


# Recognizing and Realizing Temporal Reachability Graphs

Thomas Erlebach 

Department of Computer Science, Durham University, UK

Othon Michail 

Department of Computer Science, University of Liverpool, UK

Nils Morawietz 

LaBRI, Université de Bordeaux, Talence, France

Institute of Computer Science, Friedrich Schiller University Jena, Germany

---

## Abstract

A temporal graph  $\mathcal{G} = (G, \lambda)$  can be represented by an underlying graph  $G = (V, E)$  together with a function  $\lambda$  that assigns to each edge  $e \in E$  the set of time steps during which  $e$  is present. The reachability graph of  $\mathcal{G}$  is the directed graph  $D = (V, A)$  with  $(u, v) \in A$  if and only if there is a temporal path from  $u$  to  $v$ . We study the Reachability Graph Realizability (RGR) problem that asks whether a given directed graph  $D = (V, A)$  is the reachability graph of some temporal graph. The question can be asked for undirected or directed temporal graphs, for reachability defined via strict or non-strict temporal paths, and with or without restrictions on  $\lambda$  (simple, proper, or both). Answering an open question posed by Casteigts et al. (TCS 2024), we show that all variants of the problem are NP-complete, except for two variants that become trivial in the directed case. For undirected temporal graphs, we consider the complexity of the problem with respect to the solid graph, that is, the graph containing all edges that could potentially receive a label in any realization. We show that the RGR problem is fixed-parameter tractable for the feedback edge set number of the solid graph. As we show, the latter parameter can presumably not be replaced by smaller parameters like feedback vertex set number or treedepth, since the problem is W[2]-hard for them.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Design and analysis of algorithms

**Keywords and phrases** parameterized complexity, temporal graphs, FPT algorithm, feedback edge set, directed graph recognition

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2025.93

**Related Version** *Full Version*: <https://doi.org/10.48550/arXiv.2503.15771>

**Funding** *Nils Morawietz*: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPO-GRAL).

## 1 Introduction

Temporal graphs are graphs that change over time. The vertex set is often assumed to be fixed, and the edge set can differ from one time step to the next. The study of temporal graphs has attracted significant attention in recent years [5, 22, 23]. One way to represent a temporal graph  $\mathcal{G}$  with vertex set  $V$  is as a sequence  $(G_i)_{i \in [L]}$ , where  $G_i = (V, E_i)$  is the graph containing the edges that are present in time step  $i$ . If an edge  $e$  is present in time step  $i$ , we refer to  $(e, i)$  as a *time edge*, and call  $i$  a *time label* of edge  $e$ . A *strict temporal path* from vertex  $u$  to vertex  $v$  in  $\mathcal{G}$  is a sequence of time edges forming a  $u$ - $v$ -path whose time steps are strictly increasing. A *non-strict temporal path* is defined analogously, except that the time steps only need to be non-decreasing.



© Thomas Erlebach, Othon Michail, and Nils Morawietz;  
licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 93; pp. 93:1–93:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work, we initiate the study of designing temporal graphs whose dynamics *guarantee certain reachability requirements while preventing others*, a problem naturally arising in areas like epidemic control, transportation, and logistics. For example, in epidemic control, a central goal is to maintain a minimal degree of societal connectivity for critical services while preventing broader or specific temporal connections that could facilitate disease spread.

Given a temporal graph  $\mathcal{G}$ , its temporal reachability relation can be represented as a directed graph  $D = (V, A)$ , called the *reachability graph* of  $\mathcal{G}$ , with  $(u, v) \in A$  for  $u \neq v$  if and only if there exists a (strict or non-strict, respectively) temporal path from  $u$  to  $v$  in  $\mathcal{G}$ . We can now state our problem as that of *realizing* a given reachability graph  $D$  by finding a temporal graph  $\mathcal{G}$  whose temporal connectivity is specified by  $D$ : each arc or non-arc of  $D$  represents a temporal reachability that *must* or *must not* be realized in  $\mathcal{G}$ , respectively. In particular, we want to know which directed graphs can arise as reachability graphs of temporal graphs, and how difficult it is to determine for a given directed graph  $D$  whether there exists a temporal graph  $\mathcal{G}$  with reachability graph  $D$ . Interestingly, this formulation turns out to have been posed twice recently as an open question, by Casteigts et al. [3, Open question 5] and Döring [11]. In contrast to our work, those papers aim to understand the expressivity of various temporal graph models, using reachability graphs as a tool.

Part of our contribution is to resolve this open question, by showing that the associated decision problem is NP-complete. Actually, there are a number of variations of the question, as we may ask for an undirected or a directed temporal graph  $\mathcal{G}$ , for a temporal graph with a restricted kind of labeling (simple, proper, happy)<sup>1</sup>, and may consider reachability with respect to strict or non-strict temporal paths. We show that all these variations are NP-hard if we ask for an undirected temporal graph, and also if we ask for a directed temporal graph except for two variations (strict temporal paths with arbitrary or simple labelings) that are known to become trivial in the directed case [11]. See Table 1 for an overview of these results.

On the positive side, we present the following algorithmic results. For a given digraph  $D = (V, A)$ , we refer to  $\{u, v\}$  for  $u, v \in V$  as a *solid* edge if both  $(u, v)$  and  $(v, u)$  are in  $A$ . Let  $G = (V, E)$  be the undirected graph on  $V$  whose edge set is the set of solid edges. We refer to  $G$  as the *solid graph* of  $D$ . The structural properties of solid graphs turn out to be crucial for our problem. We show that all undirected problem variants can be solved in polynomial time if the solid graph is a tree. Furthermore, we give an FPT algorithm for the feedback edge set number of the solid graph. This parameter can presumably not be replaced by smaller parameters like feedback vertex set, treedepth, or pathwidth, since two undirected versions of our problem turn out to be W[2]-hard for these parameters.

We believe that reachability graphs of temporal graphs form an interesting class of directed graphs whose study has only recently begun. Our results on their structure and on recognizing such graphs could be of independent interest in the context of algorithmic problems in directed graphs.

**Related work.** As mentioned, our work falls into the area of temporal graph realization problems. In these problems, one is given some data about the behavior of a temporal graph, and the goal is to detect whether there actually is a temporal graph with this behavior (and to compute such a temporal graph if one exists). Klobas et al. [17] introduced the problem of deciding for a given matrix of fastest travel durations and a period  $\Delta$  whether there exists a simple temporal graph  $\mathcal{G}$  with period  $\Delta$  with the property that the duration of the fastest

---

<sup>1</sup> simple  $\triangleq$  one label per edge, proper  $\triangleq$  no two adjacent edges share a label, happy  $\triangleq$  both proper and simple

■ **Table 1** The complexity results for all variants of RGR. Red cells indicate NP-hard cases and green cells indicate cases that are trivial (and thus polynomial-time solvable).

undirected	STRICT	NON-STRICT	directed	STRICT	NON-STRICT
ANY	Theorem 14	Theorem 14		Lemma 21, [11]	Theorem 22
SIMPLE	Theorem 14	Theorem 14		Lemma 21, [11]	Theorem 22
PROPER	Theorem 14			Theorem 22	
HAPPY	Theorem 14			Theorem 22	

temporal path between any pair of nodes in  $\mathcal{G}$  is equal to the value specified in the input matrix. Erlebach et al. [13] extended the problem to a multi-label version, where each edge is allowed up to  $\ell$  time labels. Motivated by the design of transportation networks, a related problem with respect to upper-bounded travel durations was considered [19, 20, 21].

In an early example of a temporal network design problem studied by Kempe et al. [15], the goal was to *reconstruct* a temporal labeling restricted to a single label per edge, so that a designated root reaches via temporal paths all vertices in a set  $P$  while avoiding those in a set  $N$ . For multi-labeled temporal graphs, Mertzios et al. [18] studied the problem of designing a temporal graph that preserves the reachability relation or all paths of an underlying static graph while minimizing either the *temporality* (maximum number of labels per edge) or the *temporal cost* (total number of labels used). Göbel et al. [14] showed that it is NP-complete to decide whether the edges of a given undirected graph can be labeled with a single label per edge in such a way that each vertex can reach every other vertex via a strict temporal path. Notably, this problem becomes solvable in linear time when only the degree sequence of the underlying graph is given [4]. Other studies have focused on variants of minimizing edge deletions [12], vertex deletions [25], or edge delays [9] to restrict reachability, motivated, for instance, by epidemic containment strategies that limit interactions. Temporal network design is an active area of research, with further related questions explored for example in [1, 16, 7].

Casteigts et al. [3] studied the relationships between the classes of reachability graphs that arise from undirected temporal graphs if different restrictions are placed on the graph (simple, proper, happy [3]) and depending on whether strict or non-strict temporal paths are considered. They showed that reachability with respect to strict temporal paths in arbitrary temporal graphs yields the widest class of reachability graphs while reachability in happy temporal graphs yields the narrowest class. The class of reachability graphs that arise from proper temporal graphs is the same as for non-strict paths in arbitrary temporal graphs, and this class is larger than the class of reachability graphs arising from non-strict reachability in simple temporal graphs. Strict reachability in simple temporal graphs was also shown to lie between the happy case and the general strict case. Döring [11] completed the picture of a two-stranded hierarchy for undirected temporal graphs and also extended the study to directed temporal graphs. As noted earlier, Casteigts et al. [3] also posed the open question of whether there is a characterization of directed graphs that arise as reachability graphs of temporal graphs (or of some restricted subclass of temporal graphs), and how hard it is to decide whether a given directed graph is the reachability graph of some temporal graph. These questions were posed again by Döring [11]. This is in particular of interest because Casteigts et al. [6] showed that several temporal graph problems can be solved in FPT time with respect to temporal parameters defined over the reachability graph. In this paper, we resolve these open questions regarding the complexity of all directed and undirected variants.

**Organization of the paper.** In Section 2, we provide a formal problem definition and define the notions used in this work. In Section 3, we show upper and lower bounds for the required number of labels per edge in any realization and provide a single exponential algorithm for all problem variants. Afterwards, in Section 4, we analyze properties and define splitting operations based on bridge edges in the solid graph of the undirected versions of our problem. These structural insights will be mainly used in our FPT algorithm in Section 6, but also immediately let us describe a polynomial-time algorithm for instances where the solid graph is a tree in Section 4.2. In Section 5, we then provide NP-hardness results for all undirected problem versions as well as parameterized intractability results for two of them with respect to feedback vertex set number and treedepth. Afterwards, in Section 6, we provide our main algorithmic result: an FPT algorithm with respect to the feedback edge set number  $\text{fes}$  of the solid graph. This algorithm is achieved in three steps: Firstly, we apply our splitting operations of Section 4 and provide a polynomial-time reduction rule to simplify the instance at hand, such that all we need to deal with is a subset  $X^*$  of vertices of size  $\mathcal{O}(\text{fes})$  for which the remainder of the graph decomposes into edge-disjoint trees that only interact with  $X^*$  via two leaves each. We call such trees *connector trees*. Secondly, we show that we can efficiently extend the set  $X^*$  to a set  $W^*$  of size  $\mathcal{O}(\text{fes})$  such that each resulting connector tree for the set  $W^*$  is more or less independent from the remainder of the graph with respect to the interactions of temporal paths in any realization. All these preprocessing steps run in polynomial time. Afterwards, our algorithm enumerates all reasonable labelings on the edges incident with vertices of  $W^*$  and tries to extend each such labeling to a realization for  $D$ . As we show, there are only FPT many such reasonable labelings and for each such labeling, there are only FPT many possible extensions that need to be checked. Finally, in Section 7, we briefly discuss the directed version of the problem and provide NP-hardness results for all but the two trivial cases. Proofs of statements with  $(\star)$  are deferred to the full version.

## 2 Preliminaries

For definitions on parameterized complexity, the Exponential Time Hypothesis (ETH), or parameters like treedepth, we refer to the textbooks [10, 8].

For natural numbers  $i, j$  with  $i \leq j$  we write  $[i]$  for the set  $\{1, \dots, i\}$  and  $[i, j]$  for the set  $\{i, \dots, j\}$ . For an undirected graph  $G = (V, E)$ , we denote an edge between vertices  $u$  and  $v$  as  $\{u, v\}$  or  $uv$ . By  $N(v) = \{u \in V \mid uv \in E\}$  we denote the set of neighbors of  $u$ . The *degree* of  $v$  is the size of  $N(v)$ . An edge  $e \in E$  is a *bridge* or *bridge edge* in a graph  $G = (V, E)$  if deleting  $e$  increases the number of connected components. A vertex of degree 1 is a *pendant* vertex, and an edge incident with a pendant vertex is called a *pendant* edge.

We assume directed graphs have no parallel arcs and no self-loops, and we denote an arc from  $u$  to  $v$  by  $(u, v)$ . A directed graph  $D$  is a *directed acyclic graph* (DAG) if it does not contain a directed cycle. The *degree* of a vertex  $v$  is the number of arcs containing  $v$ . For an undirected graph  $G$ , the *feedback edge set number* (*feedback vertex set number*) denotes the minimum number of edges (vertices) to remove from  $G$  to obtain an acyclic graph.

A temporal graph  $\mathcal{G}$  with vertex set  $V$  and lifetime  $L$  is given by a sequence  $(G_t)_{t \in [L]}$  of  $L$  static graphs  $G_t = (V, E_t)$  referred to as *snapshots* or *layers*. The graph  $\mathcal{G}_\downarrow = (V, E_\downarrow)$  with  $E_\downarrow = \bigcup_{i \in [L]} E_i$  is called the *underlying graph* of  $\mathcal{G}$ . Alternatively,  $\mathcal{G}$  can be represented by an undirected graph  $G = (V, E)$  with  $E \supseteq \bigcup_{t \in [1, L]} E_t$  and a labeling function  $\lambda : E \rightarrow 2^{[1, L]}$  that assigns to each edge  $e$  the (possibly empty) set of time steps during which  $e$  is present, that is,  $\lambda(e) = \{t \in [1, L] \mid e \in E_t\}$ . We write  $\mathcal{G} = (G, \lambda)$  in this case. In this representation we allow  $G$  to contain extra edges in addition to the edges of the underlying graph; such

edges  $e$  satisfy  $\lambda(e) = \emptyset$ . This is useful because in the problems we consider in this paper, there is a natural choice of a graph  $G = (V, E)$ , the *solid graph* defined below, that contains all edges of the underlying graph of every realization but may contain additional edges.

We assume that each layer of a temporal graph is an undirected graph unless we explicitly refer to directed temporal graphs. If  $t \in \lambda(e)$ , we refer to  $(e, t)$  as a *time edge*. A *temporal path* from  $u$  to  $v$  in  $\mathcal{G}$  is a sequence of time edges  $((e_j, t_j))_{j \in [\ell]}$  for some  $\ell$  such that  $(e_1, e_2, \dots, e_\ell)$  is a  $u$ - $v$  path in  $G$  and  $t_1 \leq t_2 \leq \dots \leq t_\ell$  holds in the non-strict case and  $t_1 < t_2 < \dots < t_\ell$  holds in the strict case.

A temporal graph is *proper* if no two adjacent edges share a label, *simple* if every edge has a single label, and *happy* if it is both proper and simple [3]. The reachability graph  $\mathcal{R}(\mathcal{G})$  of a temporal graph  $\mathcal{G}$  with vertex set  $V$  is the directed graph  $(V, A)$  with the same vertex set  $V$  and  $(u, v) \in A$  if and only if  $u \neq v$  and  $\mathcal{G}$  contains a temporal path from  $u$  to  $v$ . Note that  $\mathcal{R}(\mathcal{G})$  depends on whether we consider strict or non-strict temporal paths and can be computed in polynomial time in both cases [2]. We are interested in the following problem:

REACHABILITY GRAPH REALIZABILITY (RGR)

**Input:** A simple directed graph  $D = (V, A)$ .

**Question:** Does there exist a temporal graph  $\mathcal{G}$  with  $\mathcal{R}(\mathcal{G}) = D$ ?

For yes-instances of RGR, we are also interested in computing a temporal graph  $\mathcal{G}$  with  $\mathcal{R}(\mathcal{G}) = D$ . We refer to such a temporal graph as a *solution* or a *realization* for  $D$ , and we typically represent it by a labeling function. With the adjacency matrix representation of  $D$  in mind, we also write  $D_{uv} = 1$  for  $(u, v) \in A$  and  $D_{uv} = 0$  for  $(u, v) \notin A$ .

We can consider RGR with respect to reachability via strict temporal paths or with respect to non-strict temporal paths. Furthermore, we can require the realization of  $D$  to be simple, proper, or happy. For proper and happy temporal graphs, strict and non-strict reachability coincide. Therefore, the distinct problem variants that we can consider are ANY STRICT RGR, ANY NON-STRICT RGR, SIMPLE STRICT RGR, SIMPLE NON-STRICT RGR, PROPER RGR, and HAPPY RGR. Finally, we write DRGR instead of RGR if we are asking for a directed temporal graph that realizes  $D$ . We sometimes write URGR if we want to make it explicit that we are asking for an undirected realization.

**The solid graph.** If  $D_{uv} = 1$  and  $D_{vu} = 1$  for some  $u \neq v$ , we say that there is a *solid edge* between  $u$  and  $v$ . If  $D_{uv} = 1$  and  $D_{vu} = 0$ , we say that there is a *dashed arc* from  $u$  to  $v$ . We use  $G = (V, E)$  to denote the graph on  $V$  whose edge set is the set of solid edges, and we refer to this graph as the *solid graph* (of  $D$ ). For URGR it is clear that only solid edges can receive labels in a realization of  $D$ , as the two endpoints of an edge that is present in at least one time step can reach each other. Bridges of the solid graph must receive labels, but a solid edge  $e$  that is not a bridge may not receive labels, as the endpoints of  $e$  could reach each other via temporal paths of length greater than one.

► **Observation 1.** Let  $D$  be an instance of ANY STRICT URGR or SIMPLE STRICT URGR. Let  $\lambda$  be a realization of  $D$  and let  $\{u, v\}$  and  $\{v, w\}$  be solid edges that both receive at least one label under  $\lambda$ . If  $D$  contains neither the arc  $(u, w)$  nor the arc  $(w, u)$ , then there is a label  $\alpha$  such that  $\lambda(\{u, v\}) = \lambda(\{v, w\}) = \{\alpha\}$ .

Note that for PROPER URGR, HAPPY URGR, and all versions of NON-STRICT URGR, no realization for  $D$  can assign labels to both of two adjacent edges  $\{u, v\}$  and  $\{v, w\}$  if  $D$  contains neither  $(u, w)$  nor  $(w, u)$ .

A path  $P = (u_0 = u, \dots, u_\ell = v)$  from  $u$  to  $v$  in  $G$  is a *dense*  $u$ - $v$ -path if there exist arcs  $(u_i, u_j) \in A$  for all  $0 \leq i < j \leq \ell$ . In each undirected realization of  $D$ , each temporal path is a dense path in  $G$ .

We say that a realization  $\lambda$  is *frugal* if there is no edge  $e$  such that the set of labels assigned to  $e$  can be replaced by a smaller set while maintaining the property that  $\lambda$  is a realization. We say that a realization  $\lambda$  is *minimal* on edge  $e$  if it is impossible to obtain another realization by replacing  $\lambda(e)$  with a proper subset. A realization is minimal if it is minimal on every edge  $e$ . Note that every frugal realization is also minimal.

### 3 Basic Observations and an Exponential Algorithm

We first provide upper and lower bounds for the number of labels per edge in realizations.

► **Lemma 2** (\*). *Let  $D$  be an instance of URGR, and let  $e$  be a solid edge of  $G$  that is not part of a triangle.*

- *For ANY STRICT URGR, in each minimal realization of  $D$ ,  $e$  receives at most two labels.*
- *For all other versions of URGR, in each minimal realization of  $D$ ,  $e$  receives at most one label.*

For general edges, we show a linear upper bound with respect to the number of vertices.

► **Lemma 3** (\*). *If a graph  $D = (V, A)$  is realizable, then each minimal realization for  $D$  assigns at most  $n = |V|$  labels per edge.*

Lemma 3 implies that all versions of URGR and DRGR under consideration are in NP. Next, we show that the bound of Lemma 3 is essentially tight for ANY STRICT URGR.

► **Theorem 4** (\*). *For ANY STRICT URGR, there is an infinite family of directed graphs  $\mathcal{B}$ , such that for every  $D = (V, A) \in \mathcal{B}$ , where  $G = (V, E)$  is the solid graph of  $D$ , (i)  $D$  is realizable and some edge  $e \in E$  receives  $\Omega(n)$  labels in every realization of  $D$ , where  $n = |V|$ , and (ii)  $G$  has a feedback vertex set of size 2 and a feedback edge set of size  $\Theta(n)$ .*

The proof of Theorem 4 is based on the construction of an instance with a solid graph that consists of a number of subgraphs, called pages, that all share a common edge, called the spine. The instance is such that all edges of each page (except the spine) must receive the same single label, and the labels increase from one page to the next. Furthermore, for every  $i$  the reachability matrix specifies that the top of the  $i$ -th page can reach the bottom of the  $(i + 2)$ -th page and of all later pages. This requirement can then only be realized by adding a label whose value lies between the labels of page  $i$  and  $i + 2$  to the spine, thus forcing the spine to have a linear number of labels.

Based on the upper bounds on labels per edge/arc from Lemma 3, one can solve all problems under consideration via dynamic programming over subsets of already labeled edges/arcs. Roughly speaking, we design a dynamic program that stores entries for each subset  $A'$  of  $A$  and each time step  $i$ , on whether there is a temporal graph with lifetime  $i$  that has reachability graph  $D' := (V, A')$ . If SIMPLE RGR or HAPPY RGR is considered, the table also needs to store the already labeled edges/arcs in the first  $i$  time steps, to ensure that none of these edges/arcs receive another label.

► **Theorem 5** (\*). *Each version of URGR and DRGR under consideration can be solved in  $2^{\mathcal{O}(|A|)} \cdot n^{\mathcal{O}(1)}$  time, where  $A$  denotes the arc set of the input graph.*



## 4 Solid Bridge Edges: Properties and Splitting Operations

In this section, we show several structural results as well as three splitting operations for graphs with bridge edges in the solid graph. These insights will be important for our FPT algorithm and will also simplify instances of URGR where the solid graph is a tree.

Consider a bridge edge  $e = \{u, v\}$  whose deletion splits the solid graph  $G$  into connected components  $G_u$  and  $G_v$ , where  $G_u$  contains  $u$  and  $G_v$  contains  $v$ . A dashed arc  $(a, b)$  *spans*  $e$  if  $a \in V(G_u)$  and  $b \in V(G_v)$  or vice versa. Two arcs  $(a, b)$  and  $(c, d)$  span  $e$  in the same direction if they span  $e$  and either  $a, c \in V(G_u)$  or  $a, c \in V(G_v)$ . If  $e$  has a single label in a realization, then it must be the case that the dashed arcs spanning  $e$  are “transitive” in the following sense: If dashed arcs  $(a, b)$  and  $(c, d)$  both span  $e$  in the same direction, then  $(a, d)$  and  $(c, b)$  must also be dashed arcs. This is because for any two temporal paths that pass through  $e$  in the same direction, the part of one path up to edge  $e$  can be combined with the part of the other path after  $e$ .

► **Definition 6.** Consider a bridge edge  $e = \{u, v\}$  whose deletion splits the solid graph  $G$  into connected components  $G_u$  and  $G_v$ . The edge  $e$  is a *special bridge edge* if there exist vertices  $a$  in  $G_u$  and  $b$  in  $G_v$  such that  $D_{av} = 1$ ,  $D_{ub} = 1$  and  $D_{ab} = 0$  (or if the same condition holds with  $u$  and  $v$  exchanged). A bridge edge that is not special is called *non-special*.

Intuitively, a special bridge edge is a bridge edge  $e$  such that there are dashed arcs spanning  $e$  in the same direction for which transitivity (as outlined above) is violated.

► **Lemma 7 (★).** In each frugal realization of  $D$ , every special bridge edge is assigned two labels and every non-special bridge edge is assigned a single label.

► **Corollary 8.** Let  $D$  be an instance of any version of URGR under consideration besides ANY STRICT URGR. If the solid graph of  $D$  contains a special bridge edge, then  $D$  is a no-instance.

### 4.1 Splitting Into Subinstances

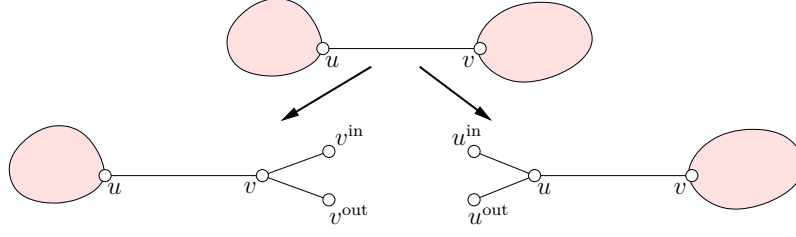
We now define splitting operations on bridges. First, consider non-special bridges.

► **Lemma 9 (Splitting at a non-special bridge).** Consider any variant of URGR. Let  $e = \{u, v\}$  be a non-special bridge edge, and let  $G_u$  and  $G_v$  be the connected components of  $G$  resulting from the deletion of  $e$ . Then the instance  $D$  is realizable if and only if the subinstances induced by  $V(G_u) \cup \{v\}$  and  $V(G_v) \cup \{u\}$  are realizable.

**Proof.** Let  $D_u$  and  $D_v$  denote the subinstances induced by  $V(G_u) \cup \{v\}$  and  $V(G_v) \cup \{u\}$ , respectively. If  $D$  is realizable, then the realization  $\lambda$  induces realizations of the two subinstances  $D_u$  and  $D_v$ . If  $D_u$  and  $D_v$  are realizable, their realizations assign a single label to  $e$ , and these realizations can be chosen so that  $e$  receives the same label in both realizations. Hence, the union of the two realizations is a realization of  $D$ . ◀

By applying Lemma 9 repeatedly to a non-pendant non-special bridge edge until no such edge exists, we obtain subinstances in which all non-special bridge edges are pendant.

Now consider special bridges. Special bridge edges cannot occur in yes-instances of any variant of URGR except ANY STRICT URGR (see Corollary 8), so we only consider ANY STRICT URGR in the following. Let edge  $e = \{u, v\}$  be a special bridge with  $G_u$  and  $G_v$  defined as above. If the instance is realizable, for every vertex in  $V(G_u)$  there are at most three possibilities for which vertices in  $V(G_v)$  it can reach, depending on whether it cannot reach  $v$ , can reach  $v$  at time  $\alpha$ , or can reach  $v$  only at time  $\beta$ , where  $\alpha$  and  $\beta$  with  $\alpha < \beta$  are



■ **Figure 1** Illustration of the splitting operation at a special bridge  $\{u, v\}$  (Lemma 11).

the labels assigned to  $e$  in the realization. The same holds with  $G_u$  and  $G_v$  exchanged. If this condition is violated, then the instance cannot be realized. The following definition captures this condition. For each vertex  $w \in V(G_u)$ , let  $R_{u \rightarrow v}(w) = \{x \in V(G_v) \mid (w, x) \in A\}$ , and define  $R_{v \rightarrow u}(x)$  for  $x \in V(G_v)$  analogously.

► **Definition 10.** Let  $e = \{u, v\}$  be a special bridge with  $G_u$  and  $G_v$  defined as above. We say that  $e$  is a special bridge edge with plausible reachability if the following conditions hold:

- If there exist vertices  $a \in V(G_u)$  and  $b \in V(G_v)$  such that  $D_{av} = D_{ub} = 1$  and  $D_{ab} = 0$ , then every vertex  $w \in V(G_u)$  satisfies  $R_{u \rightarrow v}(w) \in \{\emptyset, R_{u \rightarrow v}(a), R_{u \rightarrow v}(u)\}$  with  $\emptyset \subset R_{u \rightarrow v}(a) \subset R_{u \rightarrow v}(u)$ . Otherwise, every vertex satisfies  $R_{u \rightarrow v}(w) \in \{\emptyset, R_{u \rightarrow v}(u)\}$ .
- The same condition holds with the roles of  $G_u$  and  $G_v$  exchanged.

If there is a special bridge edge  $\{u, v\}$  with plausible reachability, we can split the instance into two subinstances that can be solved separately, as illustrated in Figure 1. Each subinstance consists of the edge  $\{u, v\}$  together with either  $G_u$  or  $G_v$ . For the subinstance containing  $\{u, v\}$  and  $G_u$ , up to two new leaf vertices are attached to  $v$ , named  $v^{\text{in}}$  and  $v^{\text{out}}$  in the figure. Here,  $v^{\text{in}}$  represents the vertices in  $G_v$  that can reach  $u$  only by traversing  $\{u, v\}$  using the larger of its two time labels, and  $v^{\text{out}}$  represents the vertices in  $G_v$  that can be reached from  $G_u$  only by traversing  $\{u, v\}$  using the larger of its two time labels. The other subinstance is handled analogously.

► **Lemma 11** (( $\star$ ), Splitting at a special bridge). Consider ANY STRICT URGR. Let  $e = \{u, v\}$  be a special bridge edge with plausible reachability, and let  $G_u$  and  $G_v$  be the connected components of  $G$  resulting from the deletion of  $e$ . Then the instance  $D$  is realizable if and only if the two subinstances constructed as follows are realizable:

- To construct subinstance  $D_u$ , take the subgraph  $D'$  of  $D$  induced by  $V(G_u) \cup \{v\}$  and attach one or two leaves to  $v$  as follows:
  - If there are vertices  $a \in V(G_u)$  and  $b \in V(G_v)$  with  $D_{av} = D_{ub} = 1$  and  $D_{ab} = 0$ , then attach a leaf (called out-leaf)  $z$  to  $v$  with  $D'_{zv} = 1$  and  $D'_{az} = 1$  for all  $a \in \{v\} \cup \{w \in V(G_u) \mid R_{u \rightarrow v}(w) = R_{u \rightarrow v}(u)\}$ , and all other entries of  $D'$  involving  $z$  equal to 0.
  - If there are vertices  $c \in V(G_v)$  and  $d \in V(G_u)$  with  $D_{cu} = D_{vd} = 1$  and  $D_{cd} = 0$ , then attach a leaf (called in-leaf)  $z'$  to  $v$  with  $D'_{vz'} = 1$  and  $D'_{z'a} = 1$  for all  $a \in \{v\} \cup R_{v \rightarrow u}(c)$  and  $D'_{z'z} = 1$  if an out-leaf  $z$  has been added, and all other entries of  $D'$  involving  $z'$  equal to 0.

Note that at least one of the two conditions above must be satisfied because  $e$  is a special bridge edge. The resulting instance  $D'$  is the desired subinstance  $D_u$ .

- Subinstance  $D_v$  is constructed analogously, with the roles of  $u$  and  $v$  exchanged.

Note that we cannot exhaustively apply the operation behind Lemma 11, since each application on a special bridge  $\{u, v\}$  results into two new instances in which edge  $\{u, v\}$  is again a special bridge. However, by recursively applying this operation on special bridges on



which the operation has not yet been applied, we eventually obtain instances, where each special bridge has at least one endpoint that has at most two other neighbors and both of them have a degree of 1.

Finally, we consider the case of two pendant edges that have a common neighbor and must receive the same single label in every realization. We argue that we can remove one of them from the instance provided their reachability requirements from/to the rest of the graph are the same. This is because a realization of the instance with one of the two pendant edges removed gives a realization of the original instance simply by labeling the pendant edge that was removed with the same label as the other pendant edge.

► **Lemma 12** (( $\star$ ), Removal of redundant pendant vertices). *Let  $D$  be an instance of URGR and let  $v$  and  $w$  be two degree-1 vertices with the same common neighbor  $u$  in  $G$  that satisfy  $D_{vw} = D_{wv} = 0$ . Then,*

- *for all variants of NON-STRICT URGR (including PROPER URGR and HAPPY URGR),  $D$  is not realizable, and*
- *for ANY STRICT URGR and SIMPLE STRICT URGR,  $D$  is realizable if and only if (i) the subinstance  $D'$  resulting from  $D$  by deleting  $v$  is realizable and (ii) for each  $x \in V \setminus \{v, w\}$ ,  $D_{vx} = D_{wx}$  and  $D_{xv} = D_{xw}$ .*

## 4.2 Algorithms for Instances with a Tree as Solid Graph

Since all edges of trees are bridges, we now describe, based on our insights about labels in frugal realizations, an algorithm for instances of URGR where the solid graph is a tree.

► **Theorem 13.** *There is a polynomial-time algorithm for solving instances of ANY STRICT URGR for which the solid edges form a tree.*

**Proof Sketch.** Let  $T = (V, E)$  denote the tree of solid edges. We first check that  $D$  satisfies several conditions that must hold if  $D$  is realizable. For example, if  $(u, v)$  is an arc, then  $(u, w)$  and  $(w, v)$  must also be arcs for every internal node  $w$  of the unique  $u$ - $v$ -path in  $T$ . If one of these conditions is violated,  $D$  is not realizable. Otherwise, we construct a linear program (LP) that has two non-negative variables  $\ell_e$  and  $h_e$  for each  $e \in E$  that represent the lower and higher label of  $e$ , respectively, in a realization. Constraints  $\ell_e = h_e$  for non-special edges and  $\ell_e + 1 \leq h_e$  for special edges ensure that each edge receives the correct number of labels. For edges  $uv$  and  $vw$  such that  $(u, w) \in A$ , the constraints  $\ell_{uv} + 1 \leq h_{vw}$  and  $h_{uv} \leq \ell_{vw}$  ensure that there is a temporal  $u$ - $w$ -path but no temporal  $w$ - $u$ -path. Call a non-arc  $(u, v)$  *minimal* if it is the only arc missing in the direction from  $u$  to  $v$  on the path from  $u$  to  $v$ . For every minimal non-arc  $(u, v)$ , we add constraints  $h_{e_i} = \ell_{e_{i+1}}$  for every pair of consecutive edges  $e_i$  and  $e_{i+1}$  on the  $u$ - $v$  path (which we show to consist of special edges except for the first and last edge, which are both non-special). For arcs  $(u, v)$  such that the first and last edge of the  $u$ - $v$ -path are non-special and all other edges are special, we add the constraint  $\sum_i (\ell_{e_{i+1}} - h_{e_i}) \geq 1$ , where the sum is over all pairs of consecutive edges  $e_i, e_{i+1}$  of the  $u$ - $v$ -path. We can show that the LP is feasible if and only if  $D$  is realizable. Furthermore, any feasible solution of the LP corresponds to a frugal realization with fractional labels, which can be made integral in a straightforward post-processing step. ◀

By testing additional conditions (such as the absence of special edges) before constructing the linear program, we can also use the approach to solve all other variants of URGR.

## 5 Hardness for Undirected Reachability Graph Realizability

In this section, we show that all considered versions of **UNDIRECTED REACHABILITY GRAPH REALIZABILITY** are NP-hard, even on instances with a constant maximum degree. We obtain these hardness results by reductions from SAT, where we have a source  $s_i$  and a terminal vertex  $t_i$  for each clause  $c_i$ , such that  $D$  contains the dashed arc  $(s_i, t_i)$ . The only way to realize such an arc is to follow a dense path that crosses through some variable gadget. We design a variable gadget for each variable  $x$ , for which the only way to realize it is to ensure that (i) only clauses that contain  $x$  positively can reach their terminal via this variable gadget, or (ii) only clauses that contain  $x$  negatively can reach their terminal over this variable gadget.

► **Theorem 14** ( $\star$ ). *Each version of URGR under consideration is NP-hard on directed graphs of constant maximum degree. Moreover, no version of URGR under consideration can be solved in  $2^{o(|V|+|A|)} \cdot n^{\mathcal{O}(1)}$  time, unless the ETH fails.*

Hence, the running time of Theorem 5 can presumably not be improved significantly.

We now strengthen our hardness result for **ANY STRICT URGR** and **SIMPLE STRICT URGR**, which will highly motivate the analysis of parameterized algorithms for the parameter “feedback edge set number” of the solid graph, which we consider in Section 6.

► **Theorem 15** ( $\star$ ). *ANY STRICT URGR and SIMPLE STRICT URGR are  $W[2]$ -hard when parameterized by the feedback vertex set number and treedepth of the solid graph.*

**Proof.** We reduce from **SET COVER** which is  $W[2]$ -hard when parameterized by  $k$  [10].

Let  $I := (U, \mathcal{F}, k)$  be an instance of **SET COVER** with  $\mathcal{F} = \{F_1, \dots, F_r\}$ . Assume without loss of generality that each element of  $U$  is contained in at least one hyperedge and that  $\mathcal{F}$  has size at least  $k$ , as otherwise  $I$  could be solved trivially. We obtain a directed graph  $D = (V, A)$  with solid graph  $G = (V, E)$  as follows: The graphs contain the vertices  $\{\top, a, a', b, b', \perp\}$ , each element  $u \in U$  as a vertex, and for each  $i \in [1, k]$  the vertex  $c_i$ . Additionally, for each  $F \in \mathcal{F}$  and each  $u \in F$ ,  $G$  contain the vertices  $w_u^F$ ,  $v_u^F$ , and  $q_u^F$ .

Next, we describe the solid edges of  $G$ , that is, the bidirectional arcs of  $D$ . See Figure 2 for the solid edges and dashed arcs of the main connection gadget of the instance.

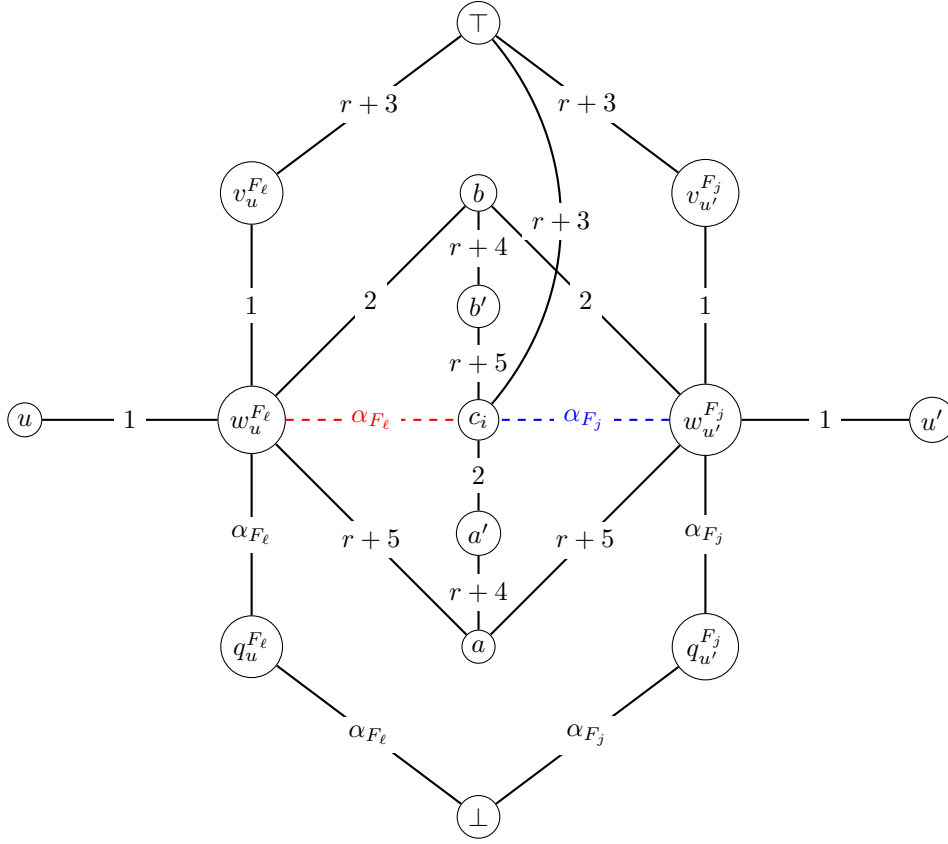
The graph  $G$  contains the edges  $\{a, a'\}$ ,  $\{b, b'\}$ , and for each  $i \in [1, k]$ , the edges  $\{a', c_i\}$ ,  $\{c_i, b'\}$ , and  $\{c_i, \top\}$ . Moreover, for each  $F \in \mathcal{F}$  and each  $u \in F$ ,  $G$  contains the edges  $\{u, w_u^F\}$ ,  $\{w_u^F, a\}$ ,  $\{w_u^F, b\}$ ,  $\{\{w_u^F, c_i\} \mid i \in [1, k]\}$ ,  $\{w_u^F, v_u^F\}$ ,  $\{w_u^F, q_u^F\}$ ,  $\{v_u^F, \perp\}$ , and  $\{q_u^F, \top\}$ .

Finally, we describe the dashed arcs:  $D$  contains the dashed arcs  $(a', b)$ ,  $(b, a)$ ,  $(a', \top)$ ,  $(b, \top)$ , and  $(\top, b')$ . For each  $i \in [1, k]$ ,  $D$  also contains the dashed arcs  $(b, c_i)$  and  $(c_i, a)$ . Let  $u \in U$ . Then,  $D$  contains the arc  $(u, \top)$ . Moreover,  $D$  contains the arcs  $(u, a)$ ,  $(u, b)$ ,  $(u, b')$ , and  $(u, c_i)$  for each  $i \in [1, k]$ . Additionally, for each hyperedge  $F \in \mathcal{F}$  with  $u \in F$ ,  $D$  contains the arcs  $(u, v_u^F)$ ,  $(v_u^F, a)$ ,  $(b, v_u^F)$ ,  $(w_u^F, \top)$ ,  $(w_u^F, b')$ ,  $(a', w_u^F)$ ,  $(q_u^F, v_u^F)$ ,  $(q_u^F, a)$ ,  $(q_u^F, b)$ ,  $(q_u^F, b')$ , and  $(q_u^F, c_i)$  for each  $i \in [1, k]$ . The only other dashed arcs in  $D$  are between  $v_u^F$ -vertices. Let  $F_x, F_y \in \mathcal{F}$  with  $x < y$  and let  $u_x \in F_x$  and  $u_y \in F_y$ . Then,  $D$  contains the arc  $(v_{u_x}^{F_x}, v_{u_y}^{F_y})$ .

This completes the construction of  $D$ . First, we show the parameter bounds.

► **Claim 16** ( $\star$ ).  $G$  has a feedback vertex set of size  $k + 6$  and treedepth  $\mathcal{O}(k)$ .

We show that  $I$  is a yes-instance of **SET COVER** if and only if  $D$  is realizable via strict temporal paths. More precisely, we show that if  $I$  is a yes-instance of **SET COVER**, then there is a simple undirected temporal graph with strict reachability graph  $D$ . This then implies the correctness of the reduction for both **ANY STRICT URGR** and **SIMPLE STRICT URGR**.



	$u$	$\top$	$v_u^{F_\ell}$	$w_u^{F_\ell}$	$q_u^{F_\ell}$	$a$	$a'$	$b$	$b'$	$c_i, i \in [1, k]$	$\perp$
$u$		1	1	1	0	1	0	1	1	1	0
$\top$	0		0	0	1	0	0	0	1	1	0
$v_u^{F_\ell}$	0	0		1	0	1	0	0	0	0	1
$w_u^{F_\ell}$	1	1	1		1	1	0	1	1	1	0
$q_u^{F_\ell}$	0	1	1	1		1	0	1	1	1	0
$a$	0	0	0	1	0		1	0	0	0	0
$a'$	0	1	0	1	0	1		0	1	1	0
$b$	0	1	1	1	0	1	0		1	1	0
$b'$	0	0	0	0	0	0	0	1		1	0
$c_i, i \in [1, k]$	0	1	0	1	0	1	1	0	1		0
$\perp$	0	0	1	0	0	0	0	0	0	0	

■ **Figure 2** Illustration of a part of the adjacency matrix of the W[2]-hardness reduction. The highlighted cells of the matrix indicate the dashed arcs. As we show, the labels  $\alpha_{F_\ell}$  and  $\alpha_{F_j}$  need to be distinct if the hyperedges  $F_\ell$  and  $F_j$  are distinct. Thus, for each  $i \in [1, k]$ , labeling both edges  $\{w_u^{F_\ell}, c_i\}$  and  $\{w_{u'}^{F_j}, c_i\}$  is not possible, as otherwise, there would be a temporal path from  $w_u^{F_\ell}$  to  $w_{u'}^{F_j}$  (or vice versa), which is not allowed.

We defer this proof to the full version and only provide an informal idea for the correctness. Intuitively, in each realization for  $D$ , for each  $i \in [1, k]$ , there can be at most one hyperedge  $F \in \mathcal{F}$  for which edges between  $c_i$  and vertices of  $W_F := \{w_u^F \mid u \in F\}$  can receive labels. This can be seen as follows: For each  $w_{u'}^{F'} \in V$ , (i) there is no dashed arc between  $w_{u'}^{F'}$  and  $\perp$  and (ii) there is no dashed arc between  $v_{u'}^{F'}$  and  $c_i$ . Hence, Observation 1 implies that, if  $\{w_{u'}^{F'}, c_i\}$  receives at least one label, then there is some  $\alpha_{F'} \in \mathbb{N}$ , such that the edges  $\{w_{u'}^{F'}, c_i\}$  and  $\{v_{u'}^{F'}, \perp\}$  receive the label set  $\{\alpha_{F'}\}$  under  $\lambda$ . Based on the dashed arcs between the  $v_{u'}^{F'}$ -vertices, the label  $\alpha_{F'}$  and the label  $\alpha_{F''}$  are distinct for distinct hyperedges  $F'$  and  $F''$ . This then implies that there can be at most one hyperedge  $F \in \mathcal{F}$  for which edges between  $c_i$  and vertices of  $W_F$  can receive labels, as otherwise, a strict temporal path between distinct  $w_{u'}^{F'}$ -vertices would be realized.

The edges with at least one label between  $c_i$  and  $w_u^F$ -vertices thus resemble a selection of at most one hyperedge of  $\mathcal{F}$  for each  $i \in [1, k]$ . Since the only dense paths from a vertex  $u \in U$  to  $\top$  are of the form  $(u, w_u^F, c_i, \top)$  for  $i \in [1, k]$  and  $F \in \mathcal{F}$  with  $u \in F$ , this selection of  $k$  hyperedges encodes a set cover, as the arcs  $(u, \top)$  are realized over such dense paths.  $\blacktriangleleft$

## 6 Parameterizing by the Feedback Edge Set Number

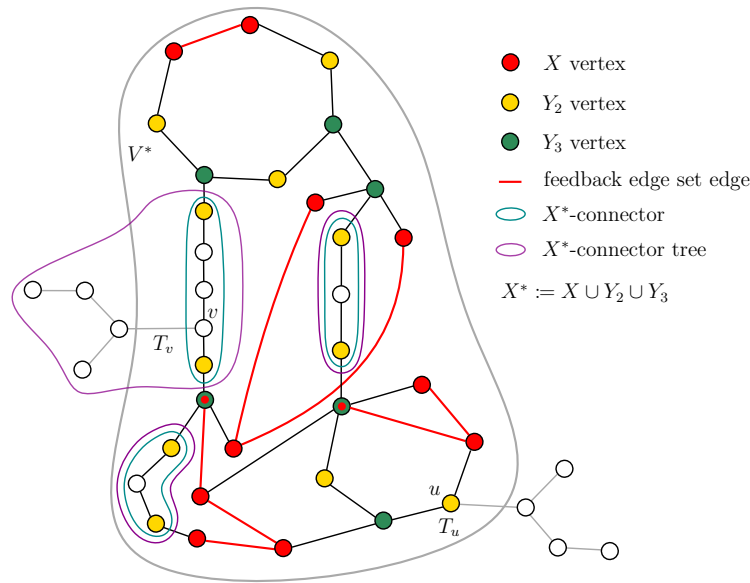
In this section we generalize our algorithm on tree instances of URGR to tree-like graphs. Recall that a feedback edge set of a graph  $G$  is a set  $F$  of edges of  $G$ , such that  $G - F$  is acyclic. We denote by  $\text{fes}$  the feedback edge set number of the solid graph  $G$  of  $D$ , that is, the size of the smallest feedback edge set of  $G$ .

► **Theorem 17.** *Each version of URGR can be solved in  $\text{fes}^{\mathcal{O}(\text{fes}^2)} \cdot n^{\mathcal{O}(1)}$  time.*

Recall that the parameter can presumably not be replaced by a smaller parameter like feedback vertex set number or treedepth (see Theorem 15). In the remainder of this section, we present the main idea for the proof of Theorem 17. We will only describe the algorithm for the most general version ANY STRICT URGR, but for all more restrictive versions, all arguments work analogously.

**Definitions and Notations.** Let  $G = (V, E)$  be the solid graph and let  $F$  be a minimum size feedback edge set of  $G$ . We assume without loss of generality that  $G$  is connected, as otherwise, we can solve each connected component independently, or detect in polynomial time that  $D$  is not realizable if there are dashed arcs between different connected components of the solid graph. Moreover, let  $X$  denote the endpoints of the edges of  $F$ . Note that  $|X| \leq 2 \cdot |F|$ . We define the set  $V^*$  as the (unique) largest subset  $S$  of vertices of  $V$  that each have at least two neighbors in  $G[S]$ , that is,  $V^*$  is the 2-core [24] of  $G$ . This set can be computed in polynomial time. Moreover, we define  $V' := V \setminus V^*$ . Note that  $X \subseteq V^*$ , since  $F$  is a minimum-size feedback edge set. Since  $G$  is connected, for each vertex  $v' \in V'$ , there is a unique vertex  $v \in V^*$  which has closest distance to  $v'$  among all vertices of  $V^*$ . For a vertex  $v \in V^*$ , denote by  $V_v$  the set of all vertices of  $V'$  for which  $v$  is the closest neighbor in  $V^*$ . Note that  $V_v$  might be empty. Since  $v'$  is in no cycle, removing  $v$  from  $G$  would result in  $v'$  ending up in a component that has no vertex of  $V^*$ . Hence,  $T_v := G[\{v\} \cup V_v]$  is a tree for which we call  $v$  the *root*. Moreover, we call  $T_v$  the *pendant tree* of  $v$ .

Recall that each vertex of  $V^*$  has degree at least 2 in  $G[V^*]$ . Furthermore,  $G - F$  is acyclic and thus  $G[V^*] - F$  is a tree. Let  $Y_3$  denote the set of all vertices of degree at least 3 in  $G[V^*] - F$ , and let  $Y_2$  be the neighbors of  $X \cup Y_3$  in  $V^* \setminus (X \cup Y_3)$ . Observe that each leaf of  $G[V^*] - F$  is a vertex of  $X$ . Moreover, in each tree with  $\ell$  leaves, there



■ **Figure 3** An illustration of the main definitions used by the FPT algorithm for parameter fes. The grey area contains the vertices of  $V^*$ , that is, the 2-core of  $G$ .

are  $\mathcal{O}(\ell)$  vertices having (i) degree at least 3 or (ii) a neighbor of degree at least 3. This implies that  $|Y_2 \cup Y_3| \in \mathcal{O}(|X|)$ . Let  $X^* := X \cup Y_2 \cup Y_3$ . See Figure 3 for an illustration of these sets of vertices and some of the main definitions used in this section. By the above,  $|X^*| \in \mathcal{O}(|X|) \subseteq \mathcal{O}(\text{fes})$ . Recall that each vertex of  $Y_2$  has degree exactly 2 in  $G[V^*]$ . Moreover, note that each vertex of  $V^* \setminus X^*$  is part of a unique path  $P$  where each internal vertex of  $P$  is from  $V^* \setminus X^*$  and the endpoints of  $P$  are from  $Y_2$ . For each such path, each internal vertex has degree exactly 2 in  $G[V^*]$  and there are at most  $|X^*| - 1$  such paths, since  $G[V^*] - F$  is acyclic. Note that  $X^*$  contains all vertices that are part of triangles in  $G[V^*]$  and thus in  $G$ . This implies that in each minimal realization of  $D$ , each edge incident with at least one vertex of  $V \setminus X^*$  receives at most two labels (see Lemma 2). We now define the above mentioned paths between the vertices of  $Y_2$  in a more general way.

► **Definition 18.** Let  $W$  with  $X^* \subseteq W \subseteq V^*$  and let  $P$  be a path of length at least 2 in  $G[V^*]$  with endpoints  $a$  and  $b$  in  $W$  and all internal vertices from  $V^* \setminus W$ . We call  $P$  a  $W$ -connector. Moreover, let  $C := G[V(P) \cup \bigcup_{q \in V(P) \setminus \{a,b\}} V_q]$ . We call  $C$  a  $W$ -connector tree and the extension of  $P$ .

Note that the endpoints of each  $X^*$ -connector are from  $Y_2$  and thus have degree 2 in  $G[V^*]$ . Moreover, each internal vertex  $v$  of a  $W$ -connector  $P$  has degree exactly 2 in  $G[V^*]$ , that is, the only two neighbors of  $v$  in  $G[V^*]$  are the predecessor and the successor of  $v$  in  $P$ . This also implies that each endpoint of a  $W$ -connector has degree exactly 2 in  $G[V^*]$ . Since each edge of  $P$  is incident with at least one vertex of  $V \setminus W \subseteq V \setminus X^*$ , in each minimal realization of  $D$ , each edge of each  $W$ -connector receives at most two labels (see Lemma 2). The latter also holds for each  $W$ -connector tree.

► **Observation 19.** Let  $W$  with  $X^* \subseteq W \subseteq V^*$ . Then, there are  $\mathcal{O}(|W| + \text{fes})$  edges between vertices of  $W$  in  $G$  and there are  $\mathcal{O}(|W| + \text{fes})$   $W$ -connectors. Moreover, for each vertex  $q \in V \setminus (W \cup \bigcup_{w \in W} V_w)$ , there is exactly one  $W$ -connector tree that contains  $q$ .

Note that this is a general property of graphs with a feedback edge set of size fes.

**Abstract Description of the Algorithm.** Our algorithm uses two preprocessing steps.

In the first step, we will present a polynomial-time reduction rule based on the splitting operations presented in Section 4. With this reduction rule, we will be able to remove vertices from large pendant trees. After exhaustive application, for each vertex  $x \in V^*$ , the pendant tree  $T_x$  will have  $\mathcal{O}(d_x)$  vertices, where  $d_x$  denotes the degree of  $x$  in  $G[V^*]$ .

In the second step, we extend the set  $X^*$  to a set  $W^* \subseteq V^*$  such that each  $W^*$ -connector has some useful properties (we will define connectors with these properties as *nice connectors*). Intuitively, a connector  $P$  with extension  $C$  is nice if (i) in a realization for  $D$ , the arcs in  $D[V(C)]$  can only be realized by temporal paths that are contained in the connector tree  $C$ , and (ii) for each arc between a vertex outside of  $C$  and a vertex inside of  $C$ , we can in polynomial time detect over which (unique) edge of the connector incident with one of its endpoints this arc is realized. As we will show, we can compute such a set  $W^*$  of size  $\mathcal{O}(\text{fes})$  in polynomial time, or correctly detect that the input graph is not realizable. By the first part, we additionally get that the total number of vertices in pendant trees that have their root in  $W^*$  is  $\mathcal{O}(|W^*| + \text{fes}) = \mathcal{O}(\text{fes})$ .

The algorithm then works as follows: We iterate over all possible partial labelings  $\lambda$  on the  $\mathcal{O}(\text{fes})$  edges incident with vertices of  $W^*$  or vertices of pendant trees that have their root in  $W^*$ . As we will show, we can assume that each of these edges will only receive  $\mathcal{O}(\text{fes})$  labels in each minimal realization. For each such labeling  $\lambda$ , we need to check whether we can extend the labeling to the so far unlabeled edges, that is, the edges that are part of any  $W^*$ -connector tree. As we will show, we can compute for each such connector  $P$  with extension  $C$  in polynomial time a set  $L_P$  of  $\mathcal{O}(1)$  labelings for the edges of  $C$ , such that if  $\lambda$  can be extended to a realization for  $D$ , then we can extend  $\lambda$  (independently from the other connectors) by one of the labelings in  $L_P$ . Since  $W^*$  has size  $\mathcal{O}(\text{fes})$ , there are only  $\mathcal{O}(\text{fes})$  many  $W^*$ -connectors (see Observation 19) and for each such connector  $P$ , the set  $L_P$  of labelings has constant size. Our algorithm thus iterates over all possible  $\mathcal{O}(1)^{\mathcal{O}(\text{fes})} = 2^{\mathcal{O}(\text{fes})}$  labeling combinations for the connectors and checks whether one of these combinations extends  $\lambda$  to a realization for  $D$ . Summarizing, we will show the following.

► **Proposition 20.** *In polynomial time, we can detect that  $D$  is not realizable, or compute a set  $W^*$  with  $X^* \subseteq W^* \subseteq V^*$ , such that the set of edges  $E^*$  that are (i) incident with at least one vertex of  $W^*$  or (ii) part of some pendant tree with root in  $W^*$ , has size  $\mathcal{O}(\text{fes})$  and where for each labeling  $\lambda$  of the edges of  $E^*$ , we can in  $2^{\mathcal{O}(\text{fes})} \cdot n^{\mathcal{O}(1)}$  time (a) compute a labeling that realizes  $D$ , or (b) detect that there no frugal realization for  $D$  that agrees with  $\lambda$  on the labels of all edges of  $E^*$ .*

The running time of this algorithm is then  $\text{fes}^{\mathcal{O}(\text{fes}^2)} \cdot n^{\mathcal{O}(1)}$ , since (i) all preprocessing steps run in polynomial time, (ii) we only have to consider  $\text{fes}^{\mathcal{O}(\text{fes}^2)}$  labelings  $\lambda$  (since we prelabel  $\mathcal{O}(\text{fes})$  edges with  $\mathcal{O}(\text{fes})$  labels each), and (iii) for each labeling  $\lambda$ , we only have to check for  $2^{\mathcal{O}(\text{fes})}$  possible ways to extend  $\lambda$  to a realization for  $D$ . To see the second part, we show that it is sufficient to assign only labels of  $\{i \cdot 2 \cdot n \mid i \in \mathcal{O}(\text{fes}^2)\}$  in  $\lambda$ .

## 6.1 Technical Highlights and Difficulties of the Algorithm

We now describe in more detail the intuition about the two preprocessing steps, as well as the algorithm to compute a constant number of labelings for each nice connector.

**Dealing with pendant trees.** Recall that  $d_x$  denotes the degree of a vertex  $x \in V^*$  in  $G[V^*]$ . To reduce each pendant tree  $T_x$  with  $x \in V^*$  to a size of  $\mathcal{O}(d_x)$ , we first apply the three splitting operations of Section 4 to each edge of  $T_x$  incident with  $x$ . Since each of these edges



is a bridge and part of a tree, one of the two resulting instances produced by the splitting operations is then a tree and can be solved in polynomial time. We thus stick with the one resulting instance which is not a tree. As we show, by applying the splitting operations, we get that (i) for each special bridge  $\{b, x\}$  of  $T_x$  incident with  $x$ ,  $b$  has at most two neighbors besides  $x$  (namely, the possible in-leaf and the possible out-leaf) and (ii) for each non-special bridge  $\{b, x\}$  of  $T_x$  incident with  $x$ ,  $b$  is a leaf. This directly implies that  $T_x$  has depth 2 and the number of leaves of  $T_x$  is  $\mathcal{O}(|B_x|)$ , where  $B_x$  denotes the neighbors of  $x$  in  $T_x$ .

Thus, to reduce the size of  $T_x$  it suffices to reduce the number of vertices in  $B_x$  to  $\mathcal{O}(d_x)$ . To do so, we show that for each external edge  $e$  (that is, an edge that is incident with  $x$  but not contained in  $T_x$ ), we can detect in polynomial time a set of at most 5 internal edges (that is, edges between  $x$  and vertices of  $B_x$ ) that “surround” or “block” the edge  $e$ . Intuitively, in each realization for  $D$ , no other internal edge can share a label with  $e$ . This defines a total of  $\mathcal{O}(d_x)$  internal edges that are surrounding or blocking. Let  $S_x$  denote these edges. Based on this set of edges, we then show that we can efficiently (i) detect that  $D$  is not realizable, or (ii) find a vertex  $b$  of  $B_x$  that (together with its possible leaf-neighbors) can safely be removed from  $D$ , if  $B_x$  has size  $\omega(|S_x|)$ . Hence, after exhaustive application of this operation, only  $\mathcal{O}(|S_x|) \subseteq \mathcal{O}(d_x)$  vertices of  $B_x$  remain. Since each of these vertices only has at most two neighbors besides  $x$ , and these neighbors are leaves, the total size of  $T_x$  is then  $\mathcal{O}(d_x)$ .

**Ensuring connectors with nice properties.** We now give a more formal idea behind the definition of nice connectors. Let  $P$  be a connector with endpoints  $a$  and  $b$ , and let  $C$  be the extension of  $P$ . Then,  $P$  is nice, if (i) there is no dense path between  $a$  and  $b$  outside of  $D[V(C)]$  and (ii) for each arc  $(u, v) \in D$  between a vertex of  $C$  and a vertex outside of  $C$ , there is no dense  $(u, v)$ -path that goes over  $a$  or there is no dense  $(u, v)$ -path that goes over  $b$ .

Since the definition of nice connectors relies on the non-existence of dense paths between specific vertex pairs, detecting whether a connector is nice can presumably not be done in polynomial time.<sup>2</sup> To still achieve the goal of finding a desired set  $W^*$  where each connector is nice, we show the following: Whenever we have a set  $W \supseteq X^*$  and a  $W$ -connector  $P$ , then we can in polynomial time (i) detect that  $D$  is not realizable or (ii) compute a constant number of vertices  $U_P$  of  $P$ , such that each subpath of  $P$  that is a  $(W \cup U_P)$ -connector is nice. That is, even though we cannot check efficiently whether the nice property is fulfilled for a given connector, we can ensure it by adding few vertices to our set. To compute the set  $W^*$ , we start with  $W^* = X^*$  and iteratively add for each  $X^*$ -connector  $P$  the set  $U_P$  to  $W^*$ . Since  $|X^*| \in \mathcal{O}(\text{fes})$  and there are  $\mathcal{O}(\text{fes})$   $X^*$ -connectors, the resulting set  $W^*$  also has size  $\mathcal{O}(\text{fes})$ . As we show, the update of  $W^*$  preserves the nice property of the connector before and after the update. In this way, we ensure that each  $W^*$ -connector is nice.

**Dealing with nice connectors.** Recall the definition of nice connectors. The first property essentially ensures that arcs in  $D[V(C)]$  can only be realized via dense paths in  $D[V(C)]$ , which makes the local realization of  $D[V(C)]$  in some sense independent from the remainder of  $D$ . The second property ensures that we only have one choice (with respect to dense paths inside of  $D[V(C)]$ ) to realize an arc  $(u, v)$  between an internal and an external vertex.

We make use of these properties, since a nice connector  $P$  and its extension  $C$  only interact with the remainder of  $D$  via exactly two edges  $e_a$  and  $e_b$ . Moreover, both these edges receive at most two labels in every frugal realization. Assuming we are given the

<sup>2</sup> This is due to the fact that finding a dense path between two specific vertices can be shown to be NP-hard by reducing from MULTICOLORED CLIQUE.

labeling for  $e_a$  and  $e_b$ , we show that we can compute a constant number of labelings  $L_P$  for the edges of  $C$ , such that if there is a realization  $\lambda$  for  $D$  agreeing with the prelabeling of  $e_a$  and  $e_b$ , then we can replace the labels of  $\lambda$  on the edges of  $C$  by the labels of some labeling in  $L_P$ . The essential idea behind this algorithm is to build a constant number of tree instances in which we simulate how the connector interacts with the remainder of  $D$  in any possible labeling. As we show, we only have to (i) consider the cases for each of the edges  $e_a$  and  $e_b$ , whether there are temporal paths in  $\lambda$  that enter or exit  $C$  via  $e_a$  ( $e_b$ ) via the smallest or largest label of  $e_a$  ( $e_b$ ), and (ii) consider the cases whether some temporal paths under  $\lambda$  enter  $C$  via  $e_a$  and leave via  $e_b$  and whether they traverse the respective edges at their smallest or largest label. These are in total only a constant number of options. For each such option, we construct a tree instance of the problem obtained from  $D[V(C)]$  by adding a constant number of vertices each to simulate the specific option. In some sense, these instances are generalizations of the instances built in the splitting rule for special bridges. Afterwards, we check for which of these instances we can find a labeling that agrees with the prelabeling on  $e_a$  and  $e_b$  and add such a labeling to  $L_P$  if it exists. The latter task can be performed in polynomial time, by using our linear program for tree instances.

## 7 The Complexity of Directed Reachability Graph Realizability

Finally, we consider the complexity of versions of DIRECTED REACHABILITY GRAPH REALIZABILITY. As already discussed by Döring [11], each directed graph  $D$  can be the strict reachability graph of a simple directed temporal graph. Hence, ANY STRICT DRGR and SIMPLE STRICT DRGR are always yes-instances and can thus be solved in polynomial time.

We now show that all DAGs and all transitive graphs are trivial yes-instances for each version of the problem.

► **Lemma 21** ( $\star$ ). *An instance  $D = (V, A)$  of DRGR is a yes-instance, if (i) we consider ANY STRICT DRGR or SIMPLE STRICT DRGR, or (ii)  $D$  is a DAG or a transitive graph.*

Here, a directed graph  $D = (V, A)$  is *transitive*, if for every distinct vertices  $u, v, w \in V$  with  $(u, v) \in A$  and  $(v, w) \in A$ , the arc  $(u, w)$  is also in  $A$ .

All other versions of DRGR are NP-hard even on graphs that are close to DAGs. More precisely, on graphs where a DAG can be obtained by removing a constant number of arcs.

► **Theorem 22** ( $\star$ ). *PROPER DRGR, HAPPY DRGR, and all considered versions of NON-STRICT DRGR are NP-hard on graphs with a constant size feedback arc set. Moreover, none of these versions of DRGR can be solved in  $2^{o(|V|+|A|)} \cdot n^{\mathcal{O}(1)}$  time, unless the ETH fails.*

## 8 Conclusion

We studied REACHABILITY GRAPH REALIZABILITY and gave for both directed and undirected temporal graphs the complete picture for the classical complexity of all settings, answering this open problem posed by Casteigts et al. [3] and Döring [11]. For URGR, we additionally showed that the problem can be solved in FPT-time for the feedback edge set number  $\text{fes}$  of the solid graph. As we showed, this parameter cannot be replaced by smaller parameters like feedback vertex set number or treedepth of the solid graph, unless  $\text{FPT} = \text{W}[2]$ .

There are several directions for future work: First, it would be interesting to see whether (some) versions of URGR admit a polynomial kernel for  $\text{fes}$ . Another interesting task is to determine whether PROPER URGR, HAPPY URGR, or NON-STRICT URGR admits an FPT algorithm for feedback vertex set or treedepth. This is not excluded by our  $\text{W}[2]$ -hardness result, since that reduction only worked for ANY STRICT URGR and SIMPLE STRICT URGR.

Finally, one could analyze the parameterized complexity of DIRECTED REACHABILITY GRAPH REALIZABILITY with respect to directed graph parameters.

## References

- 1 Eleni C. Akrida, Leszek Gaśieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61:907–944, 2017. doi:10.1007/S00224-017-9757-X.
- 2 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.
- 3 Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theor. Comput. Sci.*, 991:114434, 2024. doi:10.1016/J.TCS.2024.114434.
- 4 Arnaud Casteigts, Michelle Döring, and Nils Morawietz. Realization of Temporally Connected Graphs Based on Degree Sequences. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: <https://doi.org/10.48550/arXiv.2504.17743>.
- 5 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.
- 6 Arnaud Casteigts, Nils Morawietz, and Petra Wolf. Distance to transitivity: New parameters for taming reachability in temporal graphs. In *49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024)*, volume 306 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.36.
- 7 Esteban Christiann, Eric Sanlaville, and Jason Schoeters. On inefficiently connecting temporal networks. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAND.2024.8.
- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 9 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285:104890, 2022. doi:10.1016/J.IC.2022.104890.
- 10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 11 Michelle Döring. Simple, Strict, Proper, and Directed: Comparing Reachability in Directed and Undirected Temporal Graphs. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: <https://doi.org/10.48550/arXiv.2501.11697>.
- 12 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/J.JCSS.2021.01.007.
- 13 Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized algorithms for multi-label periodic temporal graph realization. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAND.2024.12.
- 14 F. Göbel, J. Orestes Cerdeira, and Henk Jan Veldman. Label-connected graphs and the gossip problem. *Discret. Math.*, 87(1):29–40, 1991. doi:10.1016/0012-365X(91)90068-D.

- 15 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 504–513, 2000. doi:10.1145/335305.335364.
- 16 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. *Journal of Computer and System Sciences*, 146:103564, 2024. doi:10.1016/J.JCSS.2024.103564.
- 17 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal graph realization from fastest paths. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.SAND.2024.16.
- 18 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/S00453-018-0478-6.
- 19 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Realizing temporal transportation trees. In *Proceedings of the 51st Workshop on Graph-Theoretic Concepts in Computer Science (WG 2025)*, LNCS. Springer, 2025. To appear. Full version: arXiv:2403.18513.
- 20 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Temporal Graph Realization with Bounded Stretch. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. Full version: arXiv:2504.14258. doi:10.4230/LIPICs.MFCS.2025.75.
- 21 Julia Meusel, Matthias Müller-Hannemann, and Klaus Reinhardt. Directed temporal tree realization for periodic public transport: Easy and hard cases. In *36th International Symposium on Algorithms and Computation*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. To appear. Full version: <https://doi.org/10.48550/arXiv.2504.07920>.
- 22 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 23 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, 2018.
- 24 Stephen B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983. doi:10.1016/0378-8733(83)90028-X.
- 25 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020. doi:10.1016/J.JCSS.2019.07.006.