How Many Lines to Paint the City: Exact Edge-Cover in Temporal Graphs

Argyrios Deligkas[1], Michelle Döring[2], Eduard Eiben[1], Tiger-Lily Goldsmith[1], George Skretas[2], Georg Tennigkeit[2]

[1]Royal Holloway, University of London, Egham, United Kingdom
[2]Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
argyrios.deligkas@rhul.ac.uk, michelle.doering@hpi.de, eduard.eiben@rhul.ac.uk, tigerlily.goldsmith@gmail.com, georgios.skretas@hpi.de, georg.tennigkeit@hpi.de

Abstract

Logistics and transportation networks require a large amount of resources to realise necessary connections between locations and minimizing these resources is a vital aspect of planning research. Since such networks have dynamic connections that are only available at specific times, intricate models are needed to portray them accurately. In this paper, we study the problem of minimizing the number of resources needed to realise a dynamic network, using the temporal graphs model. In a temporal graph, edges appear at specific points in time. Given a temporal graph and a natural number k, we ask whether we can cover every temporal edge exactly once using at most k temporal journeys; in a temporal journey consecutive edges have to adhere to the order of time. We conduct a thorough investigation of the complexity of the problem with respect to four dimensions: (a) whether the type of the temporal journey is a walk, a trail, or a path; (b) whether the chronological order of edges in the journey is strict or non-strict; (c) whether the temporal graph is directed or undirected; (d) whether the start and end points of each journey are given. We almost completely resolve the complexity of these problems and provide dichotomies for each of them with respect to k.

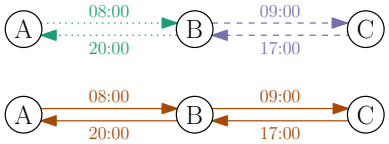
Extended version — https://arxiv.org/abs/2408.17107

1 Introduction

Networks are designed to enable the transportation of various "entities", ranging from people and physical goods to data and information. Achieving efficient transportation requires the deployment of vehicles and connections, which most often are of limited quantity. Consequently, optimizing the use of these resources is a natural challenge in network planning and design. Take, for example, a train company that decided on optimal train connections to meet public demand. Next, the company has to decide on the number of trains to be deployed on a daily schedule to realize these connections effectively in order to maximize profit. This highlights the critical issue of how to allocate limited resources efficiently to meet operational goals while balancing cost and demand.

This issue is even more apparent in networks operated by multiple companies. In such scenarios, each company independently sets its own connection times between the depots based on various and individual criteria like public service needs, profits, and resource restrictions. Their main objective is to optimally ensure their connections at the designated time. However, such decentralized planning often leads to suboptimal use of resources. This issue has been identified by the Supply Chain Management and Logistics industries, which have observed that "*while logistics improvements may be difficult for each individual customer, they can be realized with collaborative operation among multiple customers*"[1].

To illustrate, consider a scenario where company X wants to operate a line from A to B at 08:00, and from B to A at 20:00 and each connection takes one hour. Company Y wants to operate a line from B to C at 09:00, and from C to B at 17:00, also taking one hour each. Independently, both companies require one vehicle each to operate their respective lines. However, if they collaborated, they could share a vehicle for both lines with a single *journey*: A to B at 08:00, B to C at 09:00, C to B at 17:00, and B to A at 20:00.



Optimizing network resources is important across various fields, each presenting unique complexities. This poses a dilemma: should we create a specific model for each scenario, or develop a generalized, possibly simplified model to study many scenarios at once? Applied research typically chooses the former, tailoring models to individual applications, while theoretical sciences favor the latter, aiming for generalized models with broad applicability.

In this paper, we follow the second route and provide a general model that is simple both, yet realistic enough to represent multiple scenarios. We want to study the problem of finding the minimum number of resources needed to realise the connections of a network. To account for the time-specific connections of the networks, we use *temporal graphs* (Kempe, Kleinberg, and Kumar 2002; Michail 2016). A temporal graph consists of a set of vertices, a set of edges and a labeling function indicating at which time steps

[1]https://www.logisteed.com/en/3pl/joint/

Complexity		Paths	Trails	Walks	Walks with fixed terminals
directed edges	strict	NP-c ($k \geq 3$) (Thm. 4.1), paraNP-hard	NP-c ($k \geq 3$) (Cor. 4.3), paraNP-hard	P (Thm. 5.2)	P (Thm. 5.2)
	non-strict			NP-c (Thm. 5.3), W[2]-hard, XP (Prop. 5.1)	P (Thm. 6.1)
undirected edges	strict			NP-c (Thm. 5.4), FPT open, XP (Prop. 5.1)	$P^\star$ (Cor. 6.4)
	non-strict			NP-c (Thm. 5.4), FPT open, XP (Prop. 5.1)	P (Thm. 6.3)

Table 1: Overview of our results. All parameterized results are with respect to parameter k. The entry paraNP-hard indicates that NP-hardness holds even for constant k, implying that no XP or FPT algorithm exists unless P = NP. The entry $P^\star$ denotes the existence of a polynomial-time algorithm for special cases.

each edge is available. A trip through a temporal graph, has to respect these time constraints.

1.1 Our Contribution

Our contribution is twofold: (a) formalize the planning problem highlighted in the example above as an optimization problem on temporal graphs; (b) provide the landscape of its computational complexity for a rich variety of settings.

The Exact Edge-Cover. We study the following problem.

> *"Given a temporal graph and a natural number k, can we cover every temporal edge of the graph, exactly once, with at most k temporal journeys?"*

Motivated by the variety of real-life scenarios our model can capture, we perform a thorough study with respect to four dimensions. First, the type of journey, which can be a *temporal path*, *trail*, or *walk*. This distinction is important as, for example, trains usually should not revisit a station in a single trip, which can only be modeled by a temporal path. Second, the temporal nature of the journey, which can be *strict*, or *non-strict*. In a strict journey, at most one temporal edge can be used per time step, while non-strict journeys can use multiple edges instantly. Third, the type of the graph, which can be either *directed* or *undirected*. Physical networks are usually directed, whereas information networks allow for bidirected connections. Fourth, having *fixed* or *unfixed terminals* for the journeys, i.e., whether the starting and ending points of the journeys are given or not. This captures that depots might be available only at specific locations in the network. A summary of our technical results is depicted at Table 1.

Our Technical Results. The first part of our technical results focuses on the case where the terminals are not fixed. As a warm-up, we observe that when $k = 1$, the problem resembles Eulerian journeys, and provide a linear time algorithm for all our settings. We extend our positive results to $k = 2$, where we show that *almost* all versions of our problem can be reduced to a 2-SAT problem, enabling a polynomial-time solution. However, this approach fails for non-strict trails, in which the problem is NP-complete.

For $k \geq 3$, positive results become scarce. In Section 4, we study paths and trails, proving intractability for all versions of the problem for any constant k. Furthermore, we show that no version can admit an α-approximation for any constant α unless P = NP. On the other hand, walks (Section 5) offer some positive results. We observe that when the number of walks k is constant, all versions of our problem can be solved efficiently. For arbitrary k, we additionally derive an efficient algorithm for the case of strict walks on directed graphs. Unfortunately, we complement the above by showing that every other version is NP-complete.

The second part of our technical results (Section 6) focuses on fixed terminals. Here, we are given a multiset of start points and a multiset of end points, where the multiplicity of a start (or end) point indicates the number of journeys that must start (or end) at that point. We begin by observing for paths and trails our hardness reductions hold even with fixed terminals. For walks, the situation is different. We provide a dynamic program for non-strict walks on directed graphs. On undirected graphs, the problem resembles a circulation flow. To capture the temporal aspect of the flow, we require a complex construction, making this the most technically involved result of the paper. The resulting algorithm solves the problem for non-strict walks. For strict walks, the algorithm works only when the labels at each vertex are distinct. Additionally, we provide a dynamic program for graphs where there are exactly k or 0 edges per time step.

1.2 Related Work

The minimum number of resources needed to realize a network has been studied in various contexts. For example, the *minimum fleet size problem* asks for the minimum number of vehicles needed to cover every trip in a network. Unlike our model, this problem is studied only on directed networks that can be modeled to contain no cycles and aims to cover every node rather than every edge (Vazifeh et al. 2018). Another example is the *rolling stock problem*, which asks for the minimum number of wagons needed to meet the passenger demand of a train network (Alfieri et al. 2006). Here, the train lines are given with a capacity demand on each connection that needs to be met. In contrast, our model is given individual connections (each with unit capacity) and aims to find the optimal train lines.

The study of edge covering problems on static graphs was initiated by Erdös, Goodman, and Pósa (1966) and László (1968). This led to an extensive line of work studying the

minimum number of gadgets needed to cover the edges of a graph. The usual gadgets considered are cliques, cycles and paths. As far as paths are concerned, most research focuses on existential upper bounds. Chung (1980) conjectured that a graph can be covered with at most $\lceil \frac{n}{2} \rceil$ paths, which was settled by Fan (2002). For an extensive overview of the covering problem on static graphs, we refer to (Schwartz 2022).

For temporal graphs, there has been a lot of research on reachability problems adjacent to ours (Zschoche 2023; Bilò et al. 2022; Bilò et al. 2022; Enright, Meeks, and Skerman 2021; Bentert et al. 2020; Bui-Xuan, Ferreira, and Jarry 2003; Deligkas et al. 2023). The most prominent one is the temporal exploration problem, where the goal is to compute an earliest-arrival walk visiting every vertex of the graph. It was introduced by Erlebach, Hoffmann, and Kammer (2021) and has been extensively studied since (Arrighi et al. 2023; Erlebach and Spooner 2023; Adamson et al. 2022).

Recently, Bumpus and Meeks (2022), and Marino and Silva (2022) introduced an edge variant of this problem, where the goal is to decide whether the static edges of a temporal graph can be covered by an Eulerian circuit or trail. The main difference between their work and ours is that we want to cover every temporal edge, whereas they focus on covering every static edge. Additionally, they consider covering the edges with one trail, while we study an arbitrary number and different types of journeys. We highlight that no result can be transferred between the two models.

2 Preliminaries

For $n \in \mathbb{N}$, we denote $[n] := \{1, 2, \ldots, n\}$. A *temporal graph* $\mathcal{G} := \langle G, \mathcal{E} \rangle$ is defined by an *underlying* static graph $G = (V, E)$ and a sequence of edge-sets $\mathcal{E} = (E_1, E_2, \ldots, E_{t_{\max}})$ with $E = E_1 \cup E_2 \cup \cdots \cup E_{t_{max}}$. We refer to E as the set of *static* edges of $\mathcal{G}$ and to $\mathcal{E}$ as the set of *temporal* edges. The *lifetime* of $\mathcal{G}$ is $t_{\max}$. An edge $e \in E$ *has label* i if $e \in E_i$. We denote directed edges with $((u, v), t)$ and undirected edges with $(\{u, v\}, t)$. The *edgestream representation* E_S of $\mathcal{G}$ is an ordered list of all temporal edges, sorted by increasing time label. By a *snapshot* G_t of $\mathcal{G}$ we refer to the subgraph (V, E_t) containing only the edges available at time $t \in [t_{\max}]$. The temporal degree $\delta(v)$ of a vertex v denotes the number of temporal edges adjacent to v. On directed graphs, we further define $\delta^{in}(v)$ and $\delta^{out}(v)$ as the number of incoming and outgoing temporal edges of v, respectively.

Temporal Journeys. A *temporal journey* J in $\mathcal{G} = \langle (V, E), \mathcal{E} \rangle$ is a sequence of adjacent temporal edges that respect time, connecting a *start-* with an *end-terminal*. Formally, if $J = ((e_1, t_1), \ldots, (e_\ell, t_\ell))$ is a temporal journey, then for every $i \in [\ell - 1]$ holds that: $e_i \in E_{t_i}$; e_i is adjacent to e_{i+1}; and $t_i \leq t_{i+1}$. If we require that $t_i < t_{i+1}$ for all $i \in [\ell - 1]$, we call the journey *strict*, otherwise we call it *non-strict*. For a temporal journey J, we denote by $J[t_1, t_2]$ the *sub-journey* between t_1 and t_2, i.e., the sub-sequence of temporal edges (e_i, t_i) of J, such that $t_i \in [t_1, t_2]$. If the temporal graph is directed, then we have *directed* journeys, otherwise, we have *undirected* journeys. We focus on three types of temporal journeys:

- *temporal walks*, with no extra constraints;
- *temporal trails*, where no static edge is visited more than once by the journey;
- *temporal paths*, where no vertex is visited more than once by the journey.

Parameterized complexity. We refer to the standard books for a basic overview of parameterized complexity theory (Cygan et al. 2015; Downey and Fellows 2013). At a high level, parameterized complexity studies the complexity of a problem with respect to its input size, n, and the size of a parameter k. A problem is *fixed parameter tractable* by k, if it can be solved in time $f(k) \cdot \texttt{poly}(n)$, where f is a computable function. Showing that a problem is $\texttt{W}[2]$-hard rules out the existence of a fixed-parameter algorithm under the well-established assumption that $\texttt{W}[2] \neq \texttt{FPT}$.

3 Exact Edge-Cover

Exact edge-covers by paths in static graphs have been studied by Donald (1980); Pyber (1991, 1996), where this structure is called an *edge-disjoint path cover*. We define the temporal generalization on different types of journeys.

Definition 3.1 (Temporal Exact Edge-Cover by Temporal Journeys)**.** *Let $\mathcal{G} = \langle (V, E), \mathcal{E} \rangle$ be a temporal graph and $\Phi \in \{$path, walk, trail$\}$. A collection $\mathcal{J} = \{J_1, \ldots, J_k\}$ of journeys of type Φ, is called a* temporal exact edge-cover (eec) by Φ *for $\mathcal{G}$ if for all temporal edges $e \in \mathcal{E}$ there is exactly one $J_i \in \mathcal{J}$ with $e \in J_i$.*

We study the problem of finding temporal exact edge-covers of size k by the (non-)strict version of each journey type in (un)directed temporal graphs. For $\Phi \in \{\text{PATH}, \text{TRAIL}, \text{WALK}\}$, we define (N)S-T$\Phi$EEC as follows.

(NON-) STRICT TEMPORAL Φ EXACT EDGE-COVER	
Input:	A temporal graph $\mathcal{G}$ and $k \in \mathbb{N}$.
Question:	Does there exist a temporal exact edge-cover of (non-)strict Φ for $\mathcal{G}$ of size k?

Checking whether a given collection of journeys is an exact edge-cover can be done in polynomial time by verifying that each temporal edge appears exactly once and that all journeys are of type Φ. Thus, (N)S-TΦEEC is in the class NP.

3.1 One Journey – Extending Eulerian Journeys

If $k = 1$, all versions of this problem can be solved in polynomial time. While our algorithm follows the same strategy for every journey type, the approach needs to differentiate depending on whether the journeys are strict or not.

Proposition 3.2 ($\star$)**.** *For $\Phi \in \{\text{PATH}, \text{TRAIL}, \text{WALK}\}$, we can solve* 1-(N)S-T$\Phi$EEC *in polynomial time on both directed and undirected graphs.*

For strict journeys, we attach the edges in temporal order and check whether this forms the desired journey. For non-strict journeys, we additionally observe that a set of edges appearing at the same time has to form an Eulerian journey which can be found in polynomial time (Fleischner 1990).

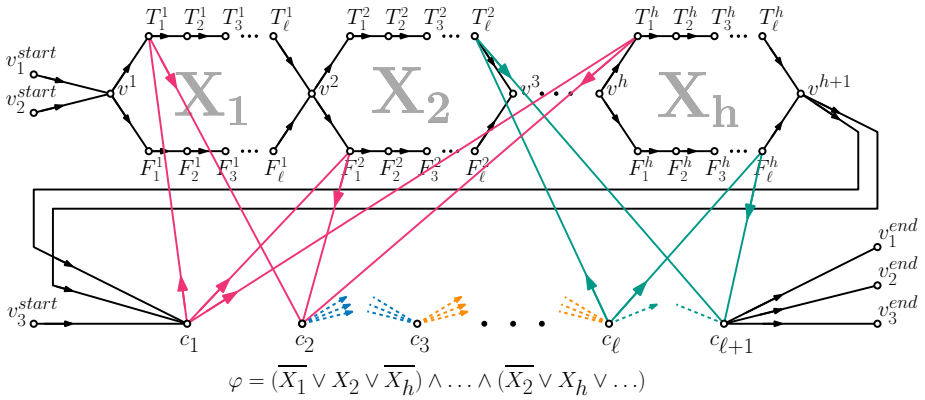


Figure 1: Illustration of the construction for the NP-hardness reduction in Thm. 4.1 for $k = 3$. The two edges from v^{h+1} to c_1 connecting the assignment-gadget on the top and satisfaction-gadget on the bottom are drawn as two separate edges. With this, every drawn edge has exactly one time label increasing with the direction of the edges.

3.2 Two Journeys – 2-SAT Approach

While 2-(N)S-TPATHEEC, 2-S-TTRAILEEC and 2-(N)S-TWALKEEC are solvable in polynomial time for both directed and undirected graphs, we show that 2-NS-TTRAILEEC is NP-hard on directed and undirected graphs.

Towards the positive results, we utilize a polynomial-time algorithm for 2-SAT.

Intuitively, as paths cannot revisit vertices we know that on directed temporal graphs every vertex can have at most two incoming and two outgoing edges, each covered by exactly one path. Now, if the temporal edges incident to a vertex are alternatingly incoming and outgoing [*in-out-in-out*], we **have to** match the first *in*-edge with the first *out*-edge. However, when both incoming edges appear before the outgoing edges [*in-in-out-out*], we **can choose** which *in* is matched with which *out*. This can be formalised via a 2-SAT formula with one variable for each vertex with *in-in-out-out* edge appearance and one clause for each vertex with *in-out-in-out* edge appearance. Setting a variable to True corresponds to the first path taking the earliest *out*-edge at that vertex and setting it to False corresponds to the first path taking the latest *out*-edge.

Theorem 3.3 ($\star$)**.** 2-(N)S-TPATHEEC, 2-S-TTRAILEEC *and* 2-(N)S-TWALKEEC *can be solved in polynomial time on directed and undirected temporal graphs.*

To show hardness for non-strict trails, we adjust the proof of Marino and Silva (2022, Theorem 10). They show that EULERIAN TRAIL – finding a single temporal trail visiting all **static** edges of a temporal graph – is NP-hard.

Theorem 3.4 ($\star$)**.** 2-NS-TTRAILEEC *is* NP*-complete on both directed and undirected temporal graphs.*

Having completely resolved our problem for one and two journeys, we now move on to the computational complexity of the general problem with k at least 3. As we will see, for larger k there is a divergence in the complexity of the problem for the three journey types, which, interestingly, is notably different from the behavior observed for $k = 2$.

4 Paths and Trails – Computational and Approximation Hardness for $k \geq 3$

We prove that all versions of our problem are hard for both paths and trails, starting with intractability for paths, then augmenting the construction to get a strong inapproximability bound, and finally extending the results to trails.

Theorem 4.1 ($\star$)**.** (N)S-TPATHEEC *is* NP*-complete on directed and undirected temporal graphs, for every* $k \geq 3$*.*

Proof sketch. All four constructions — (non-)strict paths on (un)directed graphs — follow the same idea, inspired by the SAT reduction in Klobas et al. (2023, Theorem 2). We outline the case of non-strict paths in directed graphs, which is the most concise construction, requiring just two time steps.

We reduce from NAE(k)SAT, which is known to be NP-complete (Schaefer 1978). The input is a formula φ over ℓ variables in conjunctive normal form where each clause contains exactly k literals. The goal is to check whether the formula is satisfiable such that each clause has at least one literal that is True and one that is False.

We construct a temporal graph $\mathcal{G}$ with an exact edge-cover of size k if and only if φ is satisfiable; otherwise, the cover will require size $k + 1$. See Figure 1 for an illustration.

For each variable X_i, we construct a *variable-gadget* with vertices $\{v^i, v^{i+1}\} \cup \{T^i_j, F^i_j \colon j \in [\ell]\}$ and edges $\mathcal{E}^{x_i}_T = \{(v^i, T^i_1, 1)\} \cup \{(T^i_j, T^i_{j+1}, 1) \colon j \in [\ell-1]\} \cup \{(T^i_\ell, v^{i+1}, 1)\}$ and $\mathcal{E}^{x_i}_F = \{(v^i, F^i_1, 1)\} \cup \{(F^i_j, F^i_{j+1}, 1) \colon j \in [\ell - 1]\} \cup \{(F^i_\ell, v^{i+1}, 1)\}$. All variable-gadgets together form the *assignment-gadget*.

Let C_i be a clause containing k literals. For each literal L_α in C_i, let X_α be the corresponding variable. We construct a *clause-gadget* with vertices $\{c_i, c_{i+1}\}$ and edges $\{(c_i, F^\alpha_i, 2), (F^\alpha_i, c_{i+1}, 2)\}$ for every positive literal $L_\alpha \equiv X_\alpha$ and edges $\{(c_i, T^\alpha_i, 2), (T^\alpha_i, c_{i+1}, 2)\}$ for every negative literal $L_\alpha \equiv \overline{X_\alpha}$. All clause-gadgets together form the *satisfaction-gadget*. Lastly, we connect the assignment-gadget and the satisfaction-gadget by adding $\{(v^{h+1}, c_1, 1), (v^{h+1}, c_1, 2)\}$.

Intuitively, the *assignment-gadget* is constructed out of two paths and the *satisfaction-gadget* is constructed out of k paths. If φ is satisfiable and the two paths in the assignment-gadget are chosen correctly, then they will be extendable to cover two of the k paths of the satisfaction-gadget. The remaining paths can be covered for free. If φ is not satisfiable, we will need to create an additional path no matter how the two paths in the assignment-gadget are chosen. □

Hardness of Approximation. Further extending this construction, we show that there cannot be a polynomial time algorithm that computes a constant α-approximation of (N)S-TPATHEEC for $k \geq 3$, unless $\mathtt{P} = \mathtt{NP}$.

We do so by connecting a sufficient number of copies of the constructed $\mathcal{G}$ via edges from the end-vertices of the j^{th} copy $v_{j,i}^{\text{end}}$ to the start-vertices of the $j+1^{\text{th}}$ copy $v_{j+1,i}^{\text{start}}$, as illustrated in Figure 2 for NAE(3)SAT. Each copy increases the number of paths – necessary to cover all edges if φ is unsatisfiable – by one. If the number of copies is large enough, we would be able to correctly decide whether the NAE(k)SAT formula φ is satisfiable given an α-approximation of (N)S-TPATHEEC.

Theorem 4.2 ($\star$)**.** *For all $\alpha > 1$, there is no polynomial time α-approximation algorithm for* (N)S-TPATHEEC *on directed and undirected temporal graphs unless* $\mathtt{P} = \mathtt{NP}$.

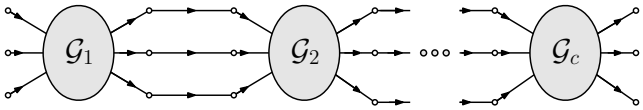


Figure 2: Illustration of Thm. 4.2 with c copies of $\mathcal{G}$: $\mathcal{G}_1, \ldots, \mathcal{G}_c$. For every $\mathcal{G}_j$, each end-vertex is connected to the corresponding start-vertex of $\mathcal{G}_{j+1}$.

Extending Paths to Trails. It is straightforward to adjust both constructions such that covering clause-edges forces revisiting edges instead of revisiting vertices in the variable-gadget. This implies the following.

Corollary 4.3. (N)S-TTRAILEEC *are* NP*-complete on directed and undirected temporal graphs.*

Corollary 4.4. *For all $\alpha > 1$, there is no polynomial time α-approximation algorithm for* (N)S-TTRAILEEC *on directed and undirected temporal graphs, unless* $\mathtt{P} = \mathtt{NP}$.

5 Walks – One Polynomial-Time Algorithm and Three Hardness Reductions for $k \geq 3$

When constructing exact edge-covers using temporal walks, we find a striking contrast to paths and trails: the problem remains polynomial-time solvable for any constant k.

Proposition 5.1 ($\star$)**.** (N)S-TWALKEEC *can be computed in time $\mathcal{O}(n^{\mathcal{O}(k)})$ which is polynomial for any constant k.*

This is due to the nature of temporal walks in exact edge-covers: they are time-respecting, connected journeys that cover each temporal edge exactly once. For a constant number k, we can brute-force the start-terminals and, at each time step, keep track of all possible positions of the k walks – there are at most n^k such positions.

For an arbitrary size of the eec, the computational complexity varies depending on the variant of the problem. For strict walks on directed graphs, we can optimize the approach from Prop. 5.1, while the problem becomes NP-complete for non-strict walks on directed graphs and for both strict and non-strict walks on undirected graphs.

5.1 Strict Walks in Directed Graphs – Two Polynomial Time Approaches

For strict walks on directed graphs, the graph transformation introduced by Wu et al. (2014) forms a directed acyclic graph. We can adjust it so that a temporal walk eec in $\mathcal{G}$ corresponds to an exact path cover (covering every vertex exactly once), which is computable in polynomial time.

A faster (linear time) algorithm can be achieved by using the endpoint-tracking approach from Prop. 5.1. Since the walks are strict, two edges at the same time step need to be taken by two different walks, and since the graph is directed, the possible extensions are clearly defined. If an edge cannot extend an existing walk, a new walk starting with that edge has to be introduced. Furthermore, the starting points of the walks at the first time step are uniquely defined. This way, at each time step, there is exactly one possible position for walks of an exact edge-cover.

Theorem 5.2 ($\star$)**.** S-TWALKEEC *on directed temporal graphs can be solved in $\mathcal{O}(|\mathcal{E}|)$ if the edgestream is given.*

5.2 Non-Strict Walks or Undirected Graphs – Hard to Compute Efficiently

We proceed with the complexity of the other three variants of walk exact edge-covers. For NS-TWALKEEC in directed graphs we prove W[2]-hardness and in undirected graphs we show that both (N)S-TWALKEEC are NP-hard.

Towards the first result, observe that non-strict walks can traverse directed cycles appearing at one time step, unlike strict walks. Given such a directed cycle in which every edge has the same time label, any non-strict walk covering these edges must either omit some edges of the cycle or return to the vertex is started at. We exploit this behavior of returning to a chosen vertex to cover a cycle, for a reduction from k-HITTING SET to NS-TWALKEEC on directed graphs.

Theorem 5.3 ($\star$)**.** NS-TWALKEEC *on directed temporal graphs is* NP*-complete and* W[2]*-hard when parameterized by the number of walks in the exact edge-cover.*

Moving on to undirected graphs, a walk must choose the direction of its starting edge. Using this choice as an assignment, we can construct a reduction from 3-SAT. The strict and non-strict variants require slightly different constructions but share the same idea: Each variable corresponds to an undirected edge at time step 1. Traversing that edge from left to right sets the variable to True, while traversing it from right to left, sets the variable to False.

Theorem 5.4 ($\star$)**.** (N)S-TWALKEEC *on undirected temporal graphs is* NP*-complete.*

6 Walks with Fixed Terminals – Polynomial Time Algorithms

In this section, we focus on exact edge-covers with fixed terminals. Observe that this also fixes the number of journeys (one journey per terminal pair), but we emphasize that this is **not** the same as having a **constant** number of journeys.

It is easy to see that our hardness constructions for paths and trails inherently fix the terminals, thereby translating directly. In contrast, for walks, we *chose* the terminals to *simulate* a Boolean assignment. This is not incidental, as having fixed terminals makes finding an eec with walks tractable.

6.1 Directed Graphs

Strict walks in directed graphs can be solved without fixing terminals using a dynamic program computing possible endpoints with increasing time. For non-strict walks with fixed terminals, this program can be adapted, as we avoid the issue of choosing the vertices to start on (in particular on a cycle). So, we initiate one walk per start-terminal and extend those at each time step, possibly by multiple edges.

Theorem 6.1 ($\star$)**.** NS-TWALKEEC *on directed temporal graphs can be solved in polynomial time if the start-terminals along with their multiplicity are part of the input.*

6.2 Undirected Graphs

In undirected graphs, even with terminals, we must choose the direction of each temporal edge connecting two walks. This resembles the problem of finding a circulation flow.

For non-strict walks, this can be simulated in a static mixed multigraph, similar to the static expansion by Kostakos (2009), and solved via SWALKEEC with fixed terminals in polynomial time using a flow algorithm.

Theorem 6.2 ($\star$)**.** SWALKEEC *with fixed terminals on static mixed multigraphs (containing directed and undirected multiedges) can be computed in polynomial time.*

Let us now state the main theorem of this section.

Theorem 6.3 ($\star$)**.** NS-TWALKEEC *on undirected temporal graphs can be computed in polynomial time if the start-terminals along with their multiplicity are part of the input.*

Proof sketch. Given a temporal graph $\mathcal{G}$ and a set of terminals, we construct a mixed static multigraph $\mathcal{S}$ with directed and undirected multiedges, and the same terminal sets. The construction will ensure that there is a temporal walk eec in $\mathcal{G}$ if and only if there is a static walk eec in $\mathcal{S}$. The claim then follows from Thm. 6.2, which states that a static walk eec with fixed terminals can be computed in polynomial time.

Terminals. For each start- and end-terminal at vertex v, we add an additional vertex with a single edge to v with a unique time label that is smaller, respectively larger, than all labels in $\mathcal{G}$. All other time labels are increased accordingly. Note that a vertex in $\mathcal{G}$ can be a start and end of multiple walks, while at the constructed terminals there is exactly one walk starting or ending. From now, the term "vertices" will refer specifically to non-terminal vertices.

Construction. For each vertex v, we create a v-gadget simulating the connections of v at the each time step. Let $T(v)$ denote the multiset of labels on edges incident to v. For each time step $t \in T(v)$, create a vertex v_t. We say these vertices are placed from *left to right* by increasing time label. Now, we connect each v_t with its right neighbor v_{t+1} to simulate the maximum amount of walks going through v time t: Let $\delta_{\leq t}(v)$ and $\delta_{\geq t+1}(v)$ be the number of temporal edges incident to v with time label $\leq t$ and $\geq t+1$, respectively. The maximum amount of walks passing through has to enter before t and leave after $t+1$. Thus, we create $\min(\delta_{\leq t}(v), \delta_{\geq t+1}(v))$ edges between v_t and v_{t+1}. To allow some of these walks to enter before t and also exit before t, half of these edges rounded up are directed from v_t to v_{t+1}, while the others are undirected. A directed-undirected edge pair can then form a cycle attachable to any walk at v_t.

For each edge $(\{v, u\}, t)$ in $\mathcal{G}$, we add $\{v_t, u_t\}$ to $\mathcal{S}$, and for each terminal in $\mathcal{G}$ connected to vertex v, we add a terminal in $\mathcal{S}$ with an edge to v_0. See Figure 3 for an illustration.

Temporal eec in $\mathcal{G}$ to static eec in $\mathcal{S}$. For every temporal walk W, we construct a static walk with the same terminals: If W sequentially traverses the edges $(\{a, v\}, t)$ and $(\{v, b\}, t')$, the corresponding static walk enters v-gadget via $\{a_t, v_t\}$, travels $v_t \rightsquigarrow v_{t'}$ using directed edges (or undirected, if no directed edges remain), and exits v-gadget via $\{v_{t'}, b_{t'}\}$. Observe that there is a sufficient amount of edges between neighbouring vertices v_t and v_{t+1} so that every walk crossing these vertices can use a different edge.

After translating the temporal walks into static walks this way, some edges between some v_t and its right neighbor v_{t+1} may remain unused. We show that this number is always even and the edges can be attached to existing walks: The number of edges between v_t and v_{t+1} equals the maximum walks possibly present in v right at time t, which is achieved when every undirected edge before t is directed towards v. If there is one less walk present, then this walk had to enter at $\{a_i, v_i\}$ and exit at $\{v_j, b_j\}$, both before t. Therefore, the walk which would have entered at $\{v_j, b_j\}$ is also missing. Since directed edges are used before undirected edges, at least half of the remaining edges are undirected. These edges form a cycle going back and forth between v_t and v_{t+1}, which can be inserted into any walk visiting v_t.

Static eec in $\mathcal{S}$ to temporal eec in $\mathcal{G}$. A static eec in $\mathcal{S}$ cannot be directly translated into a temporal eec in $\mathcal{G}$. For a temporal eec, each walk must exit a vertex via a time label greater than the one by which it entered the vertex. In $\mathcal{S}$, this corresponds to every walk exiting a gadget to the right of its entry point, which is not guaranteed due to the undirected edges. Refer to Figure 4 (top) for an illustration of a walk violating this property. However, we can modify any static eec on $\mathcal{S}$ to be time-respecting. The general idea is that any walk going "back in time" has to overlap with at least one walk with which it can swap suffixes. Refer to Figure 4 (bottom) for an illustration of such a swap, working as follows:

Let W_1 enter at v_{t_j} and exit at v_{t_i} with $t_i < t_j$. Then it used an undirected edge $\{v_{t_{i+1}}, v_{t_i}\}$. By construction, there is at least one directed edge $(v_{t_i}, v_{t_{i+1}})$ taken by some walk W_2 that entered to the left of v_{t_i} and exits to the right. Other-

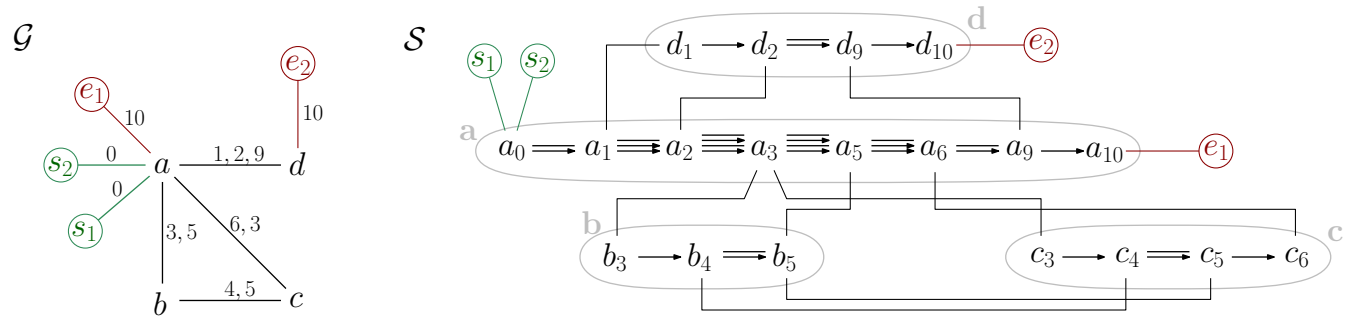


Figure 3: Illustration of how a temporal graph $\mathcal{G}$ (left) is translated into a static graph $\mathcal{S}$ (right) for Thm. 6.3, with start-terminals $V_S = \{s_1, s_2\}$ and end-terminals $V_E = \{e_1, e_2\}$. Vertices are replaced with gadgets based on their temporal degree.

wise, by the parity argument for the edges, at least one edge between v_{t_i} and $v_{t_{i+1}}$ would not be covered. We now swap the suffixes of W_1 and W_2 and therefore reduce the number of backwards taken edges by at least one. This is continued until no edges are taken backwards.

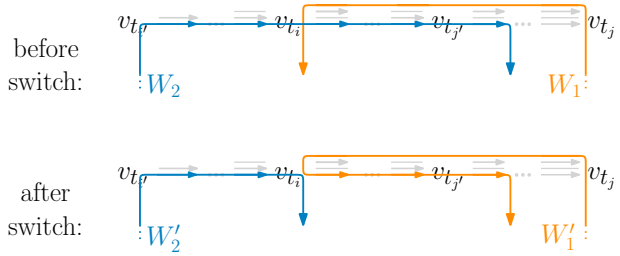


Figure 4: Suffix-switch between two walks W_1 and W_2.

Finally, we can translate the time-respecting static eec into a temporal eec by contracting the gadgets back into vertices. □

For strict walks, this construction fails as it allows exact edge-covers which are valid for non-strict walks but invalid for strict walks and, unfortunately, it cannot be directly modified to handle such cases. However, on temporal graphs where adjacent edges are never present at the same time, every strict walk is inherently non-strict, allowing the algorithm to work. Such graphs are commonly referred to as proper temporal graphs (Casteigts, Corsini, and Sarkar 2024; Christiann, Sanlaville, and Schoeters 2024).

Corollary 6.4. S-TWALKEEC *with fixed terminals on proper undirected temporal graphs, i. e.,* $|N_t(v)| \in \{0, 1\}$ *for all* $t \in [t_{\max}]$*, can be computed in polynomial time, where* $N_t(v)$ *is the multiset of neighbours of* v *at time step* t*.*

Lastly, we cover the other extreme of undirected temporal graphs with exactly k or 0 edges per time step. Here, once again a dynamic program tracking possible walk endpoints is able to compute the eecs. This is possible because each walk must cover one edge per time step.

Theorem 6.5 ($\star$)**.** S-TWALKEEC *with fixed terminals* V_S*,* V_E *on undirected temporal graphs where* $|E_t| \in \{0, k\}$ *for all* $t \in [t_{\max}]$*, can be computed in time* $\mathcal{O}(2^{|V_S|} \cdot |V_S| \cdot t_{\max})$*.*

For temporal graphs with an arbitrary number of edges per time step, the complexity of S-TWALKEEC with fixed terminals remains an open question.

7 Conclusion

We introduce exact edge-covers (eecs) on temporal graphs with three journey variants: paths, trails and walks, aiming to capture the subclass of temporal graphs that model real-world transit networks. We provide a comprehensive analysis of the computational complexity of the eec problem and observe fundamental differences between covering with paths/trails and covering with walks.

Several open problems remain, motivated by both theory and practical considerations. On the theoretical side, it would be interesting to explore whether walk eecs can be approximated and to resolve the open case for strict walks on undirected graphs with fixed terminals. Additionally, for paths, we observed that eecs are efficiently computable on DAGs and bidirected trees. So, an open question is to classify the subclasses of graphs for which these generally hard problems are polynomial-time solvable. Designing efficient algorithms for paths and trails with parameters arising from real-life applications is also an important direction.

On the applied side, several variants of the model could be explored. Our focus was on the exact covering version of the problem, as it captures scenarios where a connection cannot be used by multiple vehicles at the same time, such as railroads, where only one train can occupy a track at a time. Additionally, studying the exact version implicitly minimizes the resources, like fuel, needed to satisfy the connections. However, the non-exact case is also interesting to explore. While our negative results still apply, the algorithms would need significant adjustments or entirely new approaches.

A completely different, and "complementary", direction is to consider that the underlying temporal graph is *provided* as a collection of "few" temporal journeys and study scheduling and other combinatorial problems on this class of graphs. This natural parameter which, to the best of our knowledge, has not been studied yet, defines a structured family of temporal graphs. Can this parameter allow for efficient algorithms in temporal graphs? If yes, then this will be a very positive exception in the literature.

Acknowledgments

Michelle Döring was supported by the German Federal Ministry for Education and Research (BMBF) through the project "KI Servicezentrum Berlin Brandenburg" (01IS22092). Georg Tennigkeit was supported by the HPI Research School on Data Science and Engineering. Argyrios Deligkas was supported by EPSRC Grant EP/X039862/1 "NAfANE: New Approaches for Approximate Nash Equilibria".

References

Adamson, D.; Gusev, V. V.; Malyshev, D.; and Zamaraev, V. 2022. Faster Exploration of Some Temporal Graphs. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*, 5:1–5:10.

Alfieri, A.; Groot, R.; Kroon, L.; and Schrijver, A. 2006. Efficient circulation of railway rolling stock. *Transportation Science*, 40(3): 378–391.

Arrighi, E.; Fomin, F. V.; Golovach, P. A.; and Wolf, P. 2023. Kernelizing Temporal Exploration Problems. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, 1:1–1:18.

Bentert, M.; Himmel, A.; Nichterlein, A.; and Niedermeier, R. 2020. Efficient computation of optimal temporal walks under waiting-time constraints. *Appl. Netw. Sci.*, 5(1): 73.

Bilò, D.; D'Angelo, G.; Gualà, L.; Leucci, S.; and Rossi, M. 2022. Blackout-Tolerant Temporal Spanners. In *Algorithmics of Wireless Networks (ALGOSENSORS 2022)*, 31–44.

Bilò, D.; D'Angelo, G.; Gualà, L.; Leucci, S.; and Rossi, M. 2022. Sparse Temporal Spanners with Low Stretch. In *European Symposium on Algorithms (ESA 2022)*, 19:1–19:16.

Bui-Xuan, B.; Ferreira, A.; and Jarry, A. 2003. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *Int. J. Found. Comput. Sci.*, 14(2): 267–285.

Bumpus, B. M.; and Meeks, K. 2022. Edge Exploration of Temporal Graphs. *Algorithmica*, 688–716.

Casteigts, A.; Corsini, T.; and Sarkar, W. 2024. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theoretical Computer Science*, 991: 114434.

Christiann, E.; Sanlaville, E.; and Schoeters, J. 2024. On Inefficiently Connecting Temporal Networks. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Chung, F. 1980. On the coverings of graphs. *Discrete Mathematics*, 30(2): 89–93.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer. ISBN 978-3-319-21274-6.

Deligkas, A.; Eiben, E.; Goldsmith, T.; and Skretas, G. 2023. Being an Influencer is Hard: The Complexity of Influence Maximization in Temporal Graphs with a Fixed Source. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*, 2222–2230.

Donald, A. 1980. An upper bound for the path number of a graph. *Journal of Graph Theory*, 4(2): 189–201.

Downey, R. G.; and Fellows, M. 2013. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer.

Enright, J. A.; Meeks, K.; and Skerman, F. 2021. Assigning times to minimise reachability in temporal graphs. *J. Comput. Syst. Sci.*, 115: 169–186.

Erdös, P.; Goodman, A. W.; and Pósa, L. 1966. The Representation of a Graph by Set Intersections. *Canadian Journal of Mathematics*, 18: 106–112.

Erlebach, T.; Hoffmann, M.; and Kammer, F. 2021. On temporal graph exploration. *Journal of Computer and System Sciences*, 119: 1–18.

Erlebach, T.; and Spooner, J. T. 2023. Parameterised temporal exploration problems. *Journal of Computer and System Sciences*, 135: 73–88.

Fan, G. 2002. Subgraph Coverings and Edge Switchings. *Journal of Combinatorial Theory, Series B*, 84(1): 54–83.

Fleischner, H. 1990. *Eulerian graphs and related topics*. Elsevier.

Kempe, D.; Kleinberg, J.; and Kumar, A. 2002. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4): 820–842.

Klobas, N.; Mertzios, G. B.; Molter, H.; Niedermeier, R.; and Zschoche, P. 2023. Interference-free walks in time: Temporally disjoint paths. *Autonomous Agents and Multi-Agent Systems*, 37(1): 1.

Kostakos, V. 2009. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6): 1007–1023.

László, L. 1968. On covering of graphs. *Theory of Graphs*.

Marino, A.; and Silva, A. 2022. Eulerian Walks in Temporal Graphs. *Algorithmica*, 85: 805–830.

Michail, O. 2016. An Introduction to Temporal Graphs: An Algorithmic Perspective. *Internet Mathematics*, 12(4): 239–280.

Pyber, L. 1991. Covering the edges of a graph by ... In *Sets, Graphs and Numbers, Colloquia Mathematica Societatis János Bolyai*, volume 60, 583–610.

Pyber, L. 1996. Covering the edges of a connected graph by paths. *journal of combinatorial theory, Series B*, 66(1): 152–159.

Schaefer, T. J. 1978. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, 216–226.

Schwartz, S. 2022. An overview of graph covering and partitioning. *Discrete Mathematics*, 112884.

Vazifeh, M. M.; Santi, P.; Resta, G.; Strogatz, S. H.; and Ratti, C. 2018. Addressing the minimum fleet problem in on-demand urban mobility. *Nature*, 557(7706): 534–538.

Wu, H.; Cheng, J.; Huang, S.; Ke, Y.; Lu, Y.; and Xu, Y. 2014. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9): 721–732.

Zschoche, P. 2023. Restless Temporal Path Parameterized Above Lower Bounds. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254, 55:1–55:16.