Parameterized Temporal Exploration Problems

Thomas Erlebach
Department of Computer Science, Durham University, UK

Jakob T. Spooner
School of Computing and Mathematical Sciences, University of Leicester, UK

Abstract

In this paper we study the fixed-parameter tractability of the problem of deciding whether a given temporal graph $\mathcal{G}$ admits a temporal walk that visits all vertices (temporal exploration) or, in some problem variants, a certain subset of the vertices. Formally, a temporal graph is a sequence $\mathcal{G} = \langle G_1, ..., G_L \rangle$ of graphs with $V(G_t) = V(G)$ and $E(G_t) \subseteq E(G)$ for all $t \in [L]$ and some underlying graph G, and a temporal walk is a time-respecting sequence of edge-traversals. For the strict variant, in which edges must be traversed in strictly increasing timesteps, we give FPT algorithms for the problem of finding a temporal walk that visits a given set X of vertices, parameterized by $|X|$, and for the problem of finding a temporal walk that visits at least k distinct vertices in V, parameterized by k. For the non-strict variant, in which an arbitrary number of edges can be traversed in each timestep, we parameterize by the lifetime L of the input graph and provide an FPT algorithm for the temporal exploration problem. We also give additional FPT or W[2]-hardness results for further problem variants.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Temporal graphs, fixed-parameter tractability, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.15

Funding *Thomas Erlebach*: Supported by EPSRC grants EP/S033483/2 and EP/T01461X/1.

1 Introduction

The problem of computing a series of consecutive edge-traversals in a static (i.e., classical discrete) graph G, such that each vertex of G is an endpoint of at least one traversed edge, is a fundamental problem in algorithmic graph theory, and an early formulation was provided by Shannon [26]. Such a sequence of edge-traversals might be referred to as an *exploration* or *search* of G and, from a computational standpoint, it is easy to check whether a given graph G admits such an exploration and easy to compute one if the answer is yes – we simply carry out a depth-first search starting at an arbitrary start vertex in $V(G)$ and check whether every vertex of G is reached. We consider in this paper a decidedly more complex variant of the problem, in which we try to find an exploration of a *temporal graph*. A temporal graph $\mathcal{G} = \langle G_1, ..., G_L \rangle$ is a sequence of static graphs G_t such that $V(G_t) = V(G)$ and $E(G_t) \subseteq E(G)$ for any *timestep* $t \in [L]$ and some fixed *underlying graph* G.

A concerted effort to tackle algorithmic problems defined for temporal graphs has been made in recent years. With the addition of time to a graph's structure comes more freedom when defining a problem. Hence, many studies have focused on temporal variants of classical graph problems; for example, the travelling salesperson problem [21]; shortest paths [27]; vertex cover [3]; maximum matching [20]; network flow problems [1]; and a number of others. For more examples, we point the reader to the works of Molter [23] or Michail [21]. One seemingly common trait of the problems that many of these studies consider is the following: *Problems that are easy for static graphs often become hard on temporal graphs, and hard problems for static graphs remain hard on temporal graphs.* This certainly holds true for the problem of deciding whether a given temporal graph $\mathcal{G}$ admits a *temporal walk* W – roughly speaking, a sequence of edges traversed consecutively and during strictly increasing timesteps – such that every vertex of $\mathcal{G}$ is an endpoint of at least one edge of W (any temporal walk with this property is known as an *exploration schedule*). Indeed, Michail and Spirakis [22] showed that this problem, Temporal Exploration or TEXP for short, is NP-complete. In this paper, we consider variants of the TEXP problem from a fixed-parameter perspective and under both *strict* and *non-strict* settings. More specifically, we consider problem variants in which we look for *strict* temporal walks that traverse each consecutive edge at a timestep strictly larger than the previous, as well as variants that ask for *non-strict* temporal walks that allow an unlimited but finite number of edges to be traversed in each timestep.

Contribution. In Section 2 we prove FPT-membership for two natural parameterized variants of TEXP. Firstly, we parameterize by the size k of a fixed subset of the vertex set and ask for an exploration schedule that visits at least these vertices, providing a $O(2^k kLn^2)$-time algorithm. Secondly, we parameterize by only an integer k and ask that a computed solution visits at least k arbitrary vertices – in this case we specify, for any $\varepsilon > 0$, a randomized algorithm (based on the colour-coding technique first introduced by Alon et al. [4]) with running time $O((2e)^k Ln^3 \log \frac{1}{\varepsilon})$. A now-standard derandomization technique due to Naor et al. [24] is then utilized in order to obtain a deterministic $(2e)^k k^{O(\log k)} Ln^3 \log n$-time algorithm.

In Section 3, we consider the non-strict variant known as Non-Strict Temporal Exploration, or NS-TEXP, which was introduced in [17]. Here, a candidate exploration schedule is permitted to traverse an unlimited but finite number of edges during each timestep, and it is not too hard to see that this change alters the problem's structure quite drastically (more details in Section 3). We therefore use a different model of temporal graphs to the one considered in Section 2, which we properly define later. For this problem, we parameterize by the length L of the sequence of static graphs that comprises our input temporal graph, and provide an $O(L(L!)^2 n)$-time recursive search-tree algorithm. We also consider a generalized variant, Set NS-TEXP, in which we are supplied with m subsets of the input temporal graph's vertex set and are asked to decide whether there exists a non-strict temporal walk that visits at least one vertex belonging to each set; this problem is shown to be $W[2]$-hard via a reduction from Set Cover.

Related work. We refer the interested reader to Casteigts et al. [11] for a study of various models of dynamic graphs, and to Michail [21] for an introduction to temporal graphs and some of their associated combinatorial problems. Brodén et al. [8] consider the Temporal Travelling Salesperson Problem for complete temporal graphs with n vertices. The costs of edges are allowed to differ between 1 and 2 in each timestep. They show that when an edge's cost changes at most k times during the input graph's lifetime, the problem is NP-complete, but provide a $(2 - \frac{2}{3k})$-approximation. For the same problem, Michail and Spirakis [22] prove APX-hardness and provide a $(1.7 + \epsilon)$-approximation. Bui-Xuan et al. [9] propose multiple objectives for optimisation when computing temporal walks/paths: e.g., *fastest* (fewest number of timesteps used) and *foremost* (arriving at the destination at the earliest time possible). Michail and Spirakis [22] introduced the TEXP problem, which asks whether or not a given temporal graph admits a temporal walk that visits all vertices at least once. The problem is shown to be NP-complete when no restrictions are placed on the input, and they propose considering the problem under the *always-connected* assumption as a means of ensuring that exploration is possible (provided the lifetime of the input graph is sufficiently long). Erlebach et al. [16] consider the problem of computing foremost exploration schedules under the always-connected assumption, proving $O(n^{1-\varepsilon})$-inapproximability (for any $\varepsilon > 0$) amongst other results. Bodlaender and van der Zanden [6] examined the TEXP problem when restricted to temporal graphs whose underlying graph has pathwidth at most 2 and that are connected in each timestep, showing the problem to be NP-complete in this case. Akrida et al. [2] consider a TEXP variant called Return-To-Base TEXP, in which the underlying graph is a star and a candidate solution must return to the vertex from which it initially departed (the star's centre). They prove various hardness results and provide polynomial-time algorithms for some special cases. Casteigts et al. [12] studied the fixed-parameter tractability of the problem of finding temporal paths between a source and destination that wait no longer than Δ consecutive timesteps at any intermediate vertex. Bumpus and Meeks [10] considered, again from a fixed-parameter perspective, a temporal graph exploration variant in which the goal is no longer to visit all of the input graph's vertices at least once, but to traverse all edges of its underlying graph exactly once (i.e., computing a temporal Eulerian circuit). The problem of Non-Strict Temporal Exploration was introduced and studied in [17]. Here, a computed walk may make an unlimited number of edge-traversals in each given timestep. Amongst other things, NP-completeness of the general problem is shown, as well as $O(n^{1/2-\varepsilon})$ and $O(n^{1-\varepsilon})$-inapproximability for the problem of minimizing the arrival time of a temporal exploration in the cases where the number of timesteps required to reach any vertex v from any vertex u is bounded by $c = 2$ and $c = 3$, respectively. Notions of strict/non-strict paths which respectively allow for a single edge/unlimited number of edge(s) to be crossed in any timestep have been considered before, notably by Kempe et al. [19] and Zschoche et al. [28].

Preliminaries. For a pair of integers x, y with $x \leq y$ we denote by $[x, y]$ the set $\{z : x \leq z \leq y\}$; if $x = 1$ we write $[y]$ instead. We use standard terminology from graph theory [14], and we assume any static graph $G = (V, E)$ to be simple and undirected. A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter. The problem is in FPT (fixed-parameter tractable) if there is an algorithm that solves every instance in time $f(k) \times |I|^{O(1)}$ for some computable function f. A proof that a problem is hard for complexity class W[r] for some integer $r \geq 1$ is seen as evidence that the problem is unlikely to be contained in FPT. For more on parameterized complexity, including definitions of the complexity classes W[r], we refer to [15, 13]. We defer formal definitions of both the strict and non-strict variants of TEXP, as well as their associated temporal graph models, to Sections 2 and 3 respectively.

2 Strict TEXP parameterizations

We begin with the definition of a temporal graph:

▶ **Definition 1** (Temporal graph)**.** *A temporal graph $\mathcal{G}$ with underlying graph $G = (V, E)$, lifetime L and order n is a sequence of simple undirected graphs $\mathcal{G} = \langle G_1, G_2, ..., G_L\rangle$ such that $|V| = n$ and $G_t = (V, E_t)$ (where $E_t \subseteq E$) for all $t \in [L]$.*

For a temporal graph $\mathcal{G} = \langle G_1, ..., G_L\rangle$, the subscripts $t \in [L]$ indexing the graphs in the sequence are referred to as *timesteps* (or *steps*) and we call G_t the t-th layer. A tuple (e, t) with $e \in E(G)$ is an *edge-time pair* (or *time edge*) of $\mathcal{G}$ if $e \in E_t$. Note that the size of any temporal graph (i.e., the maximum number of time edges) is bounded by $O(Ln^2)$.

▶ **Definition 2** (Strict temporal walk)**.** *A strict temporal walk W in $\mathcal{G}$ is a tuple $W = (t_0, S)$, consisting of a start time t_0 and an alternating sequence of vertices and edge-time pairs $S = \langle v_1, (e_1, t_1), v_2, (e_2, t_2), ..., v_{l-1}, (e_{l-1}, t_{l-1}), v_l\rangle$ such that $e_i = \{v_i, v_{i+1}\}$, $e_i \in G_{t_i}$ for $i \in [l-1]$ and $1 \leq t_0 \leq t_1 < t_2 < \cdots < t_{l-1} \leq L$.*

We say that a temporal walk $W = (t_0, S)$ *visits* any vertex that is included in S. Further, W *traverses* edge e_i at time t_i for all $i \in [l-1]$ and is said to *depart from* (or start at) $v_1 \in V(\mathcal{G})$ at timestep t_0 and *arrive at* (or finish at) $v_l \in V(\mathcal{G})$ at the end of timestep t_{l-1} (or, equivalently, at the beginning of timestep $t_{l-1}+1$). Its *arrival time* is defined to be $t_{l-1}+1$. It is assumed that W is positioned at v_1 at the start of timestep $t_0 \in [t_1]$ and waits at v_1 until edge e_1 is traversed during timestep t_1. The quantity $|W| = t_{l-1} - t_0 + 1$ is called the *duration* of W.

Throughout this section we denote by $sp(u, v, t)$ the duration of a shortest (i.e., having minimum arrival time) temporal walk in $\mathcal{G}$ that starts at $u \in V(\mathcal{G})$ in timestep t and ends at $v \in V(\mathcal{G})$. (If $u = v$, $sp(u, v, t) = 0$.) We note that there is no guarantee that a walk between a pair of vertices u, v exists; in such cases we let $sp(u, v, t) = \infty$. The algorithms that we present in Sections 2.1 and 2.2 will repeatedly require us to compute such shortest walks for specific pairs of vertices $u, v \in V(\mathcal{G})$ and a timestep $t \in [L]$ – the following theorem allows us to do this:

▶ **Theorem 3** (Wu et al. [27]). *Let $\mathcal{G} = \langle G_1, ..., G_L\rangle$ be an arbitrary temporal graph. Then, for any $u \in V(\mathcal{G})$ and $t \in [L]$, one can compute in $O(Ln^2)$ time for all $v \in V(\mathcal{G})$ a temporal walk that starts at u, ends at v and has duration $sp(u, v, t)$ (or determine that no such walk exists).*

The following two definitions will be used to describe the sets of candidate solutions for each of the problems that we consider in this section:

▶ **Definition 4** ((v, t, X)-tour). *A (v, t, X)-tour W in a given temporal graph $\mathcal{G}$ is a strict temporal walk that starts at some vertex $v \in V(\mathcal{G})$ in timestep t and visits all vertices in $X \subseteq V(\mathcal{G})$. The arrival time $\alpha(W)$ of a (v, t, X)-tour W is the timestep after the timestep at the end of which W has for the first time visited all vertices in X.*

▶ **Definition 5** ((v, t, k)-tour). *A (v, t, k)-tour W in a given temporal graph $\mathcal{G}$ is a (v, t, X)-tour for some subset $X \subseteq V(\mathcal{G})$ that satisfies $|X| = k$. The arrival time $\alpha(W)$ of a (v, t, k)-tour W is the timestep after the timestep at the end of which W has for the first time visited all vertices in X.*

A (v, t, X)-tour W ((v, t, k)-tour W^*) in a temporal graph $\mathcal{G}$ is said to be *foremost* if $\alpha(W) \leq \alpha(W')$ ($\alpha(W^*) \leq \alpha(W^{*\prime})$) for any other (v, t, X)-tour W' (any other (v, t, k)-tour $W^{*\prime}$). We now formally define this section's two main problems of interest:

▶ **Definition 6** (k-FIXED TEXP). *An instance of the k-FIXED TEXP problem is given as a tuple $(\mathcal{G}, s, X, k)$ where $\mathcal{G} = \langle G_1, ..., G_L\rangle$ is an arbitrary temporal graph with underlying graph G and lifetime L; s is a start vertex in $V(\mathcal{G})$; and $X \subseteq V(\mathcal{G})$ is a set of target vertices such that $|X| = k$. The problem then asks that we decide if there exists an $(s, 1, X)$-tour W in $\mathcal{G}$.*

▶ **Definition 7** (k-ARBITRARY TEXP). *An instance of the k-ARBITRARY TEXP problem is given as a tuple $(\mathcal{G}, s, k)$ where $\mathcal{G} = \langle G_1, ..., G_L\rangle$ is an arbitrary temporal graph with underlying graph G and lifetime L; s is a start vertex in $V(\mathcal{G})$; and $k \in \mathbb{N}$. The problem then asks that we decide whether there exists an $(s, 1, k)$-tour W in $\mathcal{G}$.*

For yes-instances of k-FIXED TEXP or k-ARBITRARY TEXP, a tour with minimum arrival time (among all tours of the type sought) is called an *optimal solution.*

2.1 An FPT algorithm for k-fixed TEXP

In this section we provide a deterministic FPT-time algorithm for K-FIXED TEXP. Let $(\mathcal{G}, s, X, k)$ be an instance of K-FIXED TEXP. Our algorithm looks for an earliest arrival time $(s, 1, X)$-tour of $\mathcal{G}$ via a dynamic programming (DP) approach. We note that the approach is essentially an adaptation of an algorithm proposed (independently by Bellman [5] and Held & Karp [18]) for the classic Travelling Salesperson Problem to the parameterized problem for temporal graphs.

▶ **Theorem 8.** *It is possible to decide any instance $I = (\mathcal{G}, s, X, k)$ of K-FIXED TEXP, and return an optimal solution if I is a yes-instance, in time $O(2^k kLn^2)$, where $n = |V(\mathcal{G})|$ and L is $\mathcal{G}$'s lifetime.*

Proof. First we describe our algorithm before proving its correctness and analysing its running time. We begin by specifying a dynamic programming formula for $F(S, v)$, by which we denote the minimum arrival time of any temporal walk in $\mathcal{G}$ that starts at vertex $s \in V(\mathcal{G})$ in timestep 1, visits all vertices in $S \subseteq X$, and finishes at vertex $v \in S$. One can compute $F(S, v)$ via the following formula:

$$F(S,v) = \begin{cases} 1 + sp(s, v, 1) & (|S| = 1) \\ \min\limits_{u \in S - \{v\}} [F(S - \{v\}, u) + sp(u, v, F(S - \{v\}, u))] & (|S| > 1) \end{cases} \tag{1}$$

Note that to compute $F(S, v)$ when $|S| > 1$, Equation (1) states that we need only consider values $F(S', u)$ with $u \in S'$ and $|S'| = |S| - 1$, and so we begin by computing all values $F(S', u)$ such that $S' \subseteq X$ satisfies $|S'| = 1$ and $u \in S'$, before computing all values such that $|S'| = 2$ and $u \in S'$ and so on, until we have computed all values $F(X, u)$ where $u \in X$ (i.e., values $F(S', u)$ with $|S'| = k = |X|$). Once all necessary values have been obtained, computing the following value gives the arrival time of an optimal $(s, 1, X)$-tour:

$$F^* = \min_{v \in X} F(X, v). \tag{2}$$

If, whenever we compute a value $F(S, v)$ with $|S| > 1$, we also store alongside $F(S, v)$ a single pointer

$$p(S, v) = \arg \min_{u \in S - \{v\}} [F(S - \{v\}, u) + sp(u, v, F(S - \{v\}, u))],$$

then once we have computed F^* we can use a traceback procedure to reconstruct the walk with arrival time F^*. More specifically, let $u_1 = \arg\min_{u \in X} F(X, u)$ and $u_i = p(X - \{u_1, ..., u_{i-2}\}, u_{i-1})$ for all $i \in [2, k]$. To complete the algorithm, we then check if F^* is finite: If so, then there must be a $(s, 1, X)$-tour W in $\mathcal{G}$ with $\alpha(W) = F^*$ that visits the vertices $u_k, ..., u_1$ in that order. We can reconstruct W by concatenating the k shortest walks obtained by starting at s in timestep 1 and computing a shortest walk from s to u_k, then computing a shortest walk from u_k to u_{k-1} starting at the timestep at which u_k was reached, and so on, until u_1 is reached; once constructed, return W. If, on the other hand, $F^* = \infty$ (which is possible by the definition of $sp(u, v, t)$) then return no.

Correctness. The correctness of Equation (1) can be shown via induction on $|S|$: The base case (i.e., when $|S| = 1$) is correct since the arrival time of the foremost temporal walk that starts at s in timestep 1 and ends at a specific vertex $v \in X$ is clearly equal to one plus the duration of the foremost temporal walk between s and v starting at timestep 1.

For the general case (when $|S| > 1$), assume first that the formula holds for any set S' such that $|S'| = l$ and any vertex $u \in S'$. To see that the formula holds for all sets S with $|S| = l + 1$ and vertices $v \in S$, consider any walk W that starts in timestep 1, visits all vertices in some set S with $|S| = l + 1$ and ends at v. Let $x_1, ..., x_{l+1}$ be the order in which the vertices $x_i \in S$ are reached by W for the first time; let $x = x_{l+1} = v$ and $x' = x_l$. Note that the subwalk W' of W that begins in timestep 1 and finishes at the end of the timestep in which W arrives at x' for the first time is surely an $(s, 1, S - \{v\})$-tour, since W' visits every vertex in $S - \{x\} = S - \{v\}$. Then, by the induction hypothesis we have $\alpha(W') \geq F(S - \{v\}, x')$ because $|S - \{v\}| = l$, and since W ends at v we have

$$\begin{aligned} \alpha(W) &\geq \alpha(W') + sp(x', v, \alpha(W')) \\ &\geq F(S - \{v\}, x') + sp(x', v, F(S - \{v\}, x')). \end{aligned}$$

More generally, we can say that any $(s, 1, S)$-tour W that starts at s in timestep 1, visits all vertices in S (where $|S| = l + 1$), and finishes at $v \in S$ satisfies the above inequality for some $x' \in S - \{v\}$. Note that for any $u \in S - \{v\}$, $F(S - \{v\}, u) + sp(u, v, F(S - \{v\}, u))$ corresponds to the arrival time of a valid $(s, 1, S)$-tour, obtained by concatenating an earliest arrival time $(s, 1, S - \{v\})$-tour that ends at u and a shortest walk between u and v starting at time $F(S - \{v\}, u)$. Therefore, to compute $F(S, v)$ it suffices to compute the minimum value of $F(S - \{v\}, u) + sp(u, v, F(S - \{v\}, u)$ over all $u \in S - \{v\}$; note that this is exactly Equation (1) in the case that $|S| > 1$.

To establish the correctness of Equation (2) recall that, by Definition 4, the arrival time of any $(s, 1, X)$-tour in $\mathcal{G}$ is equal to the timestep after the timestep in which it traverses a time edge to reach the final unvisited vertex of X for the first time. Assume that I is a yes-instance and let $x^* \in X$ be the k-th unique vertex in X that is visited by some foremost $(s, 1, X)$-tour W; then, by the analysis in the previous paragraph, we must have $\alpha(W) = F(X, x^*)$ since W is foremost, so $x^* = \arg\min_{v \in X} F(X, v)$ and thus $\alpha(W) = F(X, x^*) = \min_{v \in X} F(X, v) = F^*$, as required.

The fact that the answer returned by the algorithm is correct follows from the correctness of Equations (1) and (2) and the traceback procedure, together with the fact that I is a no-instance if and only if $F^* = \infty$. The details of this second claim are not difficult to see and are omitted, but we note that it is indeed possible that $F^* = \infty$ since F^* is the summation of a number of values $sp(u, v, t)$, some of which may satisfy $sp(u, v, t) = \infty$ by definition.

Runtime analysis. Since we only compute values of $F(S, v)$ such that $v \in S$ and $1 \leq |S| \leq k$, in total we compute $O(\sum_{i=1}^{k} \binom{k}{i} i) = O(2^k k)$ values. Note that, to compute any value $F(S, v)$ with $|S| = i > 1$, Equation (1) requires that we consider the values $F(S - \{v\}, u) + sp(u, v, F(S - \{v\}, u))$ with $u \in S - \{v\}$, of which there are exactly $i - 1$. We therefore use Theorem 3 to compute (and store temporarily), for each S' with $|S'| = i - 1$ and $x \in S'$, in $O(Ln^2)$ time the value of $sp(x, y, F(S', x))$ for all $y \in V(\mathcal{G})$ immediately after computing all $F(S', x)$, and use these precomputed shortest walk durations to compute $F(S, v)$ for any S with $|S| = i$ and $v \in S$ in time $O(i) = O(k)$. Thus, we spend $O(k) + O(Ln^2) = O(Ln^2)$ (since $k \leq n$) time for each of $O(2^k k)$ values $F(S, v)$. This yields an overall time of $O(2^k kLn^2)$. Note that F^* can be computed using Equation (2) in $O(k)$ time since we take the minimum of $O(k)$ values; also note that a $(v, 1, X)$-tour with arrival time F^* can be reconstructed in time $O(kLn^2)$ using the aforedescribed traceback procedure, since we need to recompute $O(k)$ shortest walks, spending $O(Ln^2)$ time on each walk. Hence the overall running time of the algorithm is bounded by $O(2^k kLn^2)$, as claimed. ◀

We remark that k-fixed TEXP becomes TEXP if $X = V$, hence Theorem 8 also implies an FPT algorithm for TEXP parameterized by the number of vertices. Furthermore, we observe that TEXP is also FPT when parameterized by the lifetime L of the given temporal graph: If $L < n-1$, the instance is clearly a no-instance, and if $L \geq n-1$, the FPT algorithm for TEXP with parameter n is also FPT for parameter L.

2.2 FPT algorithms for k-arbitrary TEXP

The main result of this section is a randomized FPT-time algorithm for k-arbitrary TEXP that utilises the *colour-coding* technique originally presented by Alon, Yuster and Zwick [4]. There, they employed the technique primarily to detect the existence of a k-vertex simple path in a given undirected graph G. More generally, it has proven useful as a technique for finding fixed motifs (i.e., prespecified subgraphs) in static graphs/networks. We provide a high-level description of the technique and the way that we apply it at the beginning of Section 2.2.1. A standard derandomization technique (also originating from [4]) is then utilised within Section 2.2.2 to obtain a deterministic algorithm for k-arbitrary TEXP with a worse, but still FPT, running time.

2.2.1 A randomized algorithm

The algorithm of this section employs the colour-coding technique of Alon, Yuster and Zwick [4]. First, we informally sketch the structure of the algorithm behind Theorem 9: We colour the vertices of an input temporal graph uniformly at random, then by means of a DP subroutine we look for a temporal walk that begins at some start vertex s in timestep 1 and visits k vertices with distinct colours by the earliest time possible. Notice that if such a walk is found then it must be a (v, t, k)-tour, since the k vertices are distinctly coloured and therefore must be distinct. Then, the idea is to repeatedly: (1) randomly colour the input graph $\mathcal{G}$'s vertices; then (2) run the DP subroutine on each coloured version of $\mathcal{G}$. We repeat these steps enough times to ensure that, with high probability, the vertices of an optimal $(s, 1, k)$-tour are coloured with distinct colours at least once over all colourings – if this happens then the DP subroutine will surely return an optimal $(s, 1, k)$-tour or one with equal arrival time. With this high-level description in mind, we now present/analyse the algorithm:

▶ **Theorem 9.** *For every $\varepsilon > 0$, there exists a Monte Carlo algorithm that, with probability $1-\varepsilon$, decides a given instance $I = (\mathcal{G}, s, k)$ of k-arbitrary TEXP, and returns an optimal solution if I is a yes-instance, in time $O((2e)^k L n^3 \log \frac{1}{\varepsilon})$, where $n = |V(\mathcal{G})|$ and L is $\mathcal{G}'s$ lifetime.*

Proof. Let $V := V(\mathcal{G})$. We now describe our algorithm before proving it correct and analysing its running time. Let $c : V \to [k]$ be a colouring of the vertices $v \in V$. Let a walk W in $\mathcal{G}$ that starts at s and visits a vertex coloured with each colour in $D \subseteq [k]$ be known as a *D-colourful walk*; let the timestep after the timestep at the end of which W has for the first time visited vertices with k distinct colours be known as the *arrival time* of W, denoted by $\alpha(W)$. The algorithm employs a subroutine that computes, should one exist, a $[k]$-colourful walk W in $\mathcal{G}$ with earliest arrival time. Note that a D-colourful walk ($D \subseteq [k]$) in $\mathcal{G}$ is by definition an $(s, 1, |D|)$-tour in $\mathcal{G}$.

Define $H(D, v)$ to be the earliest arrival time of any D-colourful walk (where $D \subseteq [k]$) in $\mathcal{G}$ that ends at a vertex v with $c(v) \in D$. The value of $H(D, v)$ for any $D \subseteq [k]$ and v with $c(v) \in D$ can be computed via the following dynamic programming formula (within the formula we denote by $D^-_{c(v)}$ the set $D - \{c(v)\}$):

$$H(D,v) = \begin{cases} 1 + sp(s,v,1) & (|D| = 1) \\ \min\limits_{u \in V : c(u) \in D^-_{c(v)}} [H(D^-_{c(v)}, u) + sp(u, v, H(D^-_{c(v)}, u))] & (|D| > 1) \end{cases} \tag{3}$$

In order to compute $H(D, v)$ for any $D \subseteq [k]$ and vertex v with $c(v) \in D$, Equation (3) requires that we consider values $H(D - \{c(v)\}, u)$ such that $c(u) \in D - \{c(v)\}$, and so we begin by computing $H(D', v)$ for all D' with $|D'| = 1$ and v with $c(v) \in D'$, then for all D' with $|D'| = 2$ and v with $c(v) \in D'$, and so on, until all values $H([k], v)$ have been obtained. The earliest arrival time of any $[k]$-colourful walk in $\mathcal{G}$ is then given by

$$H^* = \min_{u \in V(\mathcal{G})} H([k], u). \tag{4}$$

Once H^* has been computed, we check whether its value is finite or equal to ∞. If H^* is finite then we can use a pointer system and traceback procedure (almost identical to those used in the proof of Theorem 8) to reconstruct an $(s, 1, k)$-tour with arrival time H^* if one exists; otherwise we return no. This concludes the description of the dynamic programming subroutine.

Let $r = \lceil \frac{1}{\varepsilon} \rceil$ and let W^* initially be the trivial walk that starts and finishes at vertex s in timestep 1. Perform the following two steps for $e^k \ln r$ iterations:

1. Assign colours in $[k]$ to the vertices of V uniformly at random and check if all k colours colour at least one vertex of G; if not, start next iteration. If yes, proceed to step 2.
2. Run the DP subroutine in order to find an optimal $[k]$-colourful walk W in $\mathcal{G}$ if one exists. If such a W is found then check if $\alpha(W) < \alpha(W^*)$ or W^* starts and ends at s in timestep 1 (i.e., still has its initial value), and in either case set $W^* = W$; otherwise the DP subroutine returned no and we make no change to W^*.

Once all iterations of the above steps are over, check if W^* is still equal to the walk that starts and finishes at s in timestep 1; if not then return W^*, otherwise return no. This concludes the algorithm's description.

Correctness. We focus on proving the randomized aspect of the algorithm correct and omit correctness proofs for Equations (3) and (4) since the arguments are similar to those provided in Theorem 8's proof.

If I is a no-instance then in no iteration will the DP subroutine find an $(s, 1, k)$-tour in $\mathcal{G}$. Hence in the final step the algorithm will find that W^* is equal to the walk that starts and ends at s in timestep 1 (by the correctness of Equations (3) and (4)) and return no, which is clearly correct. Assume then that I is yes-instance. Let W be an $(s, 1, k)$-tour in $\mathcal{G}$ with earliest arrival time, and let $X \subseteq V$ be the set of k vertices visited by W. Then, if during one of the $e^k \ln r$ iterations of steps 1 and 2 we colour the vertices of V in such a way that X is well-coloured (we say that a set of vertices $U \subseteq V$ is *well-coloured* by colouring c if $c(u) \neq c(v)$ for every pair of vertices $u, v \in U$), W will induce an optimal $[k]$-colourful walk in $\mathcal{G}$. The DP subroutine will then return W or some other optimal $[k]$-colourful walk W' with $\alpha(W) = \alpha(W')$ that visits a well-coloured subset of vertices X'; note that the arrival time of the best tour found in any iteration so far will then surely be $\alpha(W)$, since W has earliest arrival time.

Observe that if we colour the vertices of V with k colours uniformly at random, then, since $|X| = k$, there are k^k ways to colour the vertices in $X \subseteq V$, of which $k!$ constitute well-colourings of X. Hence after a single colouring of V we have

$$\Pr[X \text{ is well-coloured}] = \frac{k!}{k^k} > \frac{1}{e^k},$$

where the inequality follows from the fact that $k!/k^k > \sqrt{2\pi}k^{\frac{1}{2}}e^{\frac{1}{12k+1}}/e^k$ (this inequality is due to Robbins [25] and is related to Stirling's formula). Hence, after $e^k \ln r$ colourings, we have (using the standard inequality $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ for all $x \geq 1$):

$$\Pr[X \text{ is not well-coloured in any colouring}] \leq \left(1 - \frac{1}{e^k}\right)^{e^k \ln r} \leq 1/r \leq \varepsilon.$$

Thus, the probability that X is well-coloured at least once after $e^k \ln r$ colourings is at least $1 - \varepsilon$. It follows that, with probability $\geq 1 - \varepsilon$, the earliest arrival $[k]$-colourful walk returned by the algorithm after all iterations is in fact an optimal $(s, 1, k)$-tour in $\mathcal{G}$, since either W or some other $(s, 1, k)$-tour with equal arrival time will eventually be returned.

Runtime analysis. Note that the DP subroutine computes exactly the values $H(D, v)$ such that $D \subseteq [k]$ and v satisfies $c(v) \in D$. Hence there are at most $\binom{k}{i}n$ values $H(D, v)$ such that $|D| = i$, for all $i \in [k]$; this gives a total of $\sum_{i\in[k]} \binom{k}{i}n = O(2^k n)$ values. In order to compute $H(D, v)$ for any D with $|D| = i > 1$, Equation (3) requires us to consider the value of $H(D - \{c(v)\}, u) + sp(u, v, H(D - \{c(v)\}, u))$ for all u such that $c(u) \in D - \{c(v)\}$. Therefore, similar to the algorithm in the proof of Theorem 8, we compute and store, immediately after computing each value $H(D', x)$ with $|D'| = i - 1$ and $c(x) \in D'$, the value of $sp(x, y, H(D', x))$ for all $y \in V(\mathcal{G})$ in $O(Ln^2)$ time (Theorem 3). Note that there can be at most n vertices u such that $c(u) \in D - \{c(v)\}$, and so in total we spend $O(n) + O(Ln^2) = O(Ln^2)$ time on each of $O(2^k n)$ values of $H(D, v)$, giving an overall time of $O(2^k Ln^3)$. We can compute H^* in $O(n)$ time since we take the minimum of $O(n)$ values, and the traceback procedure can be performed in $O(kLn^2) = O(Ln^3)$ time since we concatenate k walks obtained using Theorem 3. Thus the overall time spent carrying out one execution of the DP subroutine is $O(2^k Ln^3)$.

Since the running time of each iteration of the main algorithm is dominated by the running time of the DP subroutine and there are $e^k \ln r = O(e^k \log \frac{1}{\varepsilon})$ iterations in total, we conclude that the overall running time of the algorithm is $O((2e)^k Ln^3 \log \frac{1}{\varepsilon})$, as claimed. This completes the proof. ◀

2.2.2 Derandomizing the algorithm of Theorem 9

The randomized colour-coding algorithm of Theorem 9 can be derandomized at the expense of incurring a $k^{O(\log k)} \log n$ factor in the running time. We employ a standard derandomization technique, presented initially in [4], which involves the enumeration of a *k-perfect family of hash functions* from $[n]$ to $[k]$. The functions in such a family will be viewed as colourings of the vertex set of the temporal graph given as input to the k-ARBITRARY TEXP problem.

Formally, a family $\mathcal{H}$ of hash functions from $[n]$ to $[k]$ is *k-perfect* if, for every subset $S \subseteq [n]$ with $|S| = k$, there exists a function $f \in \mathcal{H}$ such that f restricted to S is bijective (i.e., one-to-one). The following theorem of Naor et al. enables one to construct such a family $\mathcal{H}$ in time linear in the size of $\mathcal{H}$:

▶ **Theorem 10** (Naor, Schulman and Srinivasan [24]). *A k-perfect family $\mathcal{H}$ of hash functions f_i from $[n]$ to $[k]$, with size $e^k k^{O(\log k)} \log n$, can be computed in $e^k k^{O(\log k)} \log n$-time.*

We note that the value of $f_i(x)$ for any $f_i \in \mathcal{H}$ and $x \in [n]$ can be evaluated in $O(1)$ time.

To solve an instance of k-Arbitrary TEXP, we can now use the algorithm from the proof of Theorem 9, but instead of iterating over $e^k \ln r$ random colourings, we iterate over the $e^k k^{O(\log k)} \log n$ hash functions in the k-perfect family of hash functions constructed using Theorem 10. This ensures that the set X of k vertices visited by an optimal $(s, 1, k)$-tour is well-coloured in at least one iteration, and we obtain the following theorem.

▶ **Theorem 11.** *There is a deterministic algorithm that can solve a given instance $(\mathcal{G}, s, k)$ of k-Arbitrary TEXP in $(2e)^k k^{O(\log k)} L n^3 \log n$ time, where $n = |V(\mathcal{G})|$. If the instance is a yes-instance, the algorithm also returns an optimal solution.*

We remark that, since a temporal walk can visit at most $L + 1$ vertices in a temporal graph with lifetime L, Theorem 11 also implies an FPT algorithm for the following problem, parameterized by the lifetime L of the given temporal graph: Find a temporal walk that visits as many distinct vertices as possible.

3 Non-Strict TEXP parameterizations

In this section we consider the non-strict version of TEXP, in which a walk is allowed to traverse an unlimited number of edges in every timestep. As mentioned in the introduction, this changes the nature of the problem significantly. In particular, it means that a temporal walk positioned at a vertex v in timestep t is able to visit, during timestep t, any other vertex contained in the same connected component C as v and move to an arbitrary vertex $u \in C$, beginning timestep $t + 1$ positioned at vertex u. As such, it is no longer necessary to know the edge structure of the input temporal graph during each timestep, and we can focus only on the connected components of each layer. This leads to the following definition:

▶ **Definition 12** (Non-strict temporal graph, $\mathcal{G}$). *A non-strict temporal graph $\mathcal{G} = \langle G_1, ..., G_L \rangle$ with vertex set $V := V(\mathcal{G})$ and lifetime L is an indexed sequence of partitions (layers) $G_t = \{C_{t,1}, ..., C_{t,s_t}\}$ of V for $t \in [L]$. For all $t \in [L]$, each $v \in V$ satisfies $v \in C_{t,j}$ for a unique $j \in [s_t]$. The integer s_t denotes the number of components in layer G_t; clearly we have $s_t \in [n]$.*

A non-strict temporal walk is then defined as follows:

▶ **Definition 13** (Non-strict temporal walk, W). *A non-strict temporal walk W starting at vertex v at time t_1 in a non-strict temporal graph $\mathcal{G} = \langle G_1, ..., G_L \rangle$ is a sequence $W = C_{t_1,j_1}, C_{t_2,j_2}, ..., C_{t_l,j_l}$ of components C_{t_i,j_i} ($i \in [l]$) with $1 \le t_1 \le t_l \le L$ such that: $t_i + 1 = t_{i+1}$ for all $i \in [1, l-1]$; $C_{t_i,j_i} \in G_{t_i}$ and $j_i \in [s_{t_i}]$ for all $i \in [l]$; $C_{t_i,j_i} \cap C_{t_{i+1},j_{i+1}} \neq \emptyset$ for all $i \in [l-1]$; and $v \in C_{t_1,j_1}$.*

Let $W = C_{t_1,j_1}, C_{t_2,j_2}, ..., C_{t_l,j_l}$ be a non-strict temporal walk in some non-strict temporal graph $\mathcal{G}$ starting at some vertex $s \in C_{t_1,j_1}$. We call $l \in [L]$ the *duration* of W. The walk W is said to start at vertex $s \in C_{t_1,j_1}$ in timestep t_1 and finish at component C_{t_l,j_l} (or sometimes at some $v \in C_{t_l,j_l}$) in timestep t_l. Furthermore, W *visits* the set of vertices $\bigcup_{i \in [l]} C_{t_i,j_i}$. Note that W visits exactly one component in each of the l timesteps that make up its duration. We call W *non-strict exploration schedule starting at s* with *arrival* time l if $t_1 = 1$ and $\bigcup_{i \in [l]} C_{t_i,j_i} = V(\mathcal{G})$. As FPT algorithms for k-fixed TEXP and k-arbitrary TEXP for non-strict temporal graphs can be derived using similar techniques as in Section 2, we instead consider the following two non-strict exploration problems in this section:

▶ **Definition 14** (Non-Strict Temporal Exploration (NS-TEXP)). *An instance of NS-TEXP is given as a tuple $(\mathcal{G}, s)$, where $\mathcal{G}$ is a non-strict temporal graph with lifetime L and $s \in V(\mathcal{G})$ is a start vertex. The problem then asks whether or not $\mathcal{G}$ admits an exploration schedule that starts at s.*

▶ **Definition 15** (Set NS-TEXP). *An instance of Set NS-TEXP is given as a tuple $(\mathcal{G}, s, \mathcal{X})$, where $\mathcal{G}$ is a non-strict temporal graph with lifetime L, $s \in V(\mathcal{G})$ is a start vertex, and $\mathcal{X} = \{X_1, \dots, X_m\}$ is a set of subsets $X_i \subseteq V(\mathcal{G})$. The problem then asks whether or not there exists a non-strict temporal walk in $\mathcal{G}$ that starts at s in timestep 1 and visits at least one vertex contained in X_i for all $i \in [m]$.*

In the following two subsections we establish FPT-membership for NS-TEXP when parameterized by the lifetime L, then prove W[2]-hardness for the Set NS-TEXP problem when the same parameter is considered.

3.1 An FPT algorithm for NS-TEXP with parameter L

Let NS-TEXP-L be the variant of NS-TEXP parameterized by the lifetime L of the input temporal graph $\mathcal{G}$; let an instance of NS-TEXP-L be given as a tuple $(\mathcal{G}, s, L)$. We prove that NS-TEXP-L $\in$ FPT by specifying a bounded search tree-based algorithm.

Let $\mathcal{G} = \langle G_1, \dots, G_L \rangle$ be some non-strict temporal graph. Throughout this section we let $\mathcal{C}(\mathcal{G}) := \bigcup_{t \in [L]} G_t$, i.e., $\mathcal{C}(\mathcal{G})$ is the set of all components belonging to some layer of $\mathcal{G}$. We implicitly assume that each component $C \in \mathcal{C}(\mathcal{G})$ is *associated* with a unique layer G_t of $\mathcal{G}$ in which it is contained. If a component (seen as just a set of vertices) occurs in several layers, we thus treat these occurrences as different elements of $\mathcal{C}(\mathcal{G})$ (or of any subset thereof) because they are associated with different layers. If X is a set of components in $\mathcal{C}(\mathcal{G})$ that are associated with distinct layers (i.e., no two components in X are associated with the same layer G_t of $\mathcal{G}$), then we say that the components in X *originate from unique layers of* $\mathcal{G}$. For a set X of components that originate from unique layers of $\mathcal{G}$, we let $D(X) := \bigcup_{C \in X} C$ be the union of the vertex sets of the components in X. For any such set X, we also let $T(X) = \{t \in [L] : \text{there is a } C \in X \text{ associated with layer } G_t\}$.

Within the following, we assume that $\mathcal{G}$ admits a non-strict exploration schedule W:

▶ **Observation 16.** *Let X ($|X| \in [0, L-1]$) be a subset of the components visited by the exploration schedule W. Then there exists $C \in \mathcal{C}(\mathcal{G}) - X$ with $C \in G_t$ ($t \in [L] - T(X)$) such that $|C - D(X)| \geq (n - |D(X)|)/(L - |T(X)|)$.*

Observation 16 follows since, otherwise, W visits at most $L - |T(X)|$ components $C \in \mathcal{C}(\mathcal{G}) - X$ that each contain $|C - D(X)| < (n - |D(X)|)/(L - |T(X)|)$ of the vertices $v \notin D(X)$, and so the total number of vertices visited by W is strictly less than $|D(X)| + (L - |T(X)|) \cdot (n - |D(X)|)/(L - |T(X)|) = n$, a contradiction.

We briefly outline the main idea of our FPT result: We use a search tree algorithm that maintains a set X of components that a potential exploration schedule could visit, starting with the empty set. Then the algorithm repeatedly tries all possibilities for adding a component (from some so far untouched layer) that contains at least $(n - |D(X)|)/(L - |T(X)|)$ unvisited vertices (whose existence is guaranteed by Observation 16 if there exists an exploration schedule). It is clear that the search tree has depth L, and the main further ingredient is an argument showing that the number of candidates for the component to be added is bounded by a function of L, namely, by $(L - |T(X)|)^2$: This is because each of the $L - |T(X)|$ untouched layers can contain at most $L - |T(X)|$ components that each contain at least $(n - |D(X)|)/(L - |T(X)|)$ unvisited vertices. We now proceed to describe the details of the algorithm and its analysis.

▶ **Lemma 17.** *Let $\mathcal{G} = \langle G_1, ..., G_L \rangle$ be an arbitrary order-n non-strict temporal graph. Then, for components $C_{t_1,j_1} \in G_{t_1}$ and $C_{t_2,j_2} \in G_{t_2}$ (with $1 \le t_1 \le t_2 \le L$) one can decide, in $O((t_2 - t_1 + 1)n)$ time, whether there exists a non-strict temporal walk beginning at any vertex contained in C_{t_1,j_1} in timestep t_1 and finishing at C_{t_2,j_2} in timestep t_2.*

Proof. For any $v \in V(\mathcal{G})$ and $t \in [t_1, t_2]$, let $c(v,t)$ denote the component $C_{t,j}$ such that $v \in C_{t,j}$ during timestep t. First, precompute the values $c(v,t)$ by, for every $t \in [t_1, t_2]$, scanning each component $C \in G_t$ and setting $c(v,t) = C$ if and only if $v \in C$. Next, let $X_{t_1} = \{C_{t_1,j_1}\}$ and then consider the timesteps $t \in [t_1+1, t_2]$ in increasing order, constructing at each timestep t the set $X_t = X_{t-1} \cup \{c(v,t) : v \in \bigcup_{C \in X_{t-1}} C\}$. Finally, check whether $C_{t_2,j_2} \in X_{t_2}$, returning yes if so and no otherwise.

The correctness of the algorithm is not hard to see. To see that the claimed running time of $O((t_2 - t_1 + 1)n)$ holds, note first that precomputing the values $c(v,t)$ for any $v \in V(\mathcal{G})$ and any $t \in [t_1, t_2]$ requires $O((t_2 - t_1 + 1)n)$ time since, in each timestep $t \in [t_1, t_2]$, we simply iterate over the vertices (of which there are always n in total) contained in each component $C \in G_t$. Then, to compute X_t for each $t \in [t_1+1, t_2]$, we add $c(v,t)$ (which can be evaluated in $O(1)$ time due to our preprocessing step) to X_t for each vertex $v \in \bigcup_{C \in X_{t-1}} C$, of which there can be at most n. This second step of the algorithm also clearly requires $O((t_2 - t_1 + 1)n)$ time, and the lemma follows. ◀

Let X be a set of components originating from unique layers of $\mathcal{G}$, and let $W^?_{\mathcal{G}}(s, X) =$ yes if and only if there exists a non-strict temporal walk in $\mathcal{G}$ that starts at $s \in V(\mathcal{G})$ in timestep 1 and visits at least the components contained in X, and no otherwise.

▶ **Lemma 18.** *For any order-n non-strict temporal graph $\mathcal{G} = \langle G_1, ..., G_L \rangle$, any $s \in V(\mathcal{G})$, and any set X of components originating from unique layers of $\mathcal{G}$, $W^?_{\mathcal{G}}(s, X)$ can be computed in $O(Ln)$ time.*

Proof. Let $C_{s_1}, C_{s_2}, ..., C_{s_{|X|}}$ be an an index-ordered sequence of the components in X, with the indices $s_i \in [L]$ satisfying $C_{s_i} \in G_{s_i}$ (for all $i \in [|X|]$) and $s_i < s_{i+1}$ (for all $i \in [|X| - 1]$). Let $C_s \in G_1$ be the unique component in layer 1 such that $s \in C_s$ (note that we may have $C_{s_1} = C_s$). Now, apply the algorithm of Lemma 17 with $C_{t_1,j_1} = C_s$ and $C_{t_2,j_2} = C_{s_1}$, and then with $C_{t_1,j_1} = C_{s_i}$ and $C_{t_2,j_2} = C_{s_{i+1}}$ for all $i \in [|X| - 1]$. If the return value of any application of the algorithm of Lemma 17 is no, then we return $W^?_{\mathcal{G}}(s, X) =$ no; otherwise we return $W^?_{\mathcal{G}}(s, X) =$ yes. This concludes the algorithm's description.

Since each component C_{s_i} can only be visited in timestep s_i it is clear that any walk that visits all components of X must visit them in the specified order. The algorithm sets $W^?_{\mathcal{G}}(s, X) =$ yes if the components of X can be visited in the specified order. On the other hand, if Lemma 17 returns no for at least one pair of input components $C_{s_i}, C_{s_{i+1}}$ (or C_s, C_{s_1}), then it must be that the components cannot be visited in this order, and thus the algorithm sets $W^?_{\mathcal{G}}(s, X) =$ no. Thus, the algorithm's correctness follows from the correctness of Lemma 17's algorithm. To see that the runtime of the algorithm is bounded by $O(Ln)$, recall that each application of Lemma 17's algorithm to start/finish components C_{s_i} and $C_{s_{i+1}}$ takes $c(s_{i+1} - s_i + 1)n$ time (for a constant c hidden in the bound of Lemma 17). Thus the total amount of time spent over all applications is $c(s_1 - 1 + 1)n + \sum_{i \in [|X|-1]} c(s_{i+1} - s_i + 1)n = cn(s_{|X|} + |X| - 1) \le cn(2L - 1) = O(Ln)$, where the last inequality holds since $|X|, s_{|X|} \le L$. ◀

Now, let $\mathcal{G}$ be some input graph, and let X be some set of components originating from unique layers of $\mathcal{G}$. For any $s \in V(\mathcal{G})$, the recursive function $g(\mathcal{G}, s, X)$ (Algorithm 1) returns yes if and only if there exists a non-strict exploration schedule of $\mathcal{G}$ that starts at s and visits (at least) the components contained in X, and returns no otherwise. We prove the correctness of Algorithm 1 in Lemma 19.

Algorithm 1 Recursive function $g(\mathcal{G}, s, X)$.

```
1  if $|X| = L$ or $|D(X)| = n$ then
2      if $|D(X)| = n$ then return $W^?_{\mathcal{G}}(s, X)$
3      else return no
4  else
5      $C' \leftarrow \{C \in \mathcal{C}(\mathcal{G}) - X : |C - D(X)| \geq (n - |D(X)|)/(L - |T(X)|)\}$
6      $C^* \leftarrow C' - \{C \in C' : C \in G_t, t \in T(X)\}$.
7      if $|C^*| = 0$ then return no
8      for $C \in C^*$ do
9          if $g(\mathcal{G}, s, X \cup \{C\}) =$ yes then return yes
10     end
11     return no
12 end
```

▶ **Lemma 19.** *For any non-strict temporal graph $\mathcal{G}$, any $s \in V(\mathcal{G})$, and any set X (with $|X| \in [0, L]$) containing components originating from unique layers of $\mathcal{G}$, Algorithm 1 correctly computes $g(\mathcal{G}, s, X)$.*

Proof. We first show that $g(\mathcal{G}, s, X)$ is correct in the base case, i.e., when $|X| = L$ or $|D(X)| = n$. If we have $|D(X)| = n$, then any non-strict temporal walk that starts at s in timestep 1 and visits all components in X is an exploration schedule. Thus, the correctness of line 2 follows from the definition of the return value $W^?_{\mathcal{G}}(s, X)$ (which can be computed using Lemma 18). If $|X| = L$ and $|D(X)| < n$, i.e., we have reached line 3, then there must exist no exploration schedule that visits each of the components in X, since any non-strict temporal walk of duration at most L can visit at most L components, but at least one additional component $C \notin X$ needs to be visited to cover at least one vertex $v \notin D(X)$ – thus it is correct to return no in this case.

Otherwise, we have $|X| < L$ and $|D(X)| < n$, and are in the recursive case. Then, by Observation 16, any non-strict exploration schedule that visits all components in X must visit at least one other component $C \in \mathcal{C}(\mathcal{G}) - X$ such that $|C - D(X)| \geq (n - |D(X)|)/(L - |T(X)|)$. Line 5 computes the set C' consisting of all such components, line 6 forms from C' the set C^* by removing from C' any components that originate from layers t such that $C \in G_t$ for some $C \in X$ (since only one component can be visited in each timestep, and thus we want X to be a set of components originating from unique layers of $\mathcal{G}$). We remark that a more efficient implementation could skip layers G_t with $t \in T(X)$ already when constructing C' in line 5, but the asymptotic running-time of the overall algorithm would not be affected by this change. The correctness of line 7 follows from Observation 16. To complete the proof, we claim that the value yes is returned by line 9 if and only if there exists a non-strict temporal exploration schedule starting at s that visits all the components contained in X; we proceed by reverse induction on $|X|$. Assume first that the return value of $g(\mathcal{G}, s, X')$ is correct for any X' with $|X'| = k$ ($k \in [L]$) and let $|X| = k - 1$. Now assume that, during the execution of $g(\mathcal{G}, s, X)$, line 9 returns yes; it follows that $g(\mathcal{G}, s, X') =$ yes for some $X' = X \cup C$ with $C \in C^*$ and thus it follows from the induction hypothesis that there exists a non-strict temporal exploration schedule that starts at s and visits all the components contained in X, as required. In the other direction, assume that there exists some non-strict exploration schedule W that starts at s in timestep 1 and visits all the components in X. Note that, since the execution has reached line 9, we surely have $|C^*| > 0$; since we also have $|X| < L$ and

$|D(X)| < n$ it follows from Observation 16 that W visits at least one additional component $C \in C^*$. Then, by the induction hypothesis, we must have $g(\mathcal{G}, s, X \cup \{C\}) =$ yes; thus when the loop of lines 8–10 processes $C \in C^*$ the algorithm will return yes as required. ◀

▶ **Theorem 20.** *It is possible to decide any instance $I = (\mathcal{G}, s, L)$ of NS-TEXP-L in $O(L(L!)^2 n)$ time.*

Proof. The algorithm simply returns the value of function call $g(\mathcal{G}, s, \emptyset)$ (Algorithm 1).

By Lemma 19, $g(\mathcal{G}, v, X)$ returns yes if and only if $\mathcal{G}$ admits a non-strict exploration schedule that starts at v and visits at least the components contained in the set X (which contains $|X| \in [0, L]$ components originating from unique layers of $\mathcal{G}$), and returns no otherwise. Thus the correctness of the above follows immediately.

In order to bound the running time of the above algorithm, it suffices to bound the running time of Algorithm 1, i.e., the recursive function g. The initial call is $g(\mathcal{G}, s, \emptyset)$, and each recursive call is of the form $g(\mathcal{G}, s, X)$ where X is a set of components with size one more than the input set of the parent call. Hence, line 1 ensures that there are at most L levels of recursion in total (not including the level containing the initial call). For a call at level $i \geq 0$, the set C^* constructed in line 5 has size at most $(L-i)^2$, since at most $L-i$ components can cover at least $(n - |D(X)|)/(L-i)$ of the vertices in $V(\mathcal{G}) - D(X)$ during each of the $L-i$ steps $t \in [L] - T(X)$. Thus each call at level $i \geq 0$ makes at most $(L-i)^2$ recursive calls. The tree of recursive calls thus has at most $(L!)^2$ nodes at depth L, and hence $O((L!)^2)$ nodes in total. It follows that the overall number of calls is bounded by $O((L!)^2)$.

Next, note that if some level-i call $g(\mathcal{G}, s, X)$ is such that $|X| < L$ and $|D(X)| < n$, then line 5 computes the set C', which can be achieved in $O(Ln)$ time by, for each $t \in [L]$, scanning over the components $C \in G_t$ (which collectively contain n vertices) and adding a component $C \in G_t$ to C' if and only if $|C - D(X)| \geq (n - |D(X)|)/(L-i)$. (Note that we can maintain a map from V to $\{0, 1\}$ that records for each vertex v whether $v \in D(X)$, and hence the value $|C - D(X)|$ can be computed in $O(|C|)$ time.) To compute the set C^* in line 6 we can follow a similar approach: for each $t \in [L] - T(X)$ ($|[L] - T(X)| = L - i$), add a component $C \in G_t$ to C^* if and only if it satisfies $C \in C'$. This requires $O((L-i)n) = O(Ln)$ time, and thus lines 5–6 take $O(Ln)$ time in total. Additionally, the return value of each recursive call is checked by the foreach loop (line 9) of its parent call in $O(1)$ time – this contributes an extra $O((L!)^2)$ time over all recursive calls. On the other hand, if a call $g(\mathcal{G}, s, X)$ is such that $|X| = L$ or $|D(X)| = n$, then line 2 computes $W^?_{\mathcal{G}}(s, X)$ in $O(Ln)$ time using Lemma 18. Thus in all cases the overall work per recursive call is $O(Ln)$, and the total amount of time spent before $g(\mathcal{G}, s, \emptyset)$ is returned is $O((L!)^2) \cdot O(Ln) = O(L(L!)^2 n)$, as claimed. ◀

3.2 W[2]-hardness of Set NS-TEXP for parameter L

Our aim in this section is to show that the SET NS-TEXP problem is W[2]-hard when parameterized by the lifetime L of the input graph. The reduction is from the well-known SET COVER problem with parameter k – the maximum number of sets allowed in a candidate solution. SET COVER is known to be W[2]-hard for this parameterization [7].

▶ **Definition 21** (SET COVER). *An instance of SET COVER is given as a tuple $(U, \mathcal{S}, k)$, where $U = \{a_1, \ldots, a_n\}$ is the ground set and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set of subsets $S_i \subseteq U$. The problem then asks whether or not there exists a subset $\mathcal{S}' \subseteq \mathcal{S}$ of size at most k such that, for all $i \in [n]$, there exists an $S \in \mathcal{S}'$ such that $a_i \in S$.*

For any instance I of SET COVER that we consider, we will w.l.o.g. assume that for each $i \in [n]$ we have $a_i \in S_j$ for some $j \in [m]$.

▶ **Theorem 22.** *Set NS-TEXP parameterized by L (the lifetime of the input non-strict temporal graph) is W[2]-hard.*

Proof. Let $I = (U = \{a_1, \ldots, a_n\}, \mathcal{S} = \{S_1, \ldots, S_m\}, k)$ be an arbitrary instance of Set Cover parameterized by k. We construct a corresponding instance $I' = (\mathcal{G}, s, \mathcal{X})$ of Set NS-TEXP as follows: Let $V(\mathcal{G}) = \{s\} \cup \{x_j : j \in [m]\} \cup \{y_{i,j} : j \in [m], a_i \in S_j\}$, and define $X_i = \{y_{i,j} \in V(\mathcal{G}) : j \in [m]\}$ $(i \in [n])$ and $\mathcal{X} = \bigcup_{i\in[n]}\{X_i\}$. We set the lifetime L of $\mathcal{G}$ to $L = 2k$ and specify the components for each timestep $t \in [2k]$ as follows: In all odd steps let one component be $\{s\}\cup\{x_j : j \in [m]\}$ and let all other vertices belong to components of size 1. In even steps, for each $j \in [m]$ let there be a component $\{y_{i,j} \in V(\mathcal{G}) : i \in [n]\} \cup \{x_j\}$ and let s form a component of size 1. Since $|V(\mathcal{G})| \leq 1+m+mn = O(mn)$, $|\bigcup_{i\in[n]} X_i| = O(mn)$ and $L = 2k$ we have that the size of instance I' is $|I'| = O(kmn)$ and the parameter L is bounded solely by a function of instance I's parameter k, as required. To complete the proof, we argue that I is a yes-instance if and only if I' is a yes-instance:

($\Longrightarrow$) Assume that I is a yes-instance; then there exists a collection of sets $\mathcal{S}' \subseteq \mathcal{S}$ of size $|\mathcal{S}'| = k' \leq k$ and, for all $i \in [n]$, there exists $S \in \mathcal{S}'$ with $a_i \in S$. Let $S_{j_1}, S_{j_2}, \ldots, S_{j_{k'}}$ be an arbitrary ordering of the sets in $\mathcal{S}'$; note that $j_i \leq m$ for all $i \in [k']$. We construct a non-strict temporal walk W in $\mathcal{G}$ as follows: Starting at vertex s, for every $l \in [1, k']$, during timestep $t = 2l - 1$ visit all vertices in the current component then finish timestep $2l - 1$ positioned at x_{j_l}. The component occupied during step $2l$ will be the one containing x_{j_l} – explore all vertices contained in that component and finish step $2l$ positioned at x_{j_l}. If $k' < k$, then spend the steps of the interval $[2k'+1, 2k]$ positioned in an arbitrary component. We claim that W visits at least one vertex in X_i for all $i \in [n]$. To see this, first note that for every $i \in [n]$ there exists an $S_j \in \mathcal{S}'$ such that $a_i \in S_j$. Hence, by our reduction, it follows that a vertex $y_{i,j}$ is contained in the component containing x_j during timestep $2l$ for every $l \in [k]$ and, by its construction, W visits the component containing x_j (and thus visits $y_{i,j} \in X_i$) during timestep $2l^*$ for some l^* such that $j_{l^*} = j$. Since this holds for all $i \in [n]$ it follows that W is a feasible solution and I' is a yes-instance.

($\Longleftarrow$) Assume that I' is a yes-instance and that we have some non-strict temporal walk W that visits at least one vertex in X_i for all $i \in [n]$. We first claim that W visits any vertex of the form $y_{i,j}$ for the first time during an even step. To see this, observe that every $y_{i,j}$ lies disconnected in its own component in every odd step t, and so to visit any $y_{i,j}$ in an odd step W would need to occupy the component containing $y_{i,j}$ during step $t-1$ and finish step $t-1$ positioned at $y_{i,j}$; hence $y_{i,j}$ was already visited in step $t-1$, which is even. Therefore, in order for W to visit any $y_{i,j}$ it must be positioned, during at least one even step, at the component containing x_j. Now, to construct a collection of subsets $\mathcal{S}' \subseteq \mathcal{S}$ with size $x \leq k$, let $\mathcal{S}' = \{S_j : W$ visits the component containing x_j during some even timestep$\}$. To see that $\mathcal{S}'$ is a cover of U with size $x \leq k$, observe that W visits at least one vertex $y_{i,j}$ for every $i \in [n]$; thus, by the reduction, for every $i \in [n]$ the element a_i is contained in set S_j for some $S_j \in \mathcal{S}'$. It follows that the union of $\mathcal{S}'$'s elements covers U, and so I is a yes-instance. ◀

4 Conclusion

In this paper we have initiated the study of temporal exploration problems from the viewpoint of parameterized complexity. For both strict and non-strict temporal walks, we have shown several variants of the exploration problem to be in FPT. For the variant where we are given a family of vertex subsets and need to visit only one vertex from each subset, we have shown W[2]-hardness for the non-strict model for parameter L. (In the strict model, one can show that W[2]-hardness holds for this problem even in the case where each layer of the temporal graph is a complete graph.) An interesting question for future work is whether NS-TEXP is in FPT if the parameter is the maximum number of components in any layer.

References

1 Eleni C. Akrida, Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul G. Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46–60, 2019. doi:10.1016/j.jcss.2019.02.003.

2 Eleni C. Akrida, George B. Mertzios, and Paul G. Spirakis. The temporal explorer who returns to the base. In Pinar Heggernes, editor, *11th International Conference on Algorithms and Complexity (CIAC 2019)*, volume 11485 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2019. doi:10.1007/978-3-030-17402-6_2.

3 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020. doi:10.1016/j.jcss.2019.08.002.

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

5 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.

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

7 Bonnet, Édouard, Paschos, Vangelis Th., and Sikora, Florian. Parameterized exact and approximation algorithms for maximum k-set cover and related satisfiability problems. *RAIRO-Theoretical Informatics and Applications*, 50(3):227–240, 2016. doi:10.1051/ita/2016022.

8 Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Online and offline algorithms for the time-dependent TSP with time zones. *Algorithmica*, 39(4):299–319, 2004. doi:10.1007/s00453-004-1088-z.

9 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.

10 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. In Paola Flocchini and Lucia Moura, editors, *32nd International Workshop on Combinatorial Algorithms (IWOCA 2021)*, volume 12757 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2021. doi:10.1007/978-3-030-79987-8_8.

11 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.

12 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. doi:10.1007/s00453-021-00831-w.

13 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.

14 Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2000.

15 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

16 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.

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

18 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. doi:10.1137/0110015.

19 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002. `doi:10.1006/jcss.2002.1829`.

20 George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.STACS.2020.27`.

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

22 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016. `doi:10.1016/j.tcs.2016.04.006`.

23 Hendrik Molter. *Classic graph problems made temporal – a parameterized complexity analysis*. PhD thesis, Technical University of Berlin, Germany, 2020. URL: `https://nbn-resolving.org/urn:nbn:de:101:1-2020120901012282017374`.

24 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *IEEE 36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 182–191. IEEE Computer Society, 1995. `doi:10.1109/SFCS.1995.492475`.

25 Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

26 Claude E. Shannon. Presentation of a maze-solving machine. In Neil James Alexander Sloane and Aaron D. Wyner, editors, *Claude Elwood Shannon: Collected Papers*, pages 681–687. IEEE Press, 1993.

27 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, May 2014. `doi:10.14778/2732939.2732945`.

28 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`.