.1007/BF02251238.

32 Zhivko Prodanov Nedev. Finding an even simple path in a directed planar graph. *SIAM J. Comput.*, 29(2), 1999. doi:10.1137/S0097539797330343.

33 Mehdy Roayaei and Mohammadreza Razzazi. Parameterized complexity of directed Steiner network with respect to shared vertices and arcs. *Int. J. Found. Comput. Sci.*, 29(7), 2018. doi:10.1142/S0129054118500302.

34 Neil Robertson, P. D. Seymour, and Robin Thomas. Permanents, Pfaffian orientations, and even directed circuits. *Ann. of Math. (2)*, 150(3), 1999. doi:10.2307/121059.

35 Ildikó Schlotter and András Sebő. Odd paths, cycles and t-joins: Connections and algorithms, 2022. arXiv:2211.12862.

36 Paul Seymour and Carsten Thomassen. Characterization of even directed graphs. *J. Combin. Theory Ser. B*, 42(1), 1987. doi:10.1016/0095-8956(87)90061-X.

37 Carsten Thomassen. Even cycles in directed graphs. *Eur. J. Comb.*, 6(1), 1985. doi:10.1016/S0195-6698(85)80025-1.

38 Carsten Thomassen. The even cycle problem for planar digraphs. *J. Algorithms*, 15(1), 1993. doi:10.1006/jagm.1993.1030.

39 Vijay V. Vazirani and Mihalis Yannakakis. Pfaffian orientations, 0-1 permanents, and even cycles in directed graphs. *Discret. Appl. Math.*, 25(1-2), 1989. doi:10.1016/0166-218X(89)90053-X.

40 Paul Wollan. Packing cycles with modularity constraints. *Combinatorica*, 31(1), 2011. doi:10.1007/s00493-011-2551-5.

41 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discrete Math.*, 10(2), 1997. doi:10.1137/S0895480194274133.