Kernelizing Temporal Exploration Problems

Emmanuel Arrighi
University of Bergen, Norway
University of Trier, Germany

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Petra Wolf
University of Bergen, Norway

Abstract

We study the kernelization of exploration problems on temporal graphs. A temporal graph consists of a finite sequence of snapshot graphs $\mathcal{G} = (G_1, G_2, \ldots, G_L)$ that share a common vertex set but might have different edge sets. The non-strict temporal exploration problem (NS-TEXP for short) introduced by Erlebach and Spooner, asks if a single agent can visit all vertices of a given temporal graph where the edges traversed by the agent are present in non-strict monotonous time steps, i.e., the agent can move along the edges of a snapshot graph with infinite speed. The exploration must at the latest be completed in the last snapshot graph. The optimization variant of this problem is the k-ARB NS-TEXP problem, where the agent's task is to visit at least k vertices of the temporal graph. We show that under standard computational complexity assumptions, neither of the problems NS-TEXP nor k-ARB NS-TEXP allow for polynomial kernels in the standard parameters: number of vertices n, lifetime L, number of vertices to visit k, and maximal number of connected components per time step γ; as well as in the combined parameters $L + k$, $L + \gamma$, and $k + \gamma$. On the way to establishing these lower bounds, we answer a couple of questions left open by Erlebach and Spooner.

We also initiate the study of structural kernelization by identifying a new parameter of a temporal graph $p(\mathcal{G}) = \sum_{i=1}^{L}(|E(G_i)|) - |V(G)| + 1$. Informally, this parameter measures how dynamic the temporal graph is. Our main algorithmic result is the construction of a polynomial (in $p(\mathcal{G})$) kernel for the more general WEIGHTED k-ARB NS-TEXP problem, where weights are assigned to the vertices and the task is to find a temporal walk of weight at least k.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases Temporal graph, temporal exploration, computational complexity, kernel

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.1

Related Version *Full Version*: `https://arxiv.org/abs/2302.10110`

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528).

1 Introduction

We investigate the kernelization of exploration and connectivity tasks in temporal networks. While kernelization, and in particular, structural kernelization, appears to be a successful approach for addressing many optimization problems on *static* graphs, its applications in temporal graphs are less impressive, to say the least. A reasonable explanation for this (we will provide some evidence of that later) is that most of the structural parameters of static graphs, like treewidth, size of a feedback vertex set, or the vertex cover number, do not seem to be useful when it comes to dynamic settings. This brings us to the following question.

What structure of dynamic graphs could be "helpful" for kernelization algorithms?

We propose a new structural parameter of a temporal graph estimating how "dynamic" is the temporal graph. Our main result is an algorithm for the exploration problem on a temporal graph that produces a kernel whose size is polynomial in the new parameter. Before proceeding with the formal statement of the problem and our results, we provide a short overview of temporal graphs and kernelization.

Temporal exploration. Many networks considered nowadays show an inherently dynamic behavior, for instance connectivity in mobile ad-hoc networks, relationships in a social network, or accessibility of services in the internet. Classical graphs do not suffice to model those dynamic behaviors, which led to an intensive study of so-called *temporal graphs.* In general, temporal graphs are graphs that change over time. There are several models in the literature that consider different types of dynamic behaviors and encodings of the temporal graphs. Here, we consider a temporal graph $\mathcal{G}$ to consist of a sequence of *snapshot* graphs $G_1, G_2, \dots, G_L$ that share a common set of vertices V but might have different edge sets $E_1, E_2, \dots, E_L$. We call the graph $G = (V, \bigcup_{i=1}^{L} E_i)$ the *underlying graph* of $\mathcal{G}$. We can think of the temporal graph $\mathcal{G}$ as the graph G where only a subset of edges are present at a certain time step. The number of snapshots L is commonly referred to as the *lifetime* of $\mathcal{G}$. Due to its relevance in modeling dynamic systems, a huge variety of classical graph problems have been generalized to temporal graphs. We refer to [10, 26, 29, 31] for an introduction to temporal graphs and their associated combinatorial problems.

Graph exploration is a fundamental problem in the realm of graph theory and has been extensively studied since its introduction by Shannon in 1951 [32]. The question of whether a given graph can be explored by a single agent was generalized to temporal graphs by Michail and Spirakis [30]. They showed that deciding whether a single agent can visit all vertices of a temporal graph within a given number of time steps while crossing only one edge per time step is an NP-hard problem. This hardness motivated the study of the parameterized complexity of the temporal exploration problem performed by Erlebach and Spooner in [19].

While the temporal exploration problem, TEXP for short, allows the agent to cross only one edge per time step, in some scenarios, the network only changes slowly, way slower than the speed of an agent. Then, it is natural to assume that the agent can travel with infinite speed and explore the whole connected component in which he is currently located in a single time step. Such scenarios arise for instance in delay-tolerant networks [9]. Erlebach and Spooner generalized TEXP to allow the agent to travel with infinite speed in [17] and called this problem the *non-strict temporal exploration problem*, NS-TEXP for short. In NS-TEXP the task is to identify whether an agent can visit all vertices of the graph starting from an initial vertex v according to the following procedure. At step 1, the agent visits all vertices of the connected component C_1 of the snapshot graph G_1 containing v. At step $i \in \{2, \dots, L\}$, the agent selects to explore the vertices of one of the components C_i of G_i that have a non-empty intersection with the component explored by the agent at step $(i-1)$.

Shifting to an infinite exploration speed allows for exploration times that can be significantly shorter than the number of vertices. While for classical graphs, the question if a graph can be explored with infinite speed simply reduces to the question of connectivity for temporal graphs, NS-TEXP is NP-complete [17]. The parameterized complexity of NS-TEXP was studied by Erlebach and Spooner in [19] where FPT algorithms for the parameter *lifetime* L of the temporal graph (equivalently number of time steps during which the temporal exploration must be performed) was shown. They also studied an optimization variant of NS-TEXP, where not all but at least k arbitrary vertices must be visited during the lifetime of the graph. This variant is called k-ARB NS-TEXP. An FPT algorithm with parameter k solving k-ARB NS-TEXP was obtained in [19]. In the long version of this study [20], it was stated as an open question whether NS-TEXP is in FPT or at least in XP for the parameter maximal number of connected components in a time step, γ, and, whether K-ARB NS-TEXP is in FPT for parameter L. These open problems were also stated at the 2022 ICALP satellite workshop *Algorithmic Aspects of Temporal Graphs V*. In this work, we will answer both of these questions negatively.

Kernelization. Informally, a kernelization algorithm is a preprocessing algorithm that consecutively applies various data reduction rules in order to shrink the instance size of a parameterized problem. Kernelization is one of the major research domains of parameterized complexity and many important advances in the area are on kernelization. These advances include general algorithmic findings on problems admitting kernels of polynomial size and frameworks for ruling out polynomial kernels under certain complexity-theoretic assumptions. We refer to the book [22] for an overview of the area. A fruitful approach in kernelization (for static graphs) is the study of the impact of various structural measurements (i.e., different than just the total input size or expected solution size) on the problem complexity. Such structural parameterizations, like the minimum size of a feedback vertex set, of a vertex cover, or the treedepth, measure the non-triviality of the instance [7, 24, 25, 34].

Our contribution. Our focus is on the possibility of kernelization for the problems NS-TEXP and k-ARB NS-TEXP. In their work on the parameterized complexity of these problems, Erlebach and Spooner in [19, 20] established fixed-parameter tractability for some combinations of these problems and "standard" parameterizations like number of vertices n, lifetime L, number of vertices to visit k, and $L + \gamma$, where γ is the maximal number of connected components per time step, see Table 1. As the first step, we rule out the existence of polynomial kernels for both problems when parameterized by each of these standard parameters. Moreover, our lower bounds hold even for "combined" parameters $L + k$, $L + \gamma$, and $k + \gamma$. On the way to establish our lower bounds for kernelization, we resolve two open problems from the parameterized study of NS-TEXP and k-ARB NS-TEXP posed by Erlebach and Spooner in [19, 20]. Namely, we show that NS-TEXP is NP-complete for constant values of $\gamma \geq 5$ and that k-ARB NS-TEXP is W[1]-hard parameterized by L.

This motivates the study of structural kernelization of the exploration problems. While at the first glance, the most natural direction would be to explore the structure of the underlying graph of the temporal graph $\mathcal{G}$, this direction does not seem to bring new algorithmic results. The reason is that our lower bounds on NS-TEXP and k-ARB NS-TEXP hold for temporal graphs with *very* restricted underlying graphs. In particular, we show that the problems remain NP-hard even when the underlying graph is a tree with vertex cover number at most 2. Thus a reasonable structural parameterization, in this case, should capture not only the "static" structure of the underlying graph but also the "dynamics" of the edges in $\mathcal{G}$. With this in mind, we introduce the new parameter of a temporal graph $\mathcal{G} = (G_1, G_2, \ldots, G_L)$, $p(\mathcal{G}) = \sum_{i=1}^{L}(|E(G_i)|) - |V(G)| + 1$.

Note that if $\sum_{i=1}^{L}(|E(G_i)|) < |V(G)| - 1$, then the underlying graph is disconnected and hence the temporal graph $\mathcal{G}$ cannot be explored. The parameter $p(\mathcal{G})$ bounds both the structure and the dynamic of $\mathcal{G}$. Indeed, consider the multigraph on $V(\mathcal{G})$ obtained by inserting a copy of the edge uv for each occurrence of uv in a snapshot. Then, $p(\mathcal{G})$ corresponds to the size of a minimum feedback edge set of this multigraph. As an example,

Table 1 Overview of the parameterized complexity of NS-TEXP and k-ARB NS-TEXP. Our contribution is highlighted in red. The results stating that there is no polynomial kernel rely on the assumption that NP $\not\subseteq$ coNP/poly. Here, for a temporal graph $\mathcal{G}$, n is the number of vertices, L the lifetime, γ is the maximal number of components per time step, and $p = p(\mathcal{G})$. For entries marked with *, we get a compression to the more general variant WEIGHTED k-ARB NS-TEXP.

Param.	NS-TEXP		k-ARB NS-TEXP	
	FPT	Kernel	FPT	Kernel
p	$2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$	$\mathcal{O}(p^4)^\star$	$2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$	$\mathcal{O}(p^4)^\star$
n	$\mathcal{O}^*((2e)^n n^{\log n})$ [20]	no poly kernel	FPT in k [20]	no poly kernel
L	$\mathcal{O}^*(L(L!)^2)$ [19]	no poly kernel	W[1]-hard	no poly kernel
k	-	-	$\mathcal{O}^*((2e)^k k^{\log k})$ [20]	no poly kernel
$L + k$	-	-	FPT in k [20]	no poly kernel
γ	in P for ≤ 2 [20], NP-hard for ≥ 5	-	in P for ≤ 2 [20], NP-hard for ≥ 5	-
$L + \gamma$	FPT in L [19]	no poly kernel for $\gamma \geq 6$	$\mathcal{O}(\gamma^L n^{\mathcal{O}(1)})$	no poly kernel for $\gamma \geq 6$
$k + \gamma$	-	-	FPT in k [20]	no poly kernel

$p(\mathcal{G}) = 0$ means that the underlying graph of $\mathcal{G}$ is a tree and each edge of the underlying graph appears in exactly one snapshot. Hence, our parameter describes how far the graph is from such a tree. Our main result is a polynomial kernel for the more general problem WEIGHTED k-ARB NS-TEXP in the parameter $p = p(\mathcal{G})$. In WEIGHTED k-ARB NS-TEXP, the vertices contain weights and the task is to find a temporal walk that visits vertices with a total sum of weights of at least k for some given integer k. The obtained kernel is of size $\mathcal{O}(p^4)$ and contains a number of vertices that is linear in the parameter p.

Our results are summarized in Table 1. Due to space constraints, the proofs of some results are either sketched or omitted in this extended abstract. The full details could be found in the full arXiv version [3].

Further related work. Michail and Spirakis [30] introduced the TEXP problem and showed that the problem is NP-complete when no restrictions are placed on the input. They proposed considering the problem under the *always-connected* assumption that requires that the temporal graph is connected in every time step. Erlebach et al. [14] followed this proposition and showed that for always-connected temporal graphs, computing a foremost exploration schedule is NP-hard to approximate with ratio $\mathcal{O}(n^{1-\epsilon})$, for every $\epsilon > 0$. Bodlaender and van der Zanden [6] showed that the TEXP problem, when restricted to always-connected temporal graphs whose underlying graph has pathwidth at most 2, remains NP-complete. Bounds on the length of exploration schedules where given in [1, 14, 33] for temporal graphs where the underlying graph has a certain structure, and in [15, 16, 18] for temporal graphs where each snapshot graph has a certain structure.

Erlebach and Spooner [19] studied the TEXP problem from a parameterized perspective. Based on the color coding technique [2], they gave an FPT algorithm parameterized by k for the problems k-ARB TEXP and the non-strict variant k-ARB NS-TEXP. They also gave an FPT algorithm parameterized by the lifetime L of the temporal graph for the problems TEXP and NS-TEXP. In the respective long version [20], Erlebach and Spooner further studied the parameter maximal number of connected components per time step γ and showed that TEXP is NP-hard for $\gamma = 1$, but NS-TEXP is solvable in polynomial time for $\gamma \leq 2$. The NS-TEXP problem was introduced and studied by Erlebach and Spooner [17]. Among other things, they showed NP-completeness of the general problem, as well as $\mathcal{O}(n^{1/2-\epsilon})$ and $\mathcal{O}(n^{1-\epsilon})$-inapproximability for computing a foremost exploration schedule under the assumption that the number of time steps required to move between any pair of vertices is bounded by 2, resp. 3. Bumpus and Meeks [8] considered the parameterized complexity of a graph exploration problem that asks no longer to visit all vertices, but to traverse all *edges* of the underlying graph exactly once. They observed that for natural structural parameters of the underlying graph, the problem does not admit FPT algorithms. Similarly, Kunz et al. [28] obtained several hardness results for structural parameters of the underlying graph when studying the parameterized complexity of the problem of finding temporally disjoint paths and walks, a problem introduced by Klobas et al. [27]. Their obtained W[1]-hardness in the parameter number of vertices, that holds even for instances where the underlying graph is a star, indicates that simply considering structural parameters of the underlying graph is not sufficient to obtain FPT algorithms for temporal graph problems.

2 Preliminaries

Notations. We denote by $\mathbb{Z}$ the set of integers, by $\mathbb{N}$ the set of natural numbers including 0, by $\mathbb{N}_{>0}$ the set of positive integers, and by $\mathbb{Q}$ the set of rational numbers. Let n be a positive integer, we denote with $[n]$ the set $\{1, 2, \dots, n\}$. Given a vector $w = (w_1, w_2, \dots, w_r) \in \mathbb{Q}^r$, we let $\|w\|_\infty = \max_{i \in [r]} |w_i|$ and $\|w\|_1 = \sum_{i \in [r]} |w_i|$. Given $x \in \mathbb{Z}$, we let $\mathsf{sign}(x)$ be $+$ if $x \geq 0$ and $-$ otherwise.

Graphs. We consider a graph $G = (V, E)$ to be a static undirected graph. Given a graph G, we denote by $V(G)$ the set of vertices of G, by $E(G)$ the set of edges of G, by $N_G(x)$ the neighbors of a vertex x in G, and by $\gamma(G)$ the number of connected components of G. Let $G = (V, E)$ be a graph, given a subset of vertices $X \subseteq V(G)$ and a subset of edges $E' \subseteq E$, we define the following operation on G: $G[X] = (X, \{uv \in E \mid u, v \in X\})$, $G - X = G[V \setminus X]$ and $G - E' = (V, E \setminus E')$. We call a *walk* in a graph G an alternating sequence $W = v_0, e_1, v_1, \dots, e_r, v_r$ of vertices and edges, where $v_0, \dots, v_r \in V(G)$, $e_1, \dots, e_r \in E(G)$ and $e_i = v_{i-1}v_i$ for $i \in [r]$; note that W may visit the same vertices and edges several times. A walk without repeated vertices is called a *path*. We say that v_0 and v_r are *end-vertices* of W and W is a (v_0, v_r)*-walk*. We use $V(W) \subseteq V(G)$ to denote the set of vertices of G visited by W and denote by $E(W)$ the set of edges of G that are in W. Given a walk $W = v_0, e_1, v_1, \dots, e_r, v_r$, for $0 \leq i \leq j \leq r$, we call the sequence $W' = v_i, e_{i+1}, \dots, e_j, v_j$ a *subwalk* of W.

Temporal graphs. A *temporal graph* $\mathcal{G}$ over a set of vertices V is a sequence $\mathcal{G} = (G_1, G_2, \dots, G_L)$ of graphs such that for all $t \in [L], V(G_t) = V$. We call L the *lifetime* of $\mathcal{G}$ and for $t \in [L]$, we call $G_t = (V, E_t)$ the *snapshot graph* of $\mathcal{G}$ at *time step* t. We might refer to G_t as $\mathcal{G}(t)$. We call $G = (V, E)$ with $E = \bigcup_{t \in [L]} E_t$ the *underlying graph* of $\mathcal{G}$. We denote by $V(\mathcal{G})$ the set of vertices of $\mathcal{G}$ and by $\gamma(\mathcal{G}) = \max_{t \in L} \gamma(G_t)$ the maximum number of connected components over all snapshot graphs of $\mathcal{G}$. We write V and γ if the graph or temporal graph is clear from the context. For a temporal graph $\mathcal{G} = (G_1, \dots, G_L)$, we define the *total number of edge appearances* as $\mathfrak{m}(\mathcal{G}) = \sum_{i=1}^{L} |E(G_i)|$; we use $\mathfrak{m}$ to denote this value if $\mathcal{G}$ is clear for the context.

In the following, we will be interested in temporal walks where the agent has infinite speed within a snapshot graph. Those temporal walks are called *non-strict temporal walks* in [19]. In [19], a non-strict temporal walk is defined as a sequence of connected components. However, it is more convenient for us to consider a *non-strict temporal walk* as a *monotone* walk in the underlying graph. For a walk $W = v_0, e_1, v_1, \dots, e_r, v_r$ in the underlying graph G of $\mathcal{G} = (G_1, \dots, G_L)$, we say that W is *monotone* if there are $t_1, \dots, t_r \in [L]$ with $1 \leq t_1 \leq \dots \leq t_r \leq L$ such that $e_i \in E(G_{t_i})$ for each $i \in [r]$. The definition of a non-strict temporal walk immediately implies the following observation.

▶ **Observation 1.** *Given a temporal graph $\mathcal{G}$ and a vertex x, $\mathcal{G}$ has a non-strict temporal walk starting in x that visits exactly the vertices of a set X if and only if the underlying graph G has a monotone (x, y)-walk W for some $y \in V(G)$ such that $X = V(W)$.*

For the remainder of this paper, we are mainly interested in the computational problem of finding monotone walks that visit all vertices of a temporal graph. We might also call such a walk an *exploration schedule* or simply an *exploration*.

▶ **Definition 2** (Non-Strict Temporal Exploration (NS-TEXP)).
Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, vertex $v \in V(\mathcal{G})$.
Question: Is there a monotone walk in $\mathcal{G}$ that starts in v and visits all vertices in $V(\mathcal{G})$?

We further consider a more general variant of the NS-TEXP problem, called k-arb NS-TEXP, were we ask for a monotone walk that visits at least k vertices (instead of $|V|$).

▶ **Definition 3** (k-arbitrary Non-Strict Temporal Exploration (k-arb NS-TEXP)).
Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, vertex $v \in V(\mathcal{G})$, integer k.
Question: Is there a monotone walk in $\mathcal{G}$ that starts in v and visits at least k vertices?

We will further generalize the problem by considering the weighted version of k-arb NS-TEXP. We refer to this problem as Weighted k-arb NS-TEXP.

▶ **Definition 4** (Weighted k-arbitrary Non-Strict Temporal Exploration (Weighted k-arb NS-TEXP)).
Input: Temporal graph $\mathcal{G} = (G_1, G_2, \dots, G_L)$, positive-valued weight function $w \colon V(\mathcal{G}) \to \mathbb{N}_{>0}$, vertex $v \in V(\mathcal{G})$, integer k.
Question: Is there a monotone walk W in $\mathcal{G}$ such that W starts in v and $w(V(W)) = \sum_{u \in V(W)} w(u) \geq k$?

Note that k-arb NS-TEXP is a special case of Weighted k-arb NS-TEXP.

Parameterized complexity and kernelization. We refer to books [22] and [12] for an introduction to the field.

A *data reduction rule*, or simply, reduction rule, for a parameterized problem Q is a function $\phi \colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that ϕ is computable in time polynomial in $|I|$ and k. We say that two instances of Q are *equivalent* if the following holds: $(I, k) \in Q$ if and only if $(I', k') \in Q$. We refer to this property of the reduction rule ϕ, that it translates an instance to an equivalent one, as to the *safeness* of the reduction rule.

Informally, *kernelization* is a preprocessing algorithm that consecutively applies various data reduction rules in order to shrink the instance size as much as possible. A preprocessing algorithm takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of Q, works in polynomial in $|I|$

and k time, and returns an equivalent instance (I', k') of Q. The quality of a preprocessing algorithm $\mathcal{A}$ is measured by the size of the output. More precisely, the *output size* of a preprocessing algorithm $\mathcal{A}$ is a function $\text{size}_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined as follows:

$$\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' \ : \ (I', k') = \mathcal{A}(I, k), \ I \in \Sigma^*\}.$$

A *kernelization algorithm*, or simply a *kernel*, for a parameterized problem Q is a preprocessing algorithm $\mathcal{A}$ that, given an instance (I, k) of Q, works in polynomial in $|I|$ and k time and returns an equivalent instance (I', k') of Q such that $\text{size}_{\mathcal{A}}(k) \leq g(k)$ for some computable function $g \colon \mathbb{N} \to \mathbb{N}$. It is said that $g(\cdot)$ is the *size* of a kernel. If $g(\cdot)$ is a polynomial function, then we say that Q admits a *polynomial kernel*. It is well-known that a decidable parameterized problem is FPT if and only if it admits a kernel [13]. However, up to some reasonable complexity assumptions, there are FPT problems that have no polynomial kernels. In particular, we are using the cross-composition technique introduced in [4] and [5] to show that a parameterized problem does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly.

3 Lower Bounds

It was stated as an open problem in [19], whether NS-TEXP is in FPT with parameter γ (i.e. maximum number of components in any snapshot graph). We answer this question negatively by showing that NS-TEXP is NP-complete for $\gamma \geq 5$.

▶ **Theorem 5.** *NS-TEXP is NP-complete for $\gamma \geq 5$.*

Proof. We give a reduction from the satisfiability problem SAT which asks if a given Boolean formula has a satisfying variable assignment. Let $\varphi = \{c_1, c_2, \dots, c_m\}$ be a Boolean formula in conjunctive normal form over the variable set $X = \{x_1, x_2, \dots, x_n\}$. We construct from φ a temporal graph $\mathcal{G}$, where each snapshot graph has 4 or 5 connected components, such that $\mathcal{G}$ has a monotone walk that visits all vertices in $V(\mathcal{G})$ if and only if φ is satisfiable.

The main idea of the construction is the following. In $V(\mathcal{G})$, we have a vertex for each clause, vertices $\widehat{x}_i, x_i, \neg x_i$ for each variable x_i, and one single control vertex $\widehat{c}$. The sequence of snapshot graphs alternates between having four and five connected components. In the case of four connected components, one component collects all clause vertices, one component collects all variable vertices x and $\neg x$, one component collects all not yet processed control vertices $\widehat{x}$ and $\widehat{c}$, and one component collects all processed control vertices $\widehat{x}$. In the case of five connected components, an additional component collects a negative literal $\neg x$ of a variable x together with all clauses containing $\neg x$. In this step, the clauses containing x are incorporated into the variable component, which still contains x. This will allow us to choose a variable assignment with the exploration schedule. In the next time step, and only in this time step, the control vertex $\hat{x}$ is contained in the variable component. For all later time steps, $\hat{x}$ is contained in the component collecting the processed control vertices. Thereby, the control vertices ensure that we return to the component containing the variables in each snapshot consisting of four connected components.

We now give a more formal construction. For this construction, we are interested in the connected components of each snapshot graph but not on the actual structure of the connected components. For simplicity, we define the connected components as set of vertices and assume that each connected component forms a clique. For a subset $S \subseteq V$, we denote by $K(S)$ the set of all possible edges between vertices in S. Thereby, $(S, K(S))$ is a clique. Let $\mathcal{G} = (G_1, G_2, \dots, G_L)$ with $L = 2n + 1$ be the temporal graph constructed from $\varphi = \{c_1, c_2, \ldots, c_m\}$. We define $V(\mathcal{G}) = \{c_1, c_2, \ldots, c_m\} \cup \{x_i, \neg x_i, \widehat{x_i} \mid x_i \in X\} \cup \{\widehat{c}\}$. For $E(G_1) = K(C_{1,1}) \cup K(C_{1,2}) \cup K(C_{1,3})$ we set $C_{1,1} = \{\widehat{c}\} \cup \{\widehat{x_i} \mid x_i \in X\}$, $C_{1,2} = \{c_1, c_2, \ldots, c_m\}$, and $C_{1,3} = \{x_i, \neg x_i \mid x_i \in X\}$. The start vertex is set to x_1.

For the next time step, we define the edges of the snapshot graph as $E(G_2) = K(C_{2,1}) \cup K(C_{2,2}) \cup K(C_{2,3}) \cup K(C_{2,4})$ where $C_{2,1} = C_{1,1} = \{\widehat{c}\} \cup \{\widehat{x_i} \mid x_i \in X\}$, $C_{2,2} = \{c_1, c_2, \ldots, c_m\} \setminus \{c \in \varphi \mid x_1 \in c \vee \neg x_1 \in c\}$, $C_{2,3} = (\{x_i, \neg x_i \mid x_i \in X\} \setminus \{\neg x_1\}) \cup \{c \in \varphi \mid x_1 \in c\}$ and $C_{2,4} = \{\neg x_1\} \cup \{c \in \varphi \mid \neg x_1 \in c\}$. The intuition is that the exploration schedule visits the vertices in $C_{2,3}$ after visiting the vertices in $C_{1,3}$ if the variable x_1 gets assigned with `true` and otherwise, visits $C_{2,4}$ after $C_{1,3}$ if x_1 gets assigned with `false`. Note that no other connected component is reachable from $C_{1,3}$ as only $C_{2,3}$ and $C_{2,4}$ have nonempty intersection with $C_{1,3}$.

In the next time step, we force the exploration schedule to return to the third connected component by passing the control vertex $\widehat{x_1}$ through this connected component. Therefore, we define $E(G_3) = K(C_{3,1}) \cup K(C_{3,2}) \cup K(C_{3,3})$ with $C_{3,1} = (\{\widehat{c}\} \cup \{\widehat{x_i} \mid x_i \in X\}) \setminus \{\widehat{x_1}\}$, $C_{3,2} = \{c_1, c_2, \ldots, c_m\}$, and $C_{3,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{x_1}\}$.

For the remaining time steps, we alternate between the structure of the second and third time step with an additional connected component that collects the already processed control vertices $\widehat{x_i}$. As this connected component is monotone growing, it acts as a sink for the temporal walk, i.e., we could not leave this connected component if we ever enter it. Additionally, the first connected component containing the not yet processed control vertices is monotone shrinking, making it non-accessible from the start vertex of the temporal walk. The idea is now that the last control vertex $\widehat{c}$ will only leave the first component into the variable component in the last time step enforcing us to not go into the sink component of processed control vertices and thereby enforcing the exploration schedule to return to the variable component for each odd time step $t \geq 3$. We formalize this by defining the snapshot graphs G_j for $3 < j < 2n+1$ as follows.

First consider the case that j is even. We define $E(G_j) = K(C_{j,1}) \cup K(C_{j,2}) \cup K(C_{j,3}) \cup K(C_{j,4}) \cup K(C_{j,5})$ with $C_{j,1} = \{\widehat{c}\} \cup \{\widehat{x_i} \mid x_i \in X, i \geq \frac{j}{2}\}$, $C_{j,2} = \{c_1, c_2, \ldots, c_m\} \setminus \{c \in \varphi \mid x_{\frac{j}{2}} \in c \vee \neg x_{\frac{j}{2}} \in c\}$, $C_{j,3} = (\{x_i, \neg x_i \mid x_i \in X\} \setminus \{\neg x_{\frac{j}{2}}\}) \cup \{c \in \varphi \mid x_{\frac{j}{2}} \in c\}$, $C_{j,4} = \{\neg x_{\frac{j}{2}}\} \cup \{c \in \varphi \mid \neg x_{\frac{j}{2}} \in c\}$, and $C_{j,5} = \{\widehat{x_i} \mid x_i \in X, i < \frac{j}{2}\}$.

For the case that j is odd, we define $E(G_j) = K(C_{j,1}) \cup K(C_{j,2}) \cup K(C_{j,3}) \cup K(C_{j,4}) \cup K(C_{j,5})$ with $C_{j,1} = \{\widehat{c}\} \cup \{\widehat{x_i} \mid x_i \in X, i > \frac{j}{2}\}$, $C_{j,2} = \{c_1, c_2, \ldots, c_m\}$, $C_{j,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{x_{\frac{j-1}{2}}}\}$, $C_{j,4} = \emptyset$, and $C_{j,5} = \{\widehat{x_i} \mid x_i \in X, i < \frac{j-1}{2}\}$.

Finally, for the last time step $L = 2n+1$, we define $E(G_L) = K(C_{L,1}) \cup K(C_{L,2}) \cup K(C_{L,3}) \cup K(C_{L,4}) \cup K(C_{L,5})$ with $C_{L,1} = \emptyset$, $C_{L,2} = \{c_1, c_2, \ldots, c_m\}$, $C_{L,3} = \{x_i, \neg x_i \mid x_i \in X\} \cup \{\widehat{c}\}$, $C_{j,4} = \emptyset$, and $C_{j,5} = \{\widehat{x_i} \mid x_i \in X\}$.

Let us now analyze how a potential exploration schedule for $\mathcal{G}$ can look like. Recall that for each time step t, the first connected component $C_{t,1}$ is monotonously shrinking, i.e., for $t, t' \in [L]$ with $t \leq t'$ it holds that $C_{t,1} \supseteq C_{t',1}$. As the start vertex of the exploration schedule is contained in the third connected component, this means that (i) any exploration schedule cannot visit a first connected component as the connected component $C_{t,1}$ never has a non empty intersection with any connected component $C_{t-1,\ell}$ for $1 < t \leq L$, and $\ell \neq 1$.

Further, recall that the fifth connected component $C_{t,5}$ is monotonously growing, i.e., for $t, t' \in [L]$ with $t \leq t'$ it holds that $C_{t,1} \subseteq C_{t',1}$. Therefore, (ii) an exploration schedule cannot leave any fifth connected component.

Now consider the vertex $\widehat{c}$. For all time steps $t \in [L-1]$, this vertex is contained in the first connected component $C_{t,1}$ and therefore not reachable from the start vertex. The only time step in which $\widehat{c}$ is in another connected component is the last time step $t = 2n+1$

in which $\widehat{c}$ is contained in the third connected component $C_{t,3}$ together with the variable vertices. As an exploration schedule need to reach $\widehat{c}$ and $\widehat{c}$ is never contained in a fifth connected component, observation (ii) implies, that we must never enter a fifth connected component. As all elements in $\{\widehat{x_i} \mid x_i \in X\} \cup \{\widehat{x}\}$ do only appear in a first, third or fifth connected component, observation (i) implies that (iii) we need to visit them in a third connected component.

We further need to visit all elements in $\{c_1, c_2, \ldots, c_m\}$. Assume, we visit some of them for the first time in some second connected component in a time step t. By the construction of $\mathcal{G}$, the second connected component is only reachable from a third or fourth connected component in an odd time step. Note that the odd time step t is the only time step in which the control vertex $\widehat{x_{\frac{t-1}{2}}}$ is contained in a third connected component. As by assumption the exploration schedule visits the second connected component in time step t it will not visit $\widehat{x_{\frac{t-1}{2}}}$ in time step t. But observation (iii) implies that then, $\widehat{x_{\frac{t-1}{2}}}$ cannot be reached during the remaining walk. Hence, any exploration schedule cannot visit any second connected component and the vertices $\{c_1, c_2, \ldots, c_m\}$ must be visited in a third or fourth connected component in an even time step.

We already observed that any exploration schedule must be in the third component in any odd time step. From this component, it is only possible to visit some vertex c_j by traversing to a component containing a literal that is contained in c_j. In each even time step t, it is possible to visit those clause vertices c_j that contain a literal from the variable $x_{\frac{t}{2}}$. Thereby, we can either visit those containing the positive literal, or those containing the negative literal. An exploration schedule visiting all clause vertices $\{c_1, c_2, \ldots, c_m\}$ thereby corresponds to a satisfying variable assignment for φ. Conversely, a satisfying variable assignment for φ gives us an exploration schedule for $\mathcal{G}$ by traversing to the component containing the satisfied literal of the variable x_i in time step x_{2i}. ◀

We now shift our focus to restricting the graph class of the underlying graph. We show that NS-TEXP is NP-hard restricted to temporal graphs where the underlying graph is a tree consisting of two stars connected with an edge, or in other words, trees of diameter three. The vertex cover number of such trees is two. With a simple adaptation of the construction, we obtain that NS-TEXP is NP-hard even if every edge appears at most once, or is non-present in at most one time step and the underlying graph is a tree.

▶ **Theorem 6** (⋆). [1] *NS-TEXP is NP-complete for temporal graphs even if:*

- *the underlying graph consists of two stars connected with a bridge, or*
- *every edge of the input temporal graph appears at most once, or*
- *the underlying graph is a tree and every edge is not present in at most one time step*

For the combined parameter $\gamma + L$, there exists a trivial FPT-algorithm for the problem k-ARB NS-TEXP (and so for NS-TEXP) as this parameter bounds the size of a search tree, where γ is the branching factor and L is the depth of the tree. In this tree we consider for each time step all connected components that are reachable from the current connected component, starting with the component containing the start vertex v.

▶ **Observation 7.** *k-ARB NS-TEXP can be solved in $\mathcal{O}(\gamma^L n^{\mathcal{O}(1)})$ time.*

[1] The proofs of the claims marked with (⋆) are omitted in this extended abstract and are available in the full arXiv version [3].

In contrast, using a reduction from Hitting Set, we obtain that NS-TEXP does not have a polynomial kernel in the same parameter $\gamma + L$ unless NP $\subseteq$ coNP/poly. In fact, we show a stronger statement: if γ is constant, there does not exists a polynomial kernel in L.

▶ **Theorem 8** ($\star$). *Unless NP $\subseteq$ coNP/poly, there does not exist a polynomial kernel for NS-TEXP in the parameter L for any constant $\gamma \geq 6$.*

The size of an instance of NS-TEXP can be bounded by $n^2 \times L$ and there exists FPT algorithms in the parameter L as well as in the parameter n [20]. We already showed that under standard complexity assumptions, there does not exists a polynomial kernel for the parameter L. The next result is obtained by a cross-composition from NS-TEXP into itself.

▶ **Theorem 9** ($\star$). *Unless coNP $\subseteq$ NP/poly, NS-TEXP does not admit a polynomial kernel parameterized by n.*

In [20] an FPT-algorithm based on color coding is given for the problem k-arb NS-TEXP with parameter k. We obtain in the following that k-arb NS-TEXP parameterized by L is W[1]-hard by a reduction from multicolored independent set. We thereby answer an open question from [20]. This hardness contrasts the FPT-algorithm in parameter L for the problem NS-TEXP. We further obtain that k-arb NS-TEXP does not admit a polynomial kernel in $L + k$ unless coNP $\subseteq$ NP/poly. This result is based on a cross-composition from the problem k-arb NS-TEXP to itself.

▶ **Theorem 10** ($\star$). *k-arb NS-TEXP is W[1]-hard parameterized by L.*

▶ **Theorem 11** ($\star$). *Unless coNP $\subseteq$ NP/poly k-arb NS-TEXP does not admit a polynomial kernel parameterized by $k + L$.*

4 Polynomial Kernel for Bounded Number of Edge Appearances

In the field of static graphs, trees are nice structures that often lead to efficient algorithms. Building on this fact, structural parameters, like treewidth, defining a distance to trees were successfully introduced and used to design efficient algorithms. Unfortunately, we have seen in Theorem 6 that even when the underlying graph is a tree with vertex cover number 2 the problems we consider remain NP-hard. Therefore, in this section, we introduce a new structural parameter $p(\mathcal{G})$ which, intuitively, characterises how far $\mathcal{G}$ is from being a tree in which each edge appears exactly once. Using this new parameter, we investigate the complexity of k-arb NS-TEXP.

We restate the definition of $p(\mathcal{G})$. Recall that for a temporal graph $\mathcal{G} = (G_1, \ldots, G_L)$, we write $\mathfrak{m} = \sum_{i=1}^{L} |E(G_i)|$. Observe that if $\mathfrak{m} < n - 1$, then the underlying graph G of $\mathcal{G}$ is disconnected. Let X be the set of vertices of the connected component of G containing the source vertex v. Then, k-arb NS-TEXP for $\mathcal{G}$ is equivalent to k-arb NS-TEXP for the temporal graph $\mathcal{G}[X] = (G_1[X], \ldots, G_L[X])$. Hence, we can assume that the underlying graph is connected and $\mathfrak{m} \geq n - 1$. This allows us to consider the above-guarantee parameter $p(\mathcal{G}) = \mathfrak{m} - n + 1$. We show that k-arb NS-TEXP admits a polynomial kernel in $p(\mathcal{G})$ when the underlying graph is connected. In fact, we give a kernel for a more general variant of the problem with vertex weights called Weighted k-arb NS-TEXP.

The starting point of our kernelization algorithm is a polynomial time algorithm for Weighted k-arb NS-TEXP on temporal graphs in which each edge appears exactly once and the underlying graph is a tree (Lemma 14). In addition to this algorithm, the crucial observation that leads to a kernel is that the underlying graph of the input temporal graph

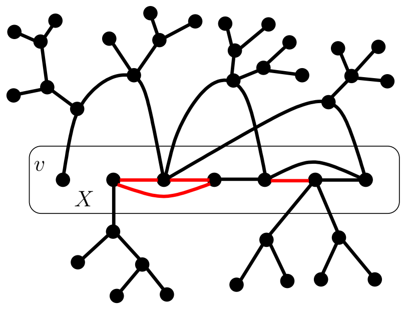

Figure 1 Structure of the underlying graph G of a temporal graph $\mathcal{G}$ when the parameter $p = \mathfrak{m} - n + 1$ is bounded. Here $|X| \leq 4p$. (See Lemma 13.)

has a feedback edge set of bounded size (Lemma 12). By combining this observation and the fact that there is a bounded number of edges repetitions, we show that the input temporal graph $\mathcal{G}$ has some specific structure. Namely, there exists a core set $X \subseteq V(\mathcal{G})$ of vertices of bounded size such that the underlying graph of the remaining graph $\mathcal{G} - X$ is a forest $\mathcal{F}$ with the following property. In $\mathcal{G} - X$, each edge appears exactly once and each tree of $\mathcal{F}$ is connected to X by at most two edges (see Figure 1 and Lemma 13). For each tree T in $\mathcal{F}$, depending on its structure and its interaction with X, we are able to describe all possible ways an exploration can visit some of the vertices of T. Using the polynomial time algorithm for trees in which edges appear only once, we can then compute the maximum weight contributed by the vertices in T for each of those cases and design a gadget of constant size that simulates the original tree T. Thereby, the gadget keeps the information of how many vertices, respectively vertex weights, were reachable in T and is the reason why we need to work with the weighted version of k-ARB-NS-TEXP. This gives us several reduction rules (Reduction Rule 1-4). To conclude the kernel, we show that after applying these reduction rules exhaustively, the obtained temporal graph has linear size in p (Claim 18). Lastly, by using an algorithm proposed by Frank and Tardos (Proposition 15), we can bound the size of the obtained weights on the vertices by $\mathcal{O}(p^4)$.

Throughout this section we assume that the temporal graphs under consideration have connected underlying graphs. Let $\mathcal{G} = (G_1, \ldots, G_L)$ be a temporal graph and G its underlying graph. For $e \in E(G)$, we denote by $A(e) = \{t \colon e \in E(G_t) \text{ for } 1 \leq t \leq L\}$ and set $m(e) = |A(e)|$. Note that $\mathfrak{m}(\mathcal{G}) = \sum_{e \in E(G)} m(e)$.

By Observation 1, an instance $(\mathcal{G}, w, v, k)$ of WEIGHTED k-ARB NS-TEXP is a yes-instance if and only if the underlying graph G has a monotone (v, x)-walk W for some $x \in V(G)$ such that $w(V(W)) \geq k$. Slightly abusing notation, we write $w(W)$ instead of $w(V(W))$ for the total weight of vertices visited by a walk.

We start by showing that the underlying graph has a feedback edge set of bounded size. Recall that a set of edges $S \subseteq E(G)$ is a *feedback edge set* of a graph G, if $G - S$ is a forest. Let $\mathcal{G} = (G_1, \ldots, G_L)$ be a temporal graph and let $p = \mathfrak{m} - n + 1$. We denote by R the set of edges of the underlying graph G appearing at least twice in $\mathcal{G}$. We refer to the edges of R as *red*. We set $B = E(G) \setminus R$ and call the edges of B *blue*.

▶ **Lemma 12** ($\star$). *Let $\mathcal{G}$ be a temporal graph and let $p = \mathfrak{m} - n + 1$. Then, the underlying graph G of $\mathcal{G}$ has a feedback edge set S of size at most p such that every red edge is in S.*

Lemma 12 implies that G has a special structure that is used in our algorithm (see Figure 1).

▶ **Lemma 13.** *Let $\mathcal{G}$ be a temporal graph with the underlying graph G and let $p = \mathfrak{m} - n + 1$. Let also $v \in V(G)$. Then, there is a set of vertices $X \subseteq V(G)$ of size at most $4p$ such that (i) $v \in X$, (ii) every red edge is in $G[X]$, (iii) $G - E(G[X])$ is a forest and (iv) each connected component T of $G - X$ is a tree with the properties that each vertex $x \in X$ has at most one neighbor in T and*

(a) *either X has a unique vertex x that has a neighbor in T,*

(b) *or X contain exactly two vertices x and y having neighbors in T.*

Furthermore, if q is the number of connected components T of $G - X$ satisfying (b), then $q \leq 4p - 1$ and $G[X]$ has at most $5p - q - 1$ edges.

Proof. By Lemma 12, G has a feedback edge set S of size at most p containing all red edges. We define $Y = \{v\} \cup \{x \in V(G) \mid x \text{ is an endpoint of an edge of } S\}$. Note that $|Y| \leq 2p + 1$. Consider the graph G' obtained from G by the iterative deletion of vertices of degree one that are not in Y. Because S is a feedback edge set, we obtain that $H = G' - S$ is a forest such that every vertex of degree at most one is in Y. It is a folklore knowledge that any tree with $\ell \geq 2$ vertices of degree one (leaves) has at most $\ell - 2$ vertices of degree at least three. This implies that the set Z of vertices of H with degree at least three has size at most $2p - 1$. We define $X = Y \cup Z$. By the construction, $|X| \leq 4p$ and (i)–(iv) are fulfilled.

Let q be the number of connected components T of $G - X$ satisfying (b). Consider the multigraph G'' obtained from $G[X]$ by the following operation: for each connected component T of $G - X$ satisfying (b), where x and y are the vertices of X having neighbors in T, add the edge xy to $G[X]$ and make it a multi-edge if $xy \in E(G[X])$. Because of (iii), $G'' - E(G[X])$ is a tree on $|X|$ vertices, therefore, we have that $q \leq |X| - 1 \leq 4p - 1$. Because G has a feedback edge set of size at most p, G'' has the same property. Therefore, G'' has at most $|X| - 1 + p \leq 5p - 1$ edges. Hence, $G[X]$ has at most $5p - q - 1$ edges. ◀

We show that Weighted k-arb NS-TEXP can be solved in polynomial time if each edge appears exactly once and the underlying graph is a tree. If an edge e appears exactly once, then we use $t(e)$ to denote the unique element of $A(e)$.

▶ **Lemma 14.** *There is an algorithm running in $\mathcal{O}(n + L)$ time that, given a temporal graph $\mathcal{T} = (F_1, \dots, F_L)$ such that its underlying graph T is a tree with $m(e) = 1$ for each $e \in E(T)$, a weight function $w\colon V(T) \to \mathbb{N}_{>0}$ and two vertices $x, y \in V(T)$, either finds the maximum weight of a monotone (x, y)-walk W in T or reports that such a walk does not exist.*

Proof sketch. The algorithm works as follow. Consider the unique (x, y)-path P in T. Let $P = v_0, e_1, v_1, \dots, e_r, v_r$ in T with $x = v_0$ and $y = v_r$. As each edge e can only be crossed in the single time step $t(e)$, there exists a monotone (x, y)-walk W in T if and only if P is monotone. Suppose P is monotone. For $i \in [r]$, let $t_i = t(e_i)$, $t_0 = 0$ and $t_{r+1} = L$. Let $i \in \{0, \dots, r\}$. Because edges cannot be crossed in two different time steps, any (x, y)-walk needs to be on v_i between time step t_i and t_{i+1}. Therefore, for any time step $t_i \leq t \leq t_{i+1}$, an (x, y)-walk can only visit the vertices of $C_t(v_i)$, where $C_t(v_i)$ correspond to the connected components of F_t containing v_i, and the set $U(v_i) = \bigcup_{t_i \leq t \leq t_{i+1}} V(C_t(v_i))$ corresponds to all vertices that can be visited while waiting on v_i. Thereby, a (x, y)-walk of maximum weight visits exactly the vertices of $U = \cup_{i \in \{0,\dots r\}} U(v_i)$. By the preconditions, the number of edges over all snapshots is $n - 1$. By using a BFS we can find the connected component containing a specific vertex in time linear in the size of the component. Therefore, we can construct U and compute its weight in time $\mathcal{O}(n + L)$. ◀

To reduce the vertex weights in our kernelization algorithm, we use the approach proposed by Etscheid et al. [21] that is based on the result of Frank and Tardos [23].

▶ **Proposition 15** ([23]). *There is an algorithm that, given a vector* $\mathsf{w} \in \mathbb{Q}^r$ *and an integer* N*, in polynomial time finds a vector* $\overline{\mathsf{w}} \in \mathbb{Z}^r$ *with* $\|\overline{\mathsf{w}}\|_\infty \le 2^{4r^3} N^{r(r+2)}$ *such that* $\mathsf{sign}(\mathsf{w} \cdot b) = \mathsf{sign}(\overline{\mathsf{w}} \cdot b)$ *for all vectors* $b \in \mathbb{Z}^r$ *with* $\|b\|_1 \le N - 1$*.*

Now we are ready to prove the main result of the section. We recall that a pair (P, Q) of subsets of $V(G)$ is called a *separation* of G if $P \cup Q = V(G)$ and there is no edge xy with $x \in P \setminus Q$ and $y \in Q \setminus P$. The set $P \cap Q$ is a *separator* and $|P \cap Q|$ is called the *order* of a separation (P, Q). If a separation has order one, then the single vertex of $P \cap Q$ is called a *cut-vertex* and we say that this vertex is a separator. Notice that it may happen that $P \subseteq Q$ or $Q \subseteq P$ and we slightly abuse the standard notation, as in these cases necessarily $P \cap Q$ does not separate G. We also recall that an edge cut of G is a partition (P, Q) of $V(G)$ and the edge cut-set is the set of all edges xy for $x \in P$ and $y \in Q$.

▶ **Theorem 16.** *Weighted k-arb NS-TEXP parameterized by* $p = \mathfrak{m} - n + 1$ *admits a kernel of size* $\mathcal{O}(p^4)$ *for connected underlying graphs such that for the output instance* $(\mathcal{G} = (G_1, \ldots, G_L), w, v, k)$*,* $\mathcal{G}$ *has* $\mathcal{O}(p)$ *vertices and edges, and* $L \in \mathcal{O}(p)$*.*

Proof sketch. Let $(\mathcal{G} = (G_1, \ldots, G_L), w, v, k)$ be an instance of Weighted k-arb NS-TEXP and let $p = \mathfrak{m} - n + 1$. Let also G be the underlying graph of $\mathcal{G}$ and let R be the set of red edges. We describe our kernelization algorithm as a series of reduction rules that are applied exhaustively whenever it is possible to apply a rule. The rules modify $\mathcal{G} = (G_1, \ldots, G_L)$ and w. Whenever we say that a rule deletes a vertex x, this means that x is deleted from $G_1, \ldots, G_L$ and G together with the incident edges. Similarly, whenever we create a vertex, this vertex is added to every graph G_i for $i \in [L]$ and G. When we either delete or add an edge, we specify G_i where this operation is performed and also assume that the corresponding operation is done for G.

To describe the first rule, we need some auxiliary notation. Let (P, Q) be a separation of G of order one with a cut-vertex x. We say that (P, Q) is *important* if

(i) $G[P]$ is a tree with at least two vertices,
(ii) the start vertex v is in Q and $R \subseteq E(G[Q])$, and
(iii) P is an inclusion-maximal set satisfying (i) and (ii).

Suppose that (P, Q) is an important separation of G with a cut-vertex x. Let $a, b \in [L]$. Then, there is a separation (P_1, P_2) of $H = G[P]$ with x being the cut-vertex such that for any $y \in N_H(x)$, $y \in P_1$ if and only if $a \le t(xy) \le b$ and, moreover, (P_1, P_2) is unique. We use $\mathcal{T}_{[a,b]}$ to denote the temporal graph $(G_1[P_1], \ldots, G_L[P_1])$ and $T_{[a,b]} = G[P_1]$. Observe that $\mathcal{T}_{[a,b]}$ may be a single-vertex temporal graph containing only x. We write $\mathcal{T}_{(a,b)}$ ($\mathcal{T}_{[a,b)}$ and $\mathcal{T}_{(a,b]}$, respectively) for $\mathcal{T}_{[a+1,b-1]}$ ($\mathcal{T}_{[a,b-1]}$ and $\mathcal{T}_{[a+1,b]}$, respectively) and we use the corresponding notation for the underlying trees. We also write $\mathcal{T}_a$ and T_a instead of $\mathcal{T}_{[a,a]}$ and $T_{[a,a]}$. Given an important separation (P, Q) of G with a cut-vertex x, we denote $A(Q) = \bigcup_{y \in N_{G[Q]}(x)} A(xy)$. We say that $i, j \in A(Q)$ are *consecutive* if $i < j$ and there is no $t \in A(Q)$ such that $i < t < j$.

Observe that as cut-vertices and blocks of a graph can be found in linear time by standard algorithmic tools (see, e.g., [11]), all important separations can be listed in linear time.

Given a cut vertex x such that one side T of the important separation is a tree, then an optimal exploration can either enter and exit T by x or end in some vertex of T. In both cases, using the algorithm from Lemma 14 we can compute the optimal exploration of T and replace $T - x$ by 2 vertices simulating those two options giving us the first reduction rule.

▶ **Reduction Rule 1.** *If there is an important separation (P, Q) of G with a cut-vertex x such that either there is $i \in A(Q)$ such that $\mathcal{T} = \mathcal{T}_i$ has at least four vertices, or there are consecutive $i, j \in A(Q)$ such that $\mathcal{T} = \mathcal{T}_{(i,j)}$ has at least four vertices, or $\mathcal{T} = \mathcal{T}_{[1,i)}$ has at least four vertices for $i = \min A(Q)$, or $\mathcal{T} = \mathcal{T}_{(j,L]}$ has at least four vertices for $j = \max A(Q)$, then do the following for $\mathcal{T}$ and the underlying tree T (see Figure 2):*

- *Call the algorithm from Lemma 14 and compute the maximum weight w_1 of a monotone (x, x)-walk in T and the maximum weight w_2 of a monotone (x, y)-walk, where maximum is taken over all $y \in V(T)$. Set $t = \min_{u \in N_T(x)} t(xu)$.*
- *Delete the vertices of $\mathcal{T}$ in $\mathcal{G}$ except x, create a new vertex y, make it adjacent to x in G_t and set $w(y) = w_1 - w(x)$.*
- *If $w_2 > w_1$, then create a new vertex z, make it adjacent to y in G_L and set $w(z) = w_2 - w_1$.*

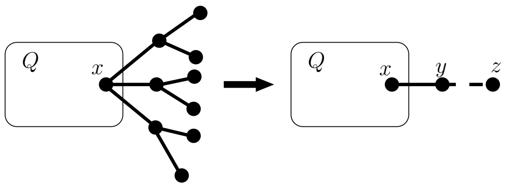

Figure 2 The application of Reduction Rule 1.

To state the next rules, we define edge cuts that are important for our algorithm. We say that an edge cut (P, Q) of G is *important* if

(i) the edge-cut set is of size two and consists of edges with distinct endpoints,
(ii) $G[P]$ is a tree, and
(iii) $v \in Q$ and $R \subseteq E(G[Q])$.

Given an important edge cut (P, Q) with an edge cut-set $\{x_1y_1, x_2y_2\}$, where $x_1, x_2 \in P$, there is the unique (x_1, x_2)-path S in the tree $G[P]$. We say that the path $y_1, y_1x_1, S, x_2y_2, y_2$ is the *backbone* of the edge cut. We also denote $\mathcal{T}_{P,Q} = (G_1[(P \cup \{y_1, y_2\})] - y_1y_2, \ldots, G_L[P \cup \{y_1, y_2\}] - y_1y_2)$. Note that the underlying graph T of $\mathcal{T}_{P,Q}$ is a tree and the unique (y_1, y_2)-path in T is the backbone.

Observe that all important edge cuts can be found in polynomial time. The next three reduction rules reduce the length of backbones. We begin by deleting irrelevant edges.

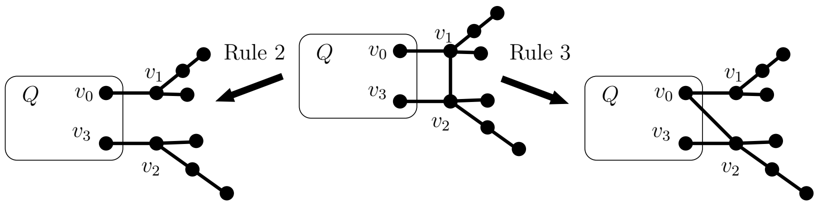

Figure 3 The application of Reduction Rule 2 and Reduction Rule 3.

▶ **Reduction Rule 2.** *If there is an important edge cut (P, Q) of G with a backbone $v_0, e_1, v_1, e_2, v_2, e_3, v_3$ such that $t(e_1) > t(e_2)$ and $t(e_3) > t(e_2)$, then modify $\mathcal{G}$ and G by deleting e_2 from $G_{t(e_2)}$ (see Figure 3). Furthermore, if the obtained underlying graph G is disconnected, then delete the vertices of the (unique) connected component that does not contain the source vertex v.*

The next rule aims at consecutive edges in backbones that occur in the same G_t.

▶ **Reduction Rule 3.** *If there is an an important edge cut (P, Q) of G with a backbone $v_0, e_1, v_1, e_2, v_2, e_3, v_3$ such that $t(e_1) = t(e_2)$, then modify $\mathcal{G}$ and G by deleting e_2 from $G_{t(e_1)}$ and adding v_0v_2 in $G_{t(e_1)}$ (see Figure 3).*

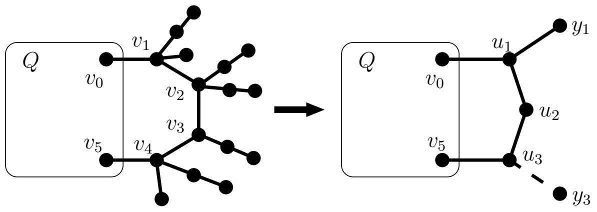

Figure 4 The application of Reduction Rule 4.

It remains to shorten monotone backbones.

▶ **Reduction Rule 4.** *If there is an important edge cut (P, Q) of G with a backbone $v_0, e_1, \dots, e_5, v_5$ such that $t(e_1) < \dots < t(e_5)$, then modify $\mathcal{G}$ and G by doing the following for $\mathcal{T} = \mathcal{T}_{P,Q}$ with its underlying tree T (see Figure 4):*

- *Call the algorithm from Lemma 14 and compute the maximum weight w_1 of a monotone (v_0, v_0)-walk in T, the maximum weight w_2 of a monotone (v_0, v_5)-walk and the maximum weight w_3 of a monotone (v_5, v_5)-walk.*
- *Delete the vertices of $V(T) \setminus \{v_0, v_5\}$, create new vertices u_1, u_2, u_3, make u_1 adjacent to v_0 in $G_{t(e_1)}$, make u_2 adjacent to u_1 and u_3 in $G_{t(e_2)}$ and $G_{t(e_4)}$, respectively, and make u_3 adjacent to v_5 in $G_{t(e_5)}$.*
- *Set $w(u_1) = w_1 - w(v_0)$, $w(u_3) = w_3 - w(v_5)$ and $w(u_2) = w_2 - w_1 - w_3$.*
- *Call the algorithm from Lemma 14 and compute the maximum weight w_0 of a monotone (v_0, u)-walk in $T - v_5$, where the maximum is taken over all $u \in V(T) \setminus \{v_5\}$. Create a vertex y_1, make y_1 adjacent to u_1 in $G_{t(e_3)}$ and set $w(y_1) = w_0 - w(v_0) - w(u_1) - w(u_2)$.*
- *Call the algorithm from Lemma 14 and compute the maximum weight w_5 of a monotone (v_5, u)-walk in T, where the maximum is taken over all $u \in V(T)$. If $w_5 > w_3$, then create a vertex y_3, make y_3 adjacent to u_3 in G_L and set $w(y_3) = w_5 - w(v_5) - w(u_3)$.*

We observe that by the definitions of Rules 1–4, we immediately obtain the following claim.

▷ Claim 17. Rules 1–4 do not increase the parameter $p = \mathfrak{m} - n + 1$.

After applying Rules 1–4, the graph has bounded size.

▷ Claim 18. If Rules 1–4 are not applicable, then $|V(\mathcal{G})| \leq 324p$ and $|E(\mathcal{G})| \leq 326p$.

The bound on the number of edges of G allows to reduce the value of L.

▶ **Reduction Rule 5.** *If there is $t \in [L]$ such that G_t has no edge, then delete G_t from $\mathcal{G}$.*

After applying this rule $L \leq 326p$ as $|E(\mathcal{G})| \leq 326p$.

Our last aim is to reduce the weights. For this, we use the algorithm from Proposition 15. Let $n = |V(G)|$ and let $V(G) = \{v_1, \dots, v_n\}$. We define $r = n + 1$ and consider the vector $\mathsf{w} = (w_0, w_1, \dots, w_n)^{\intercal} \in \mathbb{Z}^r$, where $w_0 = k$ and $w_i = w(v_i)$ for $i \in [n]$.

▶ **Reduction Rule 6.** *Apply the algorithm from Proposition 15 for w and $N = r + 1$ and find the vector $\overline{\mathsf{w}} = (\overline{w}_0, \dots, \overline{w}_n)$. Set $k := \overline{w}_0$ and set $w(v_i) := \overline{w}_i$ for $i \in [n]$.*

To see that the rule is safe, let $k' = \overline{\mathsf{w}}_0$ and let $w'(v_i) = \overline{w}_i$ for $i \in [n]$. Note that by the choice of N, for each vector $b \in \{-1,0,1\}^r$, we have that $\mathsf{sign}(\mathsf{w} \cdot b) = \mathsf{sign}(\overline{\mathsf{w}} \cdot b)$. This implies that the new weights $w'(x)$ and k' are positive integers and for every $U \subseteq V(G)$, $w(U) \geq k$ if and only if $w'(U) \geq k'$.

By Proposition 15, we obtain that $k \leq 2^{4n^3}(n+2)^{(n+1)(n+2)}$ and the same upper bound holds for $w(x)$ for every $x \in V(G)$. Because $|V(G)| = \mathcal{O}(p)$, we have that we need $\mathcal{O}(p^3)$ bits to encode k and the weight of each vertex. Then, the total bit-length of the encoding of the weights and k is $\mathcal{O}(p^4)$. Taking into account that $|V(G)| = \mathcal{O}(p)$, $|E(G)| = \mathcal{O}(p)$ and $L = \mathcal{O}(p)$, we conclude that we obtained a kernel of size $\mathcal{O}(p^4)$.

Because important separations and important edge cuts can be found in polynomial time and the algorithm from Lemma 14 is polynomial, we have that Rules 1–4 can be exhaustively applied in polynomial time. It is trivial that Reduction Rule 5 can be applied in polynomial time. Because the algorithm from Proposition 15 is polynomial, Reduction Rule 6 requires polynomial time. Therefore, the overall running time of our kernelization algorithm is polynomial. This concludes the proof. ◀

In [20], Erlebach and Spooner proved that NS-TEXP can be solved in $\mathcal{O}(2^n L n^3)$ time. This fact together with Theorem 16 implies the following corollary.

▶ **Corollary 19.** *Weighted k-arb NS-TEXP parameterized by $p = \mathfrak{m} - n + 1$ can be solved in $2^{\mathcal{O}(p)}(nL)^{\mathcal{O}(1)}$ time on temporal graphs with connected underlying graphs.*

Conclusion

We initiated the study of polynomial kernels for exploration problems on temporal graphs. We showed that for the problems NS-TEXP, k-arb NS-TEXP, and Weighted k-arb-NS-TEXP, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, there does not exist a polynomial kernel for the parameters number of vertices n, lifetime L, and number of vertices to visit k; and for the combined parameters $L+k$, $L+\gamma$, and $k+\gamma$, where γ is the maximal number of connected components per time step. In fact, by a straight forward reduction that repeats each snapshot sufficiently many times, all of our hardness results, that do not involve the parameter L, carry over to the strict settings TEXP and k-arb TEXP where a temporal exploration traverses at most one edge per time step. We showed that the temporal exploration problems remain NP-hard restricted to temporal graphs where the underlying graph is a tree of diameter three. From a parameterized complexity point of view, this eliminates most of the common structural parameters considered on the underlying graph. Nonetheless, we were able to identify a new parameter of a temporal graph $p(\mathcal{G}) = \sum_{i=1}^{L}(|E(G_i)|) - |V(G)| + 1$ that captures how close the temporal graph is to a tree where each edge appears exactly once. Our parameter can also be seen as a notion of sparsity for temporal graphs. For this parameter $p(\mathcal{G})$, we were able to obtain a polynomial kernel for Weighted k-arb NS-TEXP. Using simplified reduction rules, we can obtain a kernel of linear size for NS-TEXP. While the reduction from NS-TEXP to TEXP blows up the parameter $p(\mathcal{G})$, our reduction rules can still be adapted to obtain polynomial kernels for the strict variants of the considered exploration problems. The natural next step would be to evaluate how useful our parameter is for other problems on temporal graphs.

References

1 Duncan Adamson, Vladimir V. Gusev, Dmitriy S. Malyshev, and Viktor Zamaraev. Faster exploration of some temporal graphs. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 5:1–5:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.5.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Emmanuel Arrighi, Fedor V. Fomin, Petr A. Golovach, and Petra Wolf. Kernelizing temporal exploration problems. *CoRR*, abs/2302.10110, 2023. doi:10.48550/arXiv.2302.10110.

4 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

6 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Inf. Process. Lett.*, 142:68–71, 2019. doi:10.1016/j.ipl.2018.10.016.

7 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.

8 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, pages 1–29, 2022.

9 Sobin C. C., Vaskar Raychoudhury, Gustavo Marfia, and Ankita Singla. A survey of routing and data dissemination in delay tolerant networks. *J. Netw. Comput. Appl.*, 67:128–146, 2016. doi:10.1016/j.jnca.2016.01.002.

10 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012. doi:10.1080/17445760.2012.668546.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

12 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.

13 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1999.

14 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.

15 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.141.

16 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.36.

17 Thomas Erlebach and Jakob T. Spooner. Non-strict temporal exploration. In Andrea Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2020. doi:10.1007/978-3-030-54921-3_8.

18 Thomas Erlebach and Jakob T. Spooner. Exploration of k-edge-deficient temporal graphs. *Acta Informatica*, 59(4):387–407, 2022. doi:10.1007/s00236-022-00421-5.

19 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, volume 221 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.15.

20 Thomas Erlebach and Jakob T. Spooner. Parameterized temporal exploration problems. *CoRR*, abs/2212.01594, 2022. doi:10.48550/arXiv.2212.01594.

21 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *J. Comput. Syst. Sci.*, 84:1–10, 2017. doi:10.1016/j.jcss.2016.06.004.

22 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing.* Cambridge University Press, 2019.

23 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

24 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory of Computing Systems*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

25 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Wlodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.

26 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.

27 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: temporally disjoint paths. *Auton. Agents Multi Agent Syst.*, 37(1):1, 2023. doi:10.1007/s10458-022-09583-5.

28 Pascal Kunz, Hendrik Molter, and Meirav Zehavi. In which graph structures can we efficiently find temporally disjoint paths and walks? *CoRR*, abs/2301.10503, 2023. doi:10.48550/arXiv.2301.10503.

29 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.

30 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

31 Polina Rozenshtein and Aristides Gionis. Mining temporal networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 3225–3226. ACM, 2019. doi:10.1145/3292500.3332295.

32 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.

33 Shadi Taghian Alamouti. Exploring temporal cycles and grids. Master's thesis, Concordia University, 2020.

34 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theor. Comput. Sci.*, 494:99–111, 2013. doi:10.1016/j.tcs.2013.01.029.