Acta Informatica (2022) 59:387–407
https://doi.org/10.1007/s00236-022-00421-5

ORIGINAL ARTICLE

Exploration of k-edge-deficient temporal graphs

Thomas Erlebach[1] · **Jakob T. Spooner**[2]

Received: 10 September 2021 / Accepted: 28 March 2022 / Published online: 26 August 2022

Abstract
A temporal graph with lifetime L is a sequence of L graphs $G_1, \ldots, G_L$, called layers, all of which have the same vertex set V but can have different edge sets. The underlying graph is the graph with vertex set V that contains all the edges that appear in at least one layer. The temporal graph is always connected if each layer is a connected graph, and it is k-edge-deficient if each layer contains all except at most k edges of the underlying graph. For a given start vertex s, a temporal exploration is a temporal walk that starts at s, traverses at most one edge in each layer, and visits all vertices of the temporal graph. We show that always-connected, k-edge-deficient temporal graphs with sufficient lifetime can always be explored in $O(kn \log n)$ time steps. We also construct always-connected, k-edge-deficient temporal graphs for which any exploration requires $\Omega(n \log k)$ time steps. For always-connected, 1-edge-deficient temporal graphs, we show that $O(n)$ time steps suffice for temporal exploration.

1 Introduction

Given a simple, connected, undirected graph G and a *start vertex* $s \in V(G)$, the task of *exploring* G, i.e. computing a sequence of consecutively crossed edges $e \in E(G)$ that begins at s and visits every vertex $v \in V(G)$ at least once, is both natural and well-understood. A closely related problem was initially considered by Shannon [23], who designed a mechanical maze-solving machine which implemented a depth-first search-type technique in order to locate, within a given maze, a prespecified goal. This 'searching' problem is indeed related to graph exploration: if our task is to simply complete an exploration of G, then a solution can be straightforwardly found by performing a depth-first search (DFS) starting from s and

An extended abstract of a preliminary version of this paper has been presented at the 17th Algorithms and Data Structures Symposium (WADS 2021) [10].
T. Erlebach's research was supported by EPSRC Grants EP/S033483/1 and EP/T01461X/1.

✉ Thomas Erlebach
thomas.erlebach@durham.ac.uk

Jakob T. Spooner
jts21@leicester.ac.uk

[1] Department of Computer Science, Durham University, Durham, England

[2] School of Computing and Mathematical Sciences, University of Leicester, Leicester, England

stopping once all vertices have been visited at least once—clearly this requires $\Theta(n)$ edge traversals in total, where $n = |V(G)|$.

The graph exploration problem in the context of temporal graphs (i.e. graphs whose edge set can change over time) has also received significant attention in recent years [1, 2, 6–9, 21]. This problem, known as Temporal Exploration (TEXP), but restricted to *k-edge-deficient* temporal graphs (which we define formally later) is the focus of this paper. Given a temporal graph $\mathcal{G}$, the problem asks that we compute a temporal walk, starting at some prespecified vertex $s \in V(\mathcal{G})$, that makes at most a single edge traversal in each time step, and that visits all vertices at least once by the earliest time possible. We formally define the problem and temporal graph model in Sect. 2, but refer the interested reader to [5, 20] for more on temporal graphs in general, or [6, 21] for more details on TEXP. In a rather general setting, TEXP makes no assumptions about the input temporal graph, aside from the assumption that it is connected in each time step (i.e. *always-connected*), which ensures exploration is always possible provided the temporal graph has a sufficient lifetime [21]. This general setting allows an arbitrary number of edges from the underlying graph to be missing in each time step, and thus the graphs in different time steps can differ substantially, which leads to pessimistic bounds on the worst-case exploration time: it was shown by Erlebach et al. [6] that there are always-connected temporal graphs with n vertices that require $\Theta(n^2)$ time steps to be explored. The construction of temporal graphs with such a large exploration time from [6] uses an underlying graph that is dense (it has $\Theta(n^2)$ edges), whilst the graph in each time step is a star (with $n - 1$ edges). Thus, in each time step a quadratic number of edges of the underlying graph are missing from the graph of that time step. Therefore, it is interesting to study the question whether better exploration times can be guaranteed if the number of missing edges in each time step is small. To study this question, we also consider always-connected temporal graphs but, in contrast to previous work, we consider k-edge-deficient temporal graphs whose structure in each step is 'close' to that of its underlying graph, in the sense that at most k edges are missing. Such graphs were previously considered by Gotoh et al. [14] in a distributed setting. We assume that the temporal structure of an input temporal graph is known in full to an algorithm prior to it computing a solution, as opposed to a setting in which the structure of the graph in each step is revealed online and over time.

1.1 Our contribution

We consider Temporal Exploration on always-connected temporal graphs that are k-edge-deficient for some $k \in \mathbb{N}$. We define the property formally in Sect. 2, but essentially these are temporal graphs $\mathcal{G}$ with *underlying graph* G such that, during each time step t of $\mathcal{G}$'s lifetime, there are at most k edges $e \in E$ in the underlying graph that are untraversable in (or 'missing' from) $\mathcal{G}$. Let $n = |V(G)|$. In Sect. 3, we prove for arbitrary $k \in \mathbb{N}$ that k-edge-deficient always-connected temporal graphs can be explored in $O(kn \log n)$ time steps. In Sect. 4, we additionally show that 1-edge-deficient graphs can always be explored in $51n$ time steps, by giving a recursive exploration algorithm that exploits a number of existing structural/algorithmic results originating from traditional graph theory. In Sect. 5, we present a modification of an existing $\Omega(n \log n)$ lower bound on the number of time steps required to explore always-connected temporal graphs with planar underlying graph of maximum degree ≤ 4, presented in [6], that allows us to obtain an $\Omega(n \log k)$ bound on the worst-case time required to explore arbitrary always-connected k-edge-deficient temporal graphs. Finally, we conclude and point to directions for future work in Sect. 6.

1.2 Related work

Bui-Xuan et al. [4] propose multiple objectives for optimization when computing temporal walks/paths from one vertex to another: e.g. *fastest* (fewest steps used) and *foremost* (arriving at the destination at the earliest time possible). Brodén et al. [3] consider the TEMPORAL TRAVELLING SALESPERSON PROBLEM on a complete graph with n vertices, with edge costs that can differ between 1 and 2 in each time step. They show that when an edge's cost changes at most k times over 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 [21] prove APX-hardness and provide a $(1.7+\epsilon)$-approximation. They also consider the decision version of the TEMPORAL EXPLORATION problem, which asks whether or not a given temporal graph admits a temporal walk that visits all vertices at least once. They show that the problem is NP-complete when no restrictions are placed on the input; they also propose considering the problem under the *always-connected* assumption, which ensures that exploration is possible provided the lifetime of the input graph is sufficiently long [21]. Erlebach et al. [6] further consider the optimization variant of the TEMPORAL EXPLORATION problem under the always-connected assumption. They prove an $\Omega(n^2)$ lower bound on the time needed to explore general always-connected temporal graphs, and provide a proof that temporal graphs within this class can be explored in n^2 steps. They also prove a number of bounds on the number of time steps required to explore temporal graphs from various restricted temporal graph classes. Bodlaender and van der Zanden [2] examine TEXP when restricted to temporal graphs whose underlying graph has pathwidth at most 2, showing that it is NP-complete to decide if a temporal exploration schedule with a given arrival time exists even under this very limiting restriction. In [17, 18], Ilcinkas et al. consider TEXP restricted to temporal graphs with underlying cycle or cactus graphs, respectively. Akrida et al. [1] consider RETURN-TO- BASE TEXP in which a candidate solution must return to the vertex from which it initially departed. Erlebach et al. [7] prove an $O(dn^{1.75})$ bound on the number of time steps required to explore any temporal graph with degree bounded by d in each step, a considerable improvement over the previously best known $O(\frac{n^2 \log d}{\log n})$ bound [8]. In [9], a *non-strict* variant of TEXP is studied—here, a computed walk may make an unlimited number of edge traversals in each given time step. Notions of strict/non-strict paths which, respectively, allow for a single edge/unlimited number of edge(s) to be crossed in any time step have been considered before, notably by Kempe et al. [19] and Zschoche et al. [24]. In this paper, we only consider strict temporal walks. Gotoh et al. [15] consider TEXP on temporal graphs with underlying cycle under the so-called (H, S)-view, in which only the availability of edges at most H hops away for at most the next S time steps is known to an algorithm. Exploration of k-edge-deficient temporal graphs is studied by Gotoh et al. [14] in a distributed setting. They prove bounds on the number of *cooperating mobile agents* required to ensure that temporal graph exploration can be achieved when the vertices of a given k-edge-deficient temporal graph are anonymous (i.e. have no unique identifiers). They consider this problem under two distinct 'agent scheduler' models, one in which all agents are active during each time step (*fully synchronous*), and one in which an arbitrary subset of the agents are active during each time step (*semi-synchronous*)—in either model the agents are able to observe, in each time step, their current vertex v along with the vertices that are adjacent to their current vertex, as well as store information about this in a 'notebook'. They are also able to write information to a 'white board' stored at vertex v, that other agents can subsequently observe and write to themselves. In the semi-synchronous case, it is shown that $2k + 1$ agents are enough to ensure that exploration can be performed by the mobile agents

in a k-edge-deficient graph. For the fully synchronous case, they show that $2k$ agents are enough as long as the size (i.e. order plus number of edges) of the input temporal graph and the number of cooperating agents is known *a priori* by all agents.

2 Preliminaries

For a positive integer i, we use the notation $[i]$ as short-hand for the set $\{1, 2, \ldots, i\}$.

Definition 1 (Temporal graph) A temporal graph $\mathcal{G}$ with *lifetime* L is a sequence of static graphs $\mathcal{G} = \langle G_1, G_2, \ldots, G_L \rangle$ that all have the same vertex set V. We let $V(\mathcal{G}) := V$ and $n := |V(\mathcal{G})|$, and say $\mathcal{G}$ is of order n. The subscripts $t \in [L]$ indexing the graphs in the sequence are the discrete time steps (or simply 'steps') 1 to L. Each G_t (which may be referred to as the tth *layer*) represents the structure of $\mathcal{G}$ in time step t. The *underlying graph* $G = (V, E)$ of $\mathcal{G}$ is the graph with vertex set $V = V(\mathcal{G})$ and edge set $E = \bigcup_{t \in [L]} E_t$.

An *edge-time pair* of $\mathcal{G}$ is a pair (e, t) such that $t \in [L]$ and $e \in E(G_t)$. If $(\{u, v\}, t)$ is an edge-time pair and an agent is located at u at the start of time step t, the agent can traverse the edge $\{u, v\}$ in time step t and arrive at v at the end of time step t.

Definition 2 (Temporal walk) A temporal walk W in a temporal graph $\mathcal{G}$ is a finite sequence of vertices alternating with edge-time pairs

$$W = (v_1, (e_1, t_1), v_2, (e_2, t_2), \ldots, v_{l-1}, (e_{l-1}, t_{l-1}), v_l)$$

for some positive integer l, such that $t_1 < t_2 < \cdots < t_l$ and $e_i = \{v_i, v_{i+1}\}$ for all $i \in [l-1]$. The walk starts at vertex v_1 and ends at vertex v_l. Its arrival time, denoted by $\alpha(W)$, is the time step in which it traverses its last edge, i.e. t_{l-1}.

A temporal walk $W = (v_1, (e_1, t_1), v_2, \ldots, v_{l-1}, (e_{l-1}, t_{l-1}), v_l)$ in a temporal graph $\mathcal{G}$ is an *exploration schedule* for start vertex s if $v_1 = s$ and $\{v_1, v_2, \ldots, v_l\} = V(\mathcal{G})$. In the optimization variant F- TEXP (foremost TEXP) of TEXP, we are interested in determining, for a given temporal graph G and start vertex s, an exploration schedule W for start vertex s with minimum arrival time $\alpha(W)$.

A temporal graph $\mathcal{G} = \langle G_1, \ldots, G_L \rangle$ is *always-connected* if and only if G_t is connected for all $t \in [L]$. We consider always-connected temporal graphs with lifetime $L \geq (n-1)^2$. The following lemma from [6] will be useful and implies that such temporal graphs can always be explored in at most $(n-1)^2$ steps by repeatedly moving to a previously unvisited vertex in at most $n-1$ steps.

Lemma 1 (Reachability Lemma; Erlebach et al. [6]) *Let $\mathcal{G}$ be an arbitrary always-connected temporal graph with vertex set V, $n = |V| \geq 2$, and lifetime L. Then, an agent situated at any vertex $u \in V$ at the beginning of some time step $t \leq L - n + 2$ can reach any other vertex $v \in V$ in at most $|V| - 1 = n - 1$ steps, i.e. by the end of time step $t + n - 2$.*

We now give a formal definition of what it means for a temporal graph $\mathcal{G}$ to be k-edge-deficient:

Definition 3 (k-**edge-deficient**) Let $\mathcal{G} = \langle G_1, \ldots, G_L \rangle$ be a temporal graph with underlying graph $G = (V, E)$. Then $\mathcal{G}$ is k-edge-deficient (for $k \in \mathbb{N}$) if, for all $t \in [L]$, we have $G_t = (V, E - X_t)$ for some $X_t \subseteq E$ with $|X_t| \leq k$.

In always-connected k-edge-deficient temporal graphs, we have that $G_t = (V, E - X_t)$ is connected for all $t \in [L]$.

When constructing a walk in a k-edge-deficient temporal graph $\mathcal{G}$, we may speak of an agent *following* a walk W in the underlying graph G. By this, we mean that the agent traverses in $\mathcal{G}$ the edges in the same order as they are traversed by W, and does this whenever it is possible to do so, i.e. whenever the next edge e traversed by W is present in the current time step t. If that edge is not present, the agent is *blocked on e in step t*.

Roughly speaking, our approaches to proving exploration results for both k-edge-deficient and 1-edge-deficient temporal graphs involve splitting the lifetime of the input graph into consecutive periods of time steps, then allowing a set of *virtual agents* to occupy and attempt to explore disjoint substructures of the same temporal graph during each period. We then allow a *real agent* to follow the 'best' virtual agent (best in the sense that this agent has successfully completed the exploration of their respective substructure) in any such period. Whilst this same generic approach is employed in some sense or another whilst proving both results, the graph-theoretic machinery that enables the approach to work is decidedly different in both cases. We remark that the difference between the multi-agent approach we employ here and the multi-agent approaches of [14] is that, here, we use multiple virtual agents as a conceptual tool for proving the existence (and construction) of a single exploration schedule that can then be followed by a single real agent. In [14], however, the multiple agents are an integral part of the problem, as are the varying levels of information that the agents can retain and share with one another.

3 Exploration algorithm for k missing edges

We present an algorithm that proceeds in rounds. In each round, it considers a forest consisting of $k+1$ edge-disjoint subtrees of a spanning tree of the underlying graph and ensures that all edges of one of these trees can be traversed in the round. The following lemma, shown by Frederickson and Johnson [12, Lemma 1], allows us to split a tree T into a pair of edge-disjoint subtrees (whose union covers $E(T)$) in a balanced way:

Lemma 2 (Frederickson and Johnson [12]) *Let T be a tree with $m \geq 2$ edges. Then, one can compute in time $O(m)$ two edge-disjoint subtrees T' and T'' such that $|E(T')|, |E(T'')| \in [m/3, 2m/3]$, and such that $E(T') \cup E(T'') = E(T)$.*

Say that a set S of edge-disjoint subtrees $T' \subseteq F$ is a *subtree cover* of a forest F if, for every $e \in E(F)$ we have $e \in E(T')$ for some $T' \in S$. Call such a subtree cover S *balanced* if it satisfies the additional property that the largest tree in S contains at most three times the number of edges contained by the smallest. Our exploration method will require maintaining a subtree cover of size $k+1$ of the unexplored parts of the temporal graph, as this ensures that in each step at least one of the subtrees does not have a missing edge. The following lemma, which we prove by applying Lemma 2 to the largest tree in a balanced subtree cover, allows us to build a subtree cover of size $k+1$ from a spanning tree of the whole underlying graph, and to increase a subtree cover from size k to size $k+1$ each time one of the $k+1$ subtrees has been explored and thus removed from the subtree cover.

Lemma 3 *Let S be a balanced subtree cover of some forest F such that $|S| = x$ and $|E(F)| \geq x+1$ hold. Then, one can obtain a balanced subtree cover S' of F such that $|S'| = x+1$.*

Proof In this proof we write $|T|$ as short-hand for $|E(T)|$ for any tree T. Consider the largest tree T in S and apply to it Lemma 2 to obtain two edge-disjoint subtrees T' and T'' such that $|T'|, |T''| \in [|T|/3, 2|T|/3]$. Note that this is possible because $|E(F)| \geq x+1$, and since S

contains exactly x subtrees we have that the largest tree in S contains at least two edges, as is required by Lemma 2. Let $S' = (S - \{T\}) \cup \{T', T''\}$. We have that S' is a subtree cover of F, since S is a subtree cover of F, and since T' and T'' are edge-disjoint and cover $E(T)$. Furthermore, since S is balanced and T contains the most edges of any tree in S, we have that every tree in $S - \{T\}$ contains a number of edges in the range $[|T|/3, |T|]$. Moreover, we have $|T'|, |T''| \in [|T|/3, 2|T|/3]$, and so $|T'|, |T''| \in [|T|/3, |T|]$. Concluding, we have that the number of edges in each tree of S' lies in the range $[|T|/3, |T|]$, and so S' is balanced; it is clear from the construction of S' that $|S'| = x + 1$. □

Theorem 1 *Let $\mathcal{G} = \langle G_1, \ldots, G_L \rangle$ be an always-connected, k-edge-deficient temporal graph (for some integer-valued function k of n) with underlying graph G, lifetime $L \geq (n-1)^2$ and order n. Then, there is a bound $\tau = O(kn \log n)$ such that, for any start vertex s, there is an exploration schedule W of $\mathcal{G}$ that starts at $s \in V(\mathcal{G})$ and has arrival time $\alpha(W) \leq \min\{\tau, (n-1)^2\}$. Moreover, such a schedule can be computed in polynomial-time.*

Proof For $k \geq n - 1$, the result clearly holds as every always-connected temporal graph can be explored in at most $(n-1)^2$ time steps (by repeated application of Lemma 1), so we assume $k < n - 1$ for the rest of the proof.

Compute an arbitrary spanning tree T of G, and let $m = |E(T)| = n - 1$. Assume w.l.o.g. that $m \geq 4$, otherwise G can be explored in $O(1)$ steps using $O(1)$ applications of Lemma 1. Let $S = \{T\}$ and note that S is a balanced subtree cover of T. Now apply Lemma 3 to S k times to obtain a balanced subtree cover S^* of size $k + 1$ (this is possible since $k \leq n - 2$). In the following, we let F denote a forest (represented as a single graph) containing all edges of T that may not yet have been traversed, initially $F = T$.

We now specify our algorithm in terms of an agent that explores the graph in consecutive *rounds*. We denote by $t_{(j)}$ the first step of the jth round, and by $s_{(j)}$ the vertex at which the agent is positioned at the beginning of time step $t_{(j)}$. The balanced subtree cover at the start of the jth round is denoted by $S^*_{(j)}$, and the forest made up of edges that may not yet have been explored by $F_{(j)}$. Let $m'_{(j)} = \sum_{T_i \in S^*_{(j)}} |E(T_i)|$. Initially, we have $j = 1$, $t_{(1)} = 1$, $s_{(1)} = s$, $F_{(1)} = T$, $S^*_{(1)} = S^*$ is a balanced subtree cover of $F_{(1)}$ (with size $k + 1$), and $m'_{(1)} = m$. If $F_{(j)}$ contains more than $k + 1$ edges, execute the jth round as follows: consider the graph from step $t_{(j)} + n$ onward, and place a single virtual agent at an arbitrary vertex v_i in each of the $k + 1$ subtrees $T_i \in S^*_{(j)}$. Now, for each $i \in [k + 1]$, compute a DFS tour in T_i starting from v_i, then let the agent positioned in T_i follow that DFS tour whenever it is possible to do so for the following $6m'_{(j)}$ steps (i.e. whenever the next edge they are required to traverse is present in the current time step). Since there are $k + 1$ virtual agents following tours in edge-disjoint subtrees, and since $\mathcal{G}$ is k-edge-deficient, it follows that there are no edges missing from at least one subtree $T' \in S^*_{(j)}$ in every step. Let T_{i^*} be the subtree that had no edges missing during the largest number of steps in the considered $6m'_{(j)}$-step period. Then, T_{i^*} had no edge missing in at least $\frac{6m'_{(j)}}{k+1}$ steps. Since $|S^*_{(j)}| = k + 1$, the smallest tree in $S^*_{(j)}$ cannot contain strictly more than $\frac{m'_{(j)}}{k+1}$ edges, so because $S^*_{(j)}$ is balanced the largest tree in $S^*_{(j)}$ contains at most $\frac{3m'_{(j)}}{k+1}$ edges. Therefore, at least $\frac{6m'_{(j)}}{k+1}$ steps in which the virtual agent positioned in T_{i^*} is able to traverse an edge are enough for that agent to complete their DFS of T_{i^*} and arrive back at v_{i^*}. Using the steps in the interval $[t_{(j)}, t_{(j)} + n - 1]$, move the real agent, using Lemma 1, from $s_{(j)}$ to the vertex v_{i^*} at which the virtual agent began their tour of T_{i^*}. Let W^* be the tour followed by the virtual agent positioned in T_{i^*}; from step $t_{(j)} + n$ to step $t'_{(j)} = t_{(j)} + n + 6m'_{(j)} - 1$, let the real agent complete W^*. Once completed,

check if $> k+1$ edges of T remain untraversed; if so, consider the set $S' = S^*_{(j)} - \{T_{i^*}\}$ and note that $|S'| = k$. Observe that S' is balanced since $S^*_{(j)}$ was balanced and removing a tree cannot violate this property. Since we have $S' = S^*_{(j)} - \{T_{i^*}\}$, and since $S^*_{(j)}$ covered T, we have that S' covers the forest F' obtained from $F_{(j)}$ by removing the edges of T_{i^*}. Apply Lemma 3 to S' to obtain a balanced subtree cover S'' of F' such that $|S''| = k+1$—note that this is valid since $|E(F')| > k+1 = |S'|+1$, as is required by Lemma 3. Now, set $S^*_{(j+1)} = S''$, $F_{(j+1)} = F'$, $s_{(j+1)} = v_{i^*}$ and $t_{(j+1)} = t'_{(j)} + 1$ and start the $(j+1)$th round as above. Once a round is completed and at most $k+1$ edges remain, stop and explore the up to $2k+2$ remaining unexplored vertices one by one, using $n-1$ steps per vertex by Lemma 1 and thus $O(nk)$ steps for all these up to $2k+2$ vertices together.

Note that every vertex v in $V(T) = V(G)$ either (1) belongs to an edge of T that was traversed by the algorithm, or (2) was visited via an application of Lemma 1. Hence, the computed walk is an exploration schedule and it remains only to bound its arrival time. In each round j, a subtree containing at least a $\frac{1}{3(k+1)}$ fraction of the edges of $F_{(j)}$ is traversed in its entirety. To see this, observe that $|S^*_{(j)}| = k+1$, so the largest tree in $S^*_{(j)}$ must contain $\geq \frac{m'_{(j)}}{k+1}$ edges; because $S^*_{(j)}$ is balanced, it follows that all trees in $S^*_{(j)}$ have size $\geq \frac{m'_{(j)}}{3(k+1)}$. Hence, after x rounds, the number of edges of T that remain in $F_{(x+1)}$ is $\leq m(1-\frac{1}{3(k+1)})^x$. Thus, after $x = 3(k+1)\ln(\frac{m}{k+1}) = O(k\log m) = O(k\log n)$ (recall that $m = |E(T)| = n-1$) rounds there are $\leq m(1-\frac{1}{3(k+1)})^{3(k+1)\ln(\frac{m}{k+1})} \leq k+1$ unexplored edges remaining in $F_{(x+1)}$. As each round takes $n + 6m'_{(j)} \leq n + 6m = O(n)$ steps, the total number of steps after $O(k\log n)$ rounds is $O(kn\log n)$. A further at most $(2k+2)n$ steps are sufficient to explore the up to $2k+2$ remaining unvisited vertices. Hence, the constructed exploration schedule, call it W, has an arrival time of $\alpha(W) \leq O(kn\log n) + (2k+2)n = O(kn\log n)$ as required. If this bound on the arrival time is greater than $(n-1)^2$, use instead the exploration schedule with arrival time $(n-1)^2$ that follows from repeated application of Lemma 1.

Finally, it is easy to see that the algorithm for determining the exploration schedule can be implemented to run in polynomial time. □

We remark that in Theorem 1 we have assumed that the given temporal graph has lifetime at least $(n-1)^2$ as this assumption ensures that an exploration schedule exists for arbitrary always-connected temporal graphs. For values of k for which the bound $O(kn\log n)$ of the theorem evaluates to a value $\lambda < (n-1)^2$, it is of course sufficient to assume that the given temporal graph has lifetime at least λ.

As a consequence of Theorem 1, we get the following corollary:

Corollary 1 *Let $\mathcal{G}$ be an arbitrary, always-connected k-edge-deficient temporal graph with lifetime $L = (n-1)^2$ and let $k = o(\frac{n}{\log n})$. Then, for any start vertex $s \in V(\mathcal{G})$, $\mathcal{G}$ necessarily admits an exploration schedule W that starts at s and has arrival time $\alpha(W) = o(n^2)$.*

Let k be an arbitrary function of n and consider the restriction of F- TEXP (the optimization variant of TEXP) to k-edge-deficient temporal graphs. Then, since the algorithm of Theorem 1 is constructive and runs in polynomial time, and since the optimal exploration schedule cannot have arrival time smaller than $n-1$, we arrive at the following:

Theorem 2 *There exists a polynomial-time $O(k\log n)$-approximation algorithm for* F- TEXP *on k-edge-deficient temporal graphs.*

4 Exploration algorithm for a single missing edge

We now present our algorithm for the case when $k = 1$. We remark that exploration in this case feels, intuitively speaking, 'close' to exploration in static graphs, since there is just a single edge missing in each time step. This appears to be evidenced by the relative ease (compared to less restrictive cases such as k-edge-deficient for general k or bounded-degree temporal graphs) with which we can employ definitions/results from static graph theory in order to obtain our algorithm. We now provide some preliminary discussion before specifying the main algorithm of this section.

First, we aim to sparsify the given temporal graph because the exploration time of our method increases with the number of edges of the underlying graph. To do so, we will simply ignore some of the edges of the given temporal graph and focus on the subgraph consisting of the remaining edges. As at most one edge of the underlying graph is missing in each time step, the same property also holds for the subgraph consisting of those remaining edges, and we can focus on that subgraph for the purposes of exploration. We only need to be careful to define the subgraph in such a way that it remains connected provided that at most one edge is missing.

For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *union* of G_1 and G_2, denoted by $G_1 \cup G_2$, is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$. This definition extends naturally to the union of a finite collection of graphs.

A graph $G = (V, E)$ is *k-vertex-connected* (or simply k-connected) if, for any subset $X \subseteq V(G)$ such that $|X| < k$, the subgraph of G induced by $V - X$ is connected. For $k = 2$, we say *biconnected* instead of 2-connected. Let $G = (V, E)$ be a connected graph. An edge $e \in E$ is a *bridge* if $G' = (V, E - \{e\})$ is disconnected. A graph $G = (V, E)$ is *2-edge-connected* if it is connected and does not contain a bridge. Observe that all bridges of the underlying graph of an always-connected temporal graph are necessarily present in every time step, as otherwise the graph in a time step with a missing bridge would not be connected.

A 2-edge-connected component (abbreviated 2-ecc) of a graph G is a vertex-maximal induced subgraph $C \subseteq G$ such that C is 2-edge-connected. Note that a 2-ecc can also be a single vertex. We say that a spanning subgraph G'' of G *preserves 2-edge-connectivity* if it contains all bridges of G and, for every 2-ecc C of G, the subgraph of G'' induced by $V(C)$ is 2-edge-connected. In order to show that every connected graph G has a spanning subgraph that preserves 2-edge-connectivity and has only a linear number of edges, we make use of the following result by Nagamochi and Ibaraki.

Theorem 3 (Nagamochi and Ibaraki [22]) *Every k-connected graph $G = (V, E)$ admits a k-connected spanning subgraph $G' = (V, E')$ such that $|E'| \leq k|V|$. Moreover, G' can be computed in $O(|E|)$-time.*

An induced subgraph G' of a graph G is a *block* (also known as *biconnected component*) of G if it is biconnected and maximal.

Definition 4 (**Modified block-cut tree construction**) Let $\mathcal{B}$ be the set of blocks of a given graph G. Construct a tree T as follows:

(1) For every block $B_i \in \mathcal{B}$ create a vertex $b_i \in V(T)$, and for every vertex $u \in V(G)$ create a vertex $c_u \in V(T)$.
(2) For every vertex/block pair (u, B_i) such that $u \in V(B_i)$, add an edge $\{c_u, b_i\} \in E(T)$.

The construction detailed in Definition 4 ensures that T is indeed a tree; it is essentially the same construction that is used to produce the *block-cut tree* of a given graph, as considered

in, for example, [13] and [16]. The construction considered in those studies is well known to have this property, and the only additional vertices produced by the modified construction presented above are the vertices in T that correspond to vertices $v \in V(G)$ that are not cut vertices in G. Since these vertices belong to exactly one block, their corresponding vertices in T must have degree one. Hence, the acyclicity of T cannot be violated by their inclusion. By applying Theorem 3 to each block of a given connected graph G, we can show the following:

Lemma 4 *Let G be an arbitrary connected graph and let $\mathcal{C}$ be the set of all 2-eccs of G. Then, G admits a spanning subgraph G^* such that (i) the vertices of each 2-ecc $C \in \mathcal{C}$ form a 2-ecc in G^* with at most $4|V(C)|$ edges, and (ii) $|E(G^*)| \leq 5|V(G)|$.*

Proof Consider the set $\mathcal{C}$ containing all maximal, non-trivial (i.e. order ≥ 2), 2-edge-connected components C_i in G. We say that a 2-ecc C_i in $\mathcal{C}$ *spans* a block B of G if B is a subgraph of C_i, i.e. $V(B) \subseteq V(C_i)$ and $E(B) \subseteq E(C_i)$. For every block B of G there is one 2-ecc in $\mathcal{C}$ that spans B, because B being biconnected implies that B is 2-edge-connected and hence a subgraph of a 2-ecc. Hence, each C_i in $\mathcal{C}$ spans one or more of the blocks of G—let $B(i)$ be the set of all blocks spanned by C_i and let $x_i = |B(i)|$. To the jth block $B_{i,j} \in B(i)$ ($j \in [x_i]$) apply Theorem 3, obtaining a spanning biconnected subgraph $B'_{i,j} \subseteq B_{i,j}$ such that $|E(B'_{i,j})| \leq 2|V(B_{i,j})|$. Now, since each $B'_{i,j}$ is connected and since every vertex $v \in V(C_i)$ satisfies $v \in V(B'_{i,j}) \subseteq V(B_{i,j})$ for some $j \in [x_i]$, the graph $C'_i = \bigcup_{j\in[x_i]} B'_{i,j}$ is connected and spans $V(C_i)$. Furthermore, since every edge of C'_i belongs to exactly one block $B'_{i,j}$ (otherwise there are 2 vertices in the intersection of two distinct blocks, which contradicts their maximality), we have:

$$|E(C'_i)| = \sum_{j\in[x_i]} |E(B'_{i,j})| \leq 2 \sum_{j\in[x_i]} |V(B_{i,j})| \leq 2 \sum_{v\in V(C_i)} |D(v)|,$$

where $D(v) = \{B_{i,j} : v \in V(B_{i,j})\}$. Note that the final inequality holds since $\sum_{j\in[x_i]} |V(B_{i,j})|$ counts each vertex $v \in V(C_i)$ exactly as many times as it occurs in unique blocks $B_{i,j}$. Now, in order to show that $\sum_{v\in V(C_i)} |D(v)| \leq 2|V(C_i)|$, first apply the construction of Definition 4 to C_i so as to produce the modified block-cut tree T_i of C_i. Consider now step (1) of Definition 4, and notice that it creates a vertex $c_u \in V(T)$ for every vertex $u \in V(C_i)$ and a vertex, say b_j, for every block $B_{i,j} \in B(i)$; hence $|V(T)| = x_i + |V(C_i)|$. We now claim that $x_i \leq |V(C_i)|$. To see this, note that the following procedure assigns a distinct vertex as representative for each block of C_i: pick an arbitrary vertex v in an arbitrary block $B_{i,j} \in B(i)$ and let v be the representative of $B_{i,j}$. Now carry out a DFS of C_i starting at v and, whenever the DFS traverses for the first time an edge $\{u, w\}$ (in the direction from u to w) of a block that does not have a representative yet, let w be the representative of that block (note that u is a cut vertex and w is a vertex that has not been visited before). This shows that $x_i \leq |V(C_i)|$. By step (2) in Definition 4, an edge $e = \{c_u, b_j\}$ satisfies $e \in E(T)$ if and only if $u \in V(C_i)$ satisfies $u \in V(B_{i,j})$. In other words, we have $\sum_{v\in V(C_i)} |D(v)| = |E(T)| \leq 2|V(C_i)| - 1$, and hence we have

$$|E(C'_i)| \leq 2 \sum_{v\in V(C_i)} |D(v)| \leq 4|V(C_i)|\,. \tag{1}$$

Let G^* be the graph obtained by taking the union of all C'_i and all bridge edges in $E(G)$. The above clearly establishes property (i) for G^*. Notice that the bridge edges in $E(G)$ are precisely the edges that are not contained in some C'_i—since G and each C'_i is connected, and G^* includes all bridge edges of G, it is not hard to see that G^* is connected. Let $x < n$

denote the number of bridges in G; then, since each edge in G^* is either a bridge or belongs to C'_i for exactly one $i \in [|\mathcal{C}|]$, we have

$$|E(G^*)| = \Big[\sum_{i\in[|\mathcal{C}|]} |E(C'_i)|\Big] + x \le 4 \cdot \Big[\sum_{i\in[|\mathcal{C}|]} |V(C_i)|\Big] + n \le 5n,$$

where the inequality follows from (1) and the final inequality holds since each vertex belongs to exactly one maximal 2-edge-connected component C_i, implying that $\sum_{i\in[|\mathcal{C}|]} |V(C_i)| \le n$. This establishes property (ii) and the proof is complete. □

If $\mathcal{G}$ is a 1-edge-deficient, always-connected temporal graph with underlying graph G, and if G^* is a spanning subgraph of G that preserves 2-edge-connectivity, then the temporal graph $\mathcal{G}^*$ with underlying graph G^* that is obtained from $\mathcal{G}$ by removing all edges that are not in G^* is also always-connected and 1-edge-deficient. This also implies that every cycle C of G^* induces a connected subgraph in every time step of $\mathcal{G}^*$.

A *circuit* C in a graph G is a closed walk in G that does not repeat edges. In 1-edge-deficient temporal graphs, a circuit in the underlying graph behaves like an always-connected temporal graph with underlying cycle, as at most one edge of the circuit can be missing in each step. Thus, we get the following theorem, which was shown in [6] for always-connected cycles.

Theorem 4 (Erlebach et al. [6]) *For every 1-edge-deficient temporal graph $\mathcal{G}$ with underlying circuit C, there exists a start vertex from which the graph can be explored in at most $|E(C)|-1$ steps.*

The following theorem of Fan allows us to reduce the exploration of a 2-ecc to the exploration of at most three circuits:

Theorem 5 (Fan [11]) *The edges of any 2-edge-connected graph $G = (V, E)$ can be covered by at most three circuits. Moreover, such a cover can be computed in $O(|E| \cdot |V|)$-time.*

In the proof of Theorem 6, we will also make use of the following lemma:

Lemma 5 *Let G be an arbitrary 2-edge-connected graph G with $n = |V(G)| \ge 3$ and let $u, u' \in V(G)$. Then, one can compute a circuit $X \subseteq G$ such that $u, u' \in V(X)$ and $|E(X)| \le 2n$.*

Proof Let x be the number of blocks B_i of G, then consider a shortest path $P \subseteq G$ from u to u'. Let $x' \le x$ be the number of distinct B_i visited by P, where we say that a path *visits* a block B_i if the path contains at least one edge in $E(B_i)$. Let $B_1, B_2 \ldots, B_{x'}$ be the ordered sequence of blocks visited by P; we have $u \in V(B_1)$, $u' \in V(B_{x'})$, and $|V(B_j) \cap V(B_{j+1})| = 1$ for $j \in [x']$. W.l.o.g., assume $x' \ge 2$, since if $x' = 1$ then u and u' are contained in the same block and we can simply select a cycle containing u and u'. Additionally, let $v_{i,i+1}$ be the unique vertex contained in $V(B_i) \cap V(B_{i+1})$ for all $i \in [x' - 1]$.

To construct a circuit $X \subset G$ such that $|E(X)| \le 2n$ we select: a cycle $D_1 \subseteq B_1$ of length $\le |V(B_1)|$ containing u and $v_{1,2}$; a cycle $D_i \subseteq B_i$ of length at most $|V(B_i)|$ containing $v_{i-1,i}$ and $v_{i,i+1}$ for all $i \in [2, x' - 2]$; and a cycle $D_{x'} \subseteq B_{x'}$ of length at most $|V(B_{x'})|$ containing $v_{x'-1,x'}$ and u'. Note that such cycles are always possible to obtain since each B_i is a biconnected subgraph of G. Let $D^* = \bigcup_{i\in[x']} D_i$. We construct a circuit X in D^* in the following way: select one of the two edge-disjoint paths, say P_1^1, between u and $v_{1,2}$ in D_1; let $P_1^2 \subseteq D_1$ be the path that was not selected. For all $i \in [2, x' - 2]$, select one of the two edge-disjoint paths, say P_i^1, between $v_{i-1,i}$ and $v_{i,i+1}$, and let $P_i^2 \subseteq D_i$ be the path that was not selected. Once $v_{x'-1,x'}$ is reached, do the same as before but pick a path $P_{x'}^1$ between

$v_{x'-1,x'}$ and u', and let $P^2_{x'} \subset D_{x'}$ be the path that was not selected. Let X be the walk formed by following all P^1_i (in the direction of u') in increasing order of index, then following all P^2_i (in the direction of u) in decreasing order of index. Since X begins at u, travels to u' and then returns to u, it is clear that X is a closed walk in $D^* \subseteq G$. Furthermore, since all paths P^1_i and P^2_i are pairwise edge-disjoint, it is clear that X does not repeat edges. It follows that X is a circuit in G.

Observe that X visits exactly the vertices in $V(D^*)$ and traverses every edge in $E(D^*)$ exactly once. Hence, to bound the number of edges in $E(X)$ it suffices to bound the number of edges in $E(D^*)$. Note that, for all $i \in [x' - 1]$, the degree of $v_{i,i+1}$ in D^* is 4 since $v_{i,i+1}$ has degree 2 in D_i and degree 2 in D_{i+1}; also note that the degree of every other vertex in D^* is exactly 2 (since these vertices each belong to exactly one D_i). Since there are at most $n - x' + 1$ vertices of degree 2 in D^*, we have that the sum of the degrees of all vertices in D^* is at most $4(x' - 1) + 2(n - x' + 1) \leq 4n$; halving this quantity gives that $|E(D^*)| = |E(X)| \leq 2n$ as required. □

Note that the edges which belong to no 2-ecc of an arbitrary connected graph G are precisely the bridges of G. Hence, one can represent the structure of G as a tree T, called the 2-*ecc tree* of G, by identifying each 2-ecc with a vertex, and joining two vertices by an edge in T if and only if their corresponding 2-eccs are connected by a bridge in G. In the proof of Theorem 6, we will therefore reuse standard terminology for trees: we choose a 2-ecc C as the *root component*. If C' and C'' are 2-eccs such that C' lies on the path from C to C'' in T, then C'' is a *descendant* of C'. If C' and C'' correspond to neighbouring nodes in T and C'' is a descendant of C', then C'' is a *child* of C' and C' is the *parent* of C''. The *subtree* rooted at a 2-ecc C' consists of all 2-eccs that are descendants of C', and the subgraph of G consisting of all those 2-eccs and the bridge edges between them is said to *correspond* to that subtree. For any child C' of the root C of the 2-ecc tree, we call the subgraph of G that corresponds to the subtree rooted at C' a *child subgraph*.

Lemma 6 *Let G be an arbitrary connected graph on n vertices. Then, there is a 2-ecc C^* of G such that rooting the 2-ecc tree of G at C^* ensures that the child subgraphs (i.e. the subgraphs of G corresponding to the subtrees rooted at children of C^*) each contain at most $n/2$ vertices.*

Proof Consider the 2-ecc tree T in which each 2-edge-connected component C of G is represented by a vertex v_C. Root T at an arbitrary node $v_{\hat{C}}$, then process the vertices in a bottom up manner, labelling a vertex v_C with the integer $x_C = |\{u \in V(G) : u \in V(C')$ for a descendant C' of C in $T\}|$. Select a vertex v_{C^*} such that $x_{C^*} \geq n/2$ and such that v_{C^*} has the largest depth in T amongst all such vertices. If v_{C^*} is already the root of T, we are done. Otherwise, let $v_{C'}$ be the parent of v_{C^*} and reroot T at v_{C^*} to form a 2-ecc tree T^*, in which $v_{C'}$ is a child of v_{C^*}. Note that we have $x_C < n/2$ for every child $v_C \neq v_{C'}$ of v_{C^*} in T^*, because otherwise the algorithm would have picked v_C rather than v_{C^*}. Furthermore, we have $x_{C^*} \geq n/2$, and so the total number of vertices in all components C'' such that $v_{C''}$ is a descendant of $v_{C'}$ in T^* must be $\leq n/2$. □

Theorem 6 *Let $\mathcal{G} = \langle G_1, \ldots, G_L \rangle$ be an always-connected, 1-edge-deficient temporal graph with arbitrary underlying graph G and lifetime $L \geq 51n$, and let $|V(G)| = n$. Then, for any start vertex $s \in V(\mathcal{G})$, there is an exploration schedule W of $\mathcal{G}$ that starts at s and has arrival time $\alpha(W) \leq 51n$. Moreover, such a schedule can be computed in polynomial time.*

Proof Apply Lemma 4 to G in order to obtain a spanning subgraph $G^* \subseteq G$ (with $|E(G^*)| \leq 5n$) such that each 2-ecc C of G induces a 2-ecc C^* in G^* satisfying $|E(C^*)| \leq 4|V(C^*)|$.

Apply Lemma 6 to G^* to obtain a 2-ecc tree T of G^* with a root component C_1 such that the child subgraphs $G_i \subseteq G^*$ satisfy $|V(G_i)| \leq n/2$. Let k denote the number of 2-eccs in G^*. Let $T(n,k)$ denote the maximum number of time steps required to explore an arbitrary 1-edge-deficient, always-connected temporal graph with n vertices whose underlying graph has k 2-eccs, at most $5n$ edges, and is such that every 2-ecc C^* satisfies $|E(C^*)| \leq 4|V(C^*)|$, starting from an arbitrary vertex s of the graph at time step 1. We now specify our exploration algorithm and prove by induction on k that it produces an exploration schedule with arrival time at most $T(n,k) \leq 51n$ steps.

Base case (Arbitrary n, $k = 1$): In this case, G^* consists of a single 2-ecc C_1; without loss of generality, assume that $|V(C_1)| \geq 3$. Apply Theorem 5 to C_1, obtaining a circuit cover $\{X_1, \ldots, X_c\}$ of C_1 containing c circuits, where $1 \leq c \leq 3$. Consider now the following three time intervals, noting that $|E(X_i)| \leq |E(C_1)| \leq 5n$ for all $i \in [c]$: $I_1 = [n+1, 6n]$, $I_2 = [7n+1, 12n]$ and $I_3 = [13n+1, 18n]$. During the steps of I_i apply Theorem 4 to X_i to determine a vertex $v_i \in X_i$ such that an exploration schedule of X_i using at most $|E(X_i)| - 1 \leq 5n - 1$ time steps begins at v_i at the first step of I_i. Beginning at the start vertex $s \in V(G)$ in time step 1, employ Lemma 1 to move in at most n steps to vertex v_1, wait until the first step of interval I_1, then follow the walk obtained by the application of Theorem 4 during interval I_1. If $c > 1$, repeat these steps for all remaining circuits X_i in the computed circuit cover of C_1. Once Theorem 4 has been applied to X_c, notice that, for all $i \in [c]$, all vertices of X_i have been visited. Since $\{X_1, \ldots, X_c\}$ covers all edges of C_1 (and also all edges of G^* since G^* consists only of C_1), it follows that all vertices of G^* have been visited at least once. The number of time steps taken to achieve this is at most $c(n+5n) \leq 18n$.

Inductive step (Arbitrary n, $k > 1$): Assume that $T(n',k') \leq 51n'$ for all n' and all $k' < k$ and consider the root component C_1 of G^*. We now distinguish two cases:

Case 1: $|C_1| \geq 2$. Apply Theorem 5 to C_1 and obtain a circuit cover $X^* = \{X_1, \ldots, X_c\}$ of C_1 containing c circuits, where $1 \leq c \leq 3$. Let $V' = \{v \in V(C_1) : v \in e \text{ for some bridge } e\}$ be the set of vertices in $V(C_1)$ that are incident with at least one bridge in G^*. Construct a function $\gamma : V' \to X^*$ by arbitrarily mapping each vertex $v \in V'$ to some circuit $X_i \in X^*$ such that $v \in X_i$. Recall that we root the 2-ecc tree T of G^* at C_1. For each child C_i of C_1 in T, we denote by G_i the child subgraph of G^* corresponding to the subtree of T rooted at C_i. Let $\mathsf{Br} = \{e \in E(G^*) : e \text{ is a bridge and } e \cap V' \neq \emptyset\}$, i.e. Br is the set of bridges of G^* that connect C_1 to child subgraphs. For any $v \in V'$, let $\beta(v) = \{G_i : \{v,u\} \in \mathsf{Br} \text{ for some } u \in G_i\}$. In other words, $\beta(v)$ is the set of child subgraphs that are connected to v via a bridge. For $i \in [c]$, let $F_i = \bigcup_{\{v \in V' : \gamma(v) = X_i\}} \beta(v)$. This means that F_i is the set of child subgraphs with the property that the bridge that connects the child subgraph to C_1 is incident with a vertex $v \in V'$ that has been mapped to X_i by γ. See Fig. 1 for an illustration of a circuit X_i and the child subgraphs in F_i that are attached to it via bridges.

For $i \in [c]$, let $G_{X_i} \subseteq G^*$ be the subgraph of G^* induced by the vertices of X_i and of all the child subgraphs in F_i, i.e. by the vertex set $\{v \in V(G) : v \in V(X_i) \text{ or } v \in V(G_j) \text{ for some } G_j \in F_i\}$. Our aim is, for each $i \in [c]$, to move a real agent from its current vertex to some vertex $s_i \in V(X_i)$ (by Lemma 1 the agent uses at most n steps when initially moving to $V(C_1)$ from s and then at most $|V(C_1)|$ steps for each of the remaining $c-1$ circuits), then construct a closed walk in G_{X_i} that will be followed (in opposite directions) by two virtual agents whenever it is possible for them to do so; i.e. whenever they are not blocked on the next edge they need to traverse. We then let the real agent follow the walk of the virtual agent that finishes their walk first. Let t_i be the time step after the real agent first reaches $s_i \in V(X_i)$. Place the virtual agents at s_i at the start of time step t_i and let them do the

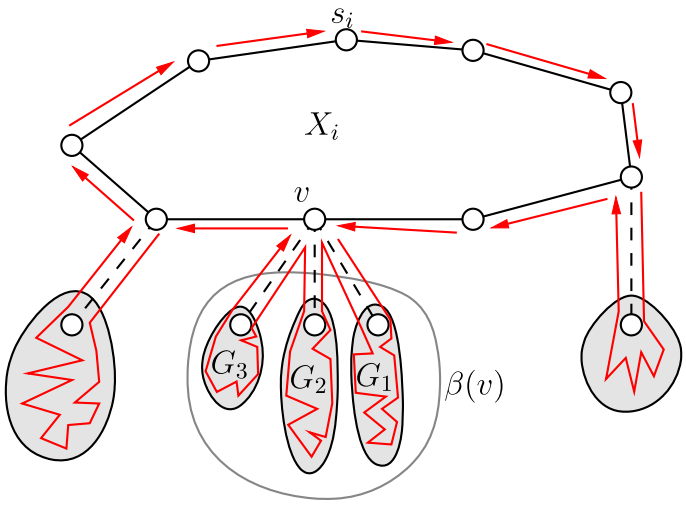

Fig. 1 Illustration of a circuit X_i and the child subgraphs in F_i (indicated as shapes filled in solid grey) that are attached to it via bridges (drawn dashed), as well as the walk followed by agent CW (drawn in red) starting at s_i

following: move along the edges of X_i, one in the clockwise direction (agent CW) and the other in the counterclockwise direction (agent CCW). Whenever an agent $\mathsf{A}^* \in \{\mathsf{CW}, \mathsf{CCW}\}$ reaches for the first time a vertex $v \in V'$ such that $\gamma(v) = X_i$, that agent processes the subgraphs $G_j \in \beta(v)$, in increasing order of indices if $\mathsf{A}^* = \mathsf{CW}$ and in decreasing order otherwise, in the following way (the only exception is with vertex s_i: if $s_i \in V'$ and $\gamma(s_i) = X_i$, then agent CW processes each $G_j \in \beta(s_i)$ immediately at the start of the walk before traversing any edge of X_i, whilst agent CCW processes them only when it returns to s_i after having traversed all edges of X_i): check (1) if the other agent, say $\mathsf{A}^{**} \in \{\mathsf{CW}, \mathsf{CCW}\} - \mathsf{A}^*$, currently occupies some vertex in G_j; if not, then check (2) if A^{**} is currently positioned at vertex v and needs to process next the same subgraph G_j as agent A^*.

If neither (1) or (2) hold, then agent A^* traverses the bridge connecting G_j and $v \in V(X_i)$, follows a DFS in G_j (i.e. traverses edges of G_j in the order of a static DFS in G_j, and waits at the current vertex whenever the next edge to be traversed is not available), then re-traverses the connecting bridge towards X_i and continues processing the remaining $G_{j'} \in \beta(v)$ with $j' \neq j$ in the same way. If (1) holds, then agent A^* should ignore G_j and continue processing the other $G_{j'} \in \beta(v)$. If (2) holds, then arbitrarily select one of the agents to process G_j in the same way as when neither (1) nor (2) hold, and let the other agent ignore G_j and continue processing the remaining $G_{j'} \in \beta(v)$ with $j' \neq j$. See Fig. 1 for an illustration of the walk followed by agent CW if it is never prevented from entering a child subgraph due to an occurrence of (1) or (2).

Note that neither agent should traverse the next edge of X_i until all $G_j \in \beta(v)$ have been processed in the above way. Since the two agents process the subgraphs in opposite orders, it is ensured that at most a single subgraph $G_j \in F_i$ is ignored by an agent during the entire execution of the virtual agent procedure on G_{X_i}. This is because, after either case (1) or (2) occurs for the first time, the subgraphs $G_j \in F_i$ that remain to be explored by A^* are precisely those that have already been explored by A^{**}, and vice versa. Also note that if either (1) or (2) occurs, then at no point during the remainder of the procedure's execution on G_{X_i} will the agents become blocked on the same edge.

Both agents continue in this way until the first time step in which both agents are blocked on the same edge e (if this happens at all). Note that we must have $e \in E(X_i)$, since the way the agents process each subgraph G_j ensures that they will never be positioned in any subgraph $G_j \in F_i$ during the same time step.

Now, if every edge of G^* were to be present in every time step, it would take each agent at most $\mathsf{Exp}(X_i) = |E(X_i)| + \sum_{G_j \in F_i} 2|V(G_j)|$ steps to carry out their respective walk in G_{X_i}: 1 step to traverse each of the edges of X_i, $2|V(G_j)| - 2$ steps spent exploring G_j via a DFS, and 2 steps spent traversing each of the bridge edges connecting X_i and a $G_j \in F_i$. Since G^* is 1-edge-deficient, it is possible for the agents to both be blocked on the same edge during the same time step. We distinguish two subcases, first recalling that t_i denotes the time step in which the exploration of G_{X_i} begins. We use t_i' to denote an upper bound on the time step by which the exploration of G_{X_i} (possibly except one subgraph G_{j^*}) is completed by at least one of the two agents.

Case 1.1: If the agents are never blocked on the same edge e during any step t in $[t_i, t_i']$ for $t_i' = t_i + 2\mathsf{Exp}(X_i)$, then, in each time step $t \in [t_i, t_i']$, at least one of the two agents is able to cross the next edge of their respective walk. In this case, we have that by the end of time step t_i', the agent A^* that was blocked on an edge in the least number of time steps $t \in [t_i, t_i']$ will have not been blocked in $\geq \mathsf{Exp}(X_i)$ time steps and, as such, will have arrived again at $s_i \in V(X_i)$. At this point, A^* will have finished an exploration of at least the subgraph of G^* induced by $V(G_{X_i}) - V(G_j)$ in at most $2\mathsf{Exp}(X_i)$ steps (for some $G_j \in F_i$ which was possibly ignored by A^* when it was about to process G_j because the other agent occupied some vertex in $V(G_j)$).

Case 1.2: The agents are both blocked on the same edge $e \in X_i$ during a time step $t \in [t_i, t_i + 2\mathsf{Exp}(X_i)]$. Let $t_i' = t_i + 2\mathsf{Exp}(X_i) + |E(X_i)|$. Check whether or not e is present during any step $t' \in [t+1, t+|E(X_i)|]$. If yes, wait until that step, then let both agents cross e. If not, let both agents apply Lemma 1 in X_i, using at most $|E(X_i)| - 1$ time steps to move to the opposite endpoint of e, then continue attempting to traverse the next edge of their walk whenever possible. Notice that, during any step $t' \in [t_i, t-1]$, at least one of the two agents was able to cross the next edge in their respective walk, since t is the first time step in which both agents are blocked on the same edge. When the agents are blocked on e during step t, they either wait at their current vertex for at most $|E(X_i)| - 1$ steps until e is present again, or spend $\leq |E(X_i)| - 1$ steps reaching the opposite endpoint of e by applying Lemma 1 in X_i. In either case, it takes at most $|E(X_i)| - 1$ steps for them to reach the opposite endpoint of e. At this point, observe that the vertices $x \in V(X_i)$ and the $G_j \in F_i$ that remain to be explored/processed by agent CW are exactly those that have already been explored/processed by agent CCW (and vice versa). Hence, it follows that the two agents will not be blocked on the same edge again for the remainder of their respective walks. In all the remaining steps, since the sets of vertices unexplored by the walks of the two agents are disjoint, we again have that at least one of the two agents will be able to cross the next edge of their respective walk in all steps $t' \in [t + |E(X_i)|, t_i']$. Concluding, during the entire time interval $[t_i, t_i']$, there are $\leq |E(X_i)|$ steps in which neither of the agents can cross the next edge of their respective walk, and by step $t_i' = t_i + 2\mathsf{Exp}(X_i) + |E(X_i)| \leq t_i + 2\mathsf{Exp}(X_i) + 4|V(C_1)|$ (where the inequality holds since $|E(X_i)| \leq |E(C_1)| \leq 4|V(C_1)|$ by our application of Lemma 4), it is ensured that the agent who was blocked during the least number of steps since the start of step t_i has completed their exploration of G_{X_i} in at most $2\mathsf{Exp}(X_i) + 4|V(C_1)|$ steps. Note that this agent has not skipped any of the child subgraphs in F_i, since the two agents were blocked on the same edge of X_i and thus it could never happen that one agent skipped a child

subgraph because the other was already in that subgraph or wanted to enter it in the same step.

After processing all c circuits X_i in this way, there will be at most c subgraphs that have not yet been explored. We next reduce those unexplored subgraphs to at most one: whilst there are two or more unexplored subgraphs, we repeatedly (1) choose a circuit X in C_1 with $|E(X)| \leq 2|V(C_1)|$ that contains two vertices of V' that have an incident bridge leading to an unexplored subgraph (this is possible by Lemma 5), and then (2) move to a vertex in X in at most $|V(C_1)|$ steps (by Lemma 1) and process X and 2 of the at most three unexplored subgraphs in the same way that we processed X_i for $i \in [c]$ (i.e. via the virtual agent procedure). Since we have $|E(X)| \leq 2|V(C_1)|$ and there are two subgraphs G_j and $G_{j'}$ (both satisfying $|V(G_j)|, |V(G_{j'})| \leq n/2$), we have:

$$\begin{aligned} \mathsf{Exp}(X) &= |E(X)| + 2|V(G_j)| + 2|V(G_{j'})| && (2)\\ &\leq 2(|V(C_1)| + |V(G_j)| + |V(G_{j'})|) && (3)\\ &\leq 2n && (4) \end{aligned}$$

Therefore, by the same reasoning that was used in **Case 1.1** and **Case 1.2**, the virtual agent that completes their walk in the subgraph induced by $V(C_1) \cup V(G_j) \cup V(G_{j'})$ first will have either explored at least one of G_j and $G_{j'}$ in $\leq 2\mathsf{Exp}(X)$ steps, or explored both G_j and $G_{j'}$ in $\leq 2\mathsf{Exp}(X) + |E(X)| \leq 2\mathsf{Exp}(X) + 2|V(C_1)|$ steps. Note that, since there are at most three subgraphs that remain unexplored by the real agent after applying the virtual agent procedure in all G_{X_i}, we can be required to perform the above subgraph reduction procedure at most twice. After doing so, the real agent will have visited all vertices in $\bigcup_{i\in[c]}[V(G_{X_i})]$, possibly apart from those that belong to a single subgraph $G' \in F_i$ (for exactly one $i \in [c]$). If such a subgraph G' exists then we move the real agent (in $|V(C_1)| - 1$ steps using Lemma 1) to the unique vertex v in C_1 such that $G' \in \beta(v)$, then use an additional time step to traverse the bridge $\{v, v'\}$ (recall that bridges are always present since the temporal graph is always-connected); this requires $|V(C_1)|$ steps in total. We then apply the algorithm recursively to G', starting at vertex v'.

In order to now bound the overall worst-case exploration time of the algorithm, we assume that there does exist a subgraph G' that requires a recursive application of the algorithm, and determine the worst-case number of time steps required by the real agent to explore $\bigcup_{i\in[c]}[V(G_{X_i})] - V(G') = V(G^*) - V(G')$, then reach an arbitrary vertex in $V(G')$. We can assume w.l.o.g. that our computed circuit cover $\{X_1, \ldots, X_c\}$ has size $c = 3$ – it is clear that this is the worst-case number of circuits we need to process. Let $r = |V(C_1)|$. Recall that in **Case 1.1** we have an upper bound of $2\mathsf{Exp}(X_i)$ on the amount of time required to explore all of $V(G_{X_i}) - V(G'')$ for a single subgraph $G'' \in F_i$ that was possibly missed by the virtual agent that finished their walk in G_{X_i} first; in **Case 1.2** we obtained a $2\mathsf{Exp}(X_i) + 4r$ upper bound on the amount of time required to explore all of G_{X_i}. Since $\mathsf{Exp}(X_i) = |E(X_i)| + \sum_{G_j\in F_i} 2|V(G_j)| \leq 4r + 2\sum_{G_j\in F_i} |V(G_j)|$ (where the inequality follows from our application of Lemma 4), we have:

$$\sum_{i\in[3]} 2\mathsf{Exp}(X_i) \leq 24r + 4\sum_{i\in[3]}\sum_{G_j\in F_i} |V(G_j)| \leq 24r + 4(n-r) = 20r + 4n$$

and so in the worst case at least $20r + 4n$ steps are spent exploring all X_i (possibly missing a single subgraph $G'' \in F_i$ for each $i \in [3]$) and an additional $4r$ steps may also be needed whilst processing each X_i if the virtual agents get blocked on the same edge $e \in E(X_i)$ during the same time step. Similarly, in the worst case, we require up to $2\mathsf{Exp}(X)$ steps whilst processing each of up to two additional circuits X (with two child subgraphs attached to each

X) using the virtual agent procedure. By Eqs. (2)–(4) we have $2\mathsf{Exp}(X) \leq 4n$, so in the worst case at least $4n$ steps are spent processing each additional circuit, and an additional $2r$ steps may be needed whilst processing each circuit if the virtual agents get blocked on the same edge of a circuit X.

We now introduce four parameters to be used in order to distinguish the cases that can arise during an execution of the algorithm: let $x \in \{0, \ldots, 3\}$ be the number of times we miss a subgraph over all applications of the virtual agent procedure to the circuits in $\{X_1, \ldots, X_3\}$. Let $y \in \{0, \ldots, 2\}$ be the number of times we need to compute an additional circuit X and carry out the virtual agent procedure on the subgraph induced by X and two unexplored subgraphs. Let $y' \in \{0, \ldots, y\}$ be the number of times we do not miss a subgraph across all y applications of the virtual agent procedure on additionally constructed circuits X (note that we have $y' \leq 1$ because as soon as we explore both unexplored subgraphs attached to some circuit X we either (1) have no unexplored subgraphs remaining and exploration is complete; or (2) have 1 remaining subgraph and we need to recursively apply the algorithm). Let $z \in \{0, 1\}$ be the number of times we recursively apply the algorithm to a final unexplored subgraph.

The worst-case number of time steps required to explore all but at most a single subgraph G' (and then be positioned at some vertex in G') for each possible combination of x, y, y', z can then be upper bounded by the following sum:

$$\begin{aligned} f(x, y, y', z) = \; & (20r + 4n) \\ & + (3 - x)4r \\ & + y \cdot 4n \\ & + y' \cdot 2r \\ & + (n + (2 + y + z)r); \end{aligned}$$

where $20r + 4n$ steps (and possibly more) are needed to process all circuits X_i; $(3-x)4r$ is an upper bound on the number of additional steps needed to process circuits X_i (incurred when the virtual agents become blocked on the same edge of some X_i); $y \cdot 4n$ steps (and possibly more) are needed to process $y \leq 2$ additional circuits X with two unexplored subgraphs attached to each; $y' \cdot 2r$ is an upper bound on the amount of additional steps needed to process y additional circuits X (incurred when the virtual agents get blocked on the same edge of some X); and $n + (2 + y + z)r$ is an upper bound on the number of steps spent using the reachability lemma to: (1) reach, in at most n steps, an arbitrary vertex in X_1 from start vertex s; (2) reach, in at most r steps, an arbitrary vertex in each X_i with $i \in [2, 3]$ from some vertex in $V(C_1)$ at which the last application of the virtual agent procedure finished; (3) reach, in at most r steps, each of y additional circuits as part of the unexplored subgraph reduction procedure; and (3) reach, in at most r steps, the final unexplored subgraph should one exist.

Next, we enumerate all combinations of values for x, y, y' and z attainable during a run of the exploration algorithm, before listing the worst-case exploration time for each case. First note that if $x = 0$, then no applications of the subgraph reduction procedure or any recursive call are required and so we must have $y = y' = z = 0$ (**Case A**). If $x = 1$, then no application of the subgraph reduction procedure is required, so we must have $y = y' = 0$, and we need $z = 1$ recursive calls to explore the $x = 1$ missed subgraph (**Case B**). If $x = 2$, then we require $y = 1$ application of the subgraph reduction procedure; after this application, we may either explore both subgraphs and have $y' = 1$ so that $z = 0$ recursive calls are required (**Case C**), or have $y' = 0$ and require exactly $z = 1$ recursive calls (**Case D**). Note that if $x = 3$, then we require at least 1 application (and at most two applications) of the subgraph

reduction procedure. If we have $x = 3$ and $y = 1$, then we must have $y' = 1$ since our single application of the subgraph reduction procedure reduced the number of unexplored subgraphs from 3 to 1, and we require exactly $z = 1$ recursive calls (**Case E**). If $x = 3$ and $y = 2$, then a subgraph must have been missed during the first application of the subgraph reduction procedure, so we can either have: $y' = 1$ (so that both remaining unexplored subgraphs are explored during the second application) and require $z = 0$ recursive calls (**Case F**); or $y' = 0$, so that we miss a subgraph during both applications of the subgraph reduction procedure and require $z = 1$ recursive calls to explore the final remaining subgraph (**Case G**).

Note that in any of the identified cases, if $z = 0$, then no recursive call is required and so the worst-case exploration time achievable in that case is upper bounded by the maximum corresponding value of $f(x, y, y', z)$ over all possible values of r. If $z = 1$, then there is one remaining subgraph G' that contains $k' < k$ 2-eccs (since G' surely does not contain C_1) and satisfies $|V(G')| \leq \min\{n/2, n - r\}$ (by our application of Lemma 6). Then, by the induction hypothesis, the time required to explore G' is upper bounded by $T(|V(G')|, k') \leq T(\min\{n/2, n - r\}, k') \leq 51 \min\{n/2, n - r\}$. As such, the worst-case exploration time achieved in any case in which $z = 1$ is upper bounded by the sum of the maximum corresponding value of $f(x, y, y', z)$ plus $51 \min\{n/2, n - r\}$, over all possible values of r. The list below gives the worst-case bounds on the arrival time $\alpha(W)$ of the exploration schedule W obtained by our exploration algorithm in Cases A–G; we distinguish two subcases—namely $r \leq n/2$ and $r > n/2$—whenever $z = 1$:

- **Case A** $x = 0, y = 0, y' = 0, z = 0$: $\alpha(W) \leq f(0, 0, 0, 0) = 34r + 5n \leq 39n$
- **Case B** $x = 1, y = 0, y' = 0, z = 1$: $f(1, 0, 0, 1) = 31r + 5n$

  $r \leq n/2$: $\alpha(W) \leq 15.5n + 5n + 51(n/2) = 46n$
  $r > n/2$: $\alpha(W) \leq 31r + 5n + 51(n - r) = 56n - 20r < 46n$

- **Case C** $x = 2, y = 1, y' = 1, z = 0$: $\alpha(W) \leq f(2, 1, 1, 0) = 29r + 9n \leq 38n$
- **Case D** $x = 2, y = 1, y' = 0, z = 1$: $f(2, 1, 0, 1) = 28r + 9n$

  $r \leq n/2$: $\alpha(W) \leq 14n + 9n + 51(n/2) = 48.5n$
  $r > n/2$: $\alpha(W) \leq 28r + 9n + 51(n - r) = 60n - 23r < 48.5n$

- **Case E** $x = 3, y = 1, y' = 1, z = 1$: $f(3, 1, 1, 1) = 26r + 9n$

  $r \leq n/2$: $\alpha(W) \leq 13n + 9n + 51(n/2) = 47.5n$
  $r > n/2$: $\alpha(W) \leq 26r + 9n + 51(n - r) = 60n - 25r < 47.5n$

- **Case F** $x = 3, y = 2, y' = 1, z = 0$: $\alpha(W) \leq f(3, 2, 1, 0) = 26r + 13n \leq 39n$
- **Case G** $x = 3, y = 2, y' = 0, z = 1$: $f(3, 2, 0, 1) = 25r + 13n$

  $r \leq n/2$: $\alpha(W) \leq 12.5n + 13n + 51(n/2) = 51n$
  $r > n/2$: $\alpha(W) \leq 25r + 13n + 51(n - r) = 64n - 26r < 51n$

Clearly then, the overall worst-case time required of the described algorithm to visit all $v \in V(G^*)$ at least once (and hence complete an exploration of $\mathcal{G}$) is upper bounded by the maximum value appearing in the above list, i.e. by $51n$ which occurs in **Case G**. Hence, we have $T(n, k) \leq 51n$, as required.

Case 2: $|C_1| = 1$. Let v be the single vertex contained in $V(C_1)$. Move the real agent to v in at most n steps using Lemma 1; once arrived at v, we know C_1 has been fully explored. Consider the set $\beta(v)$ of the child subgraphs that are connected to v via bridges. Again we use a virtual agent approach: the agents CW and CCW process the $G_j \in \beta(v)$ in increasing and decreasing order of their indices, respectively. Each time they begin processing a new G_j, they (1) check to see if the other agent is situated at a vertex of G_j. If so, they ignore G_j

and continue processing the other $G_{j'} \in \beta(v)$ with $j' \neq j$. Then, if (1) does not hold, they check (2) if the other agent is currently positioned at v and also needs to process G_j next.

If neither of (1) or (2) hold, then the agent moves over the bridge edge connecting v and $G_j \in \beta(v)$, follows a DFS in G_j and then returns to v via the same bridge edge before processing the next G_j. If (1) holds, then the agent ignores subgraph G_j and the other agent continues their DFS in G_j; if (2) holds, we arbitrarily nominate one of the two agents to explore G_j via a DFS and the other agent ignores G_j. Note that in this way, the two agents will never both be blocked on the same edge in some time step t—they cannot be blocked on a bridge edge incident to v (since $\mathcal{G}$ is always-connected and so bridges in $E(G^*)$ are always present), and they cannot be blocked on an edge $e \in E(G_j)$ for any j, since the exploration procedure ensures they are never positioned in G_j at the same time.

Since traversing a bridge edge towards a subgraph G_j, carrying out a DFS in G_j and then re-traversing the bridge towards v again takes $2|V(G_j)|$ steps in total, it would require of an agent completing a static exploration of G^* exactly $\sum_{G_j \in \beta(v)} 2|V(G_j)|$ steps to process all G_j in this way. Since $\mathcal{G}$ is 1-edge-deficient, it is possible for an individual agent to be blocked on an edge during some time step, but not both agents on the same edge in the same time step. Hence, in every time step at least one of the two agents will be able to cross the next edge of their respective walk, and so in the worst case the agent that first finishes processing all subgraphs in the described manner does so in at most $2 \cdot \sum_{G_j \in \beta(v)} 2|V(G_j)| \leq 4|V(G^*)| = 4n$ steps, having missed a single unexplored subgraph $G' \in \beta(v)$ which has $k' < k$ 2-eccs (since G' surely does not contain C_1) and satisfies $|V(G')| \leq n/2$ (by our application of Lemma 6). In this situation, we move the real agent to an arbitrary vertex in G' in 1 step (since G' is connected to v by a bridge, and since bridges must be present in every layer of an always-connected temporal graph) and recursively apply the exploration algorithm to G'. By the inductive hypothesis, the amount of time required to explore G' is at most $T(|V(G')|, k-1)) \leq T(n/2, k-1) \leq 51(n/2) = 25.5n$. Therefore, since by Lemma 1 we require $\leq n$ steps to initially reach vertex $v \in V(C_1)$ from the start vertex s of the exploration, which can be anywhere in the graph, then $\leq 4n$ steps to follow the walk of the best agent, 1 step to move from v to a vertex in G' over a bridge, and finally another $\leq 25.5n$ steps to explore all of G', it follows that the entire exploration of $\mathcal{G}$ takes $\leq 5n + 1 + 25.5n = 30.5n + 1$ steps in total, which is less than $51n$ for all $n \geq 1$, as required.

Finally, we remark that all steps in the construction of the exploration schedule can be implemented to run in polynomial time, and thus we have a polynomial-time algorithm for computing such schedules. □

Note that, in Theorem 6, it was sufficient to assume a lifetime of at least $51n$ since our exploration method ensures that the exploration is completed after at most $51n$ steps. Furthermore, our upper bound of Theorem 6 together with the obvious lower bound of $n-1$ on the arrival time of any exploration schedule implies the following corollary.

Corollary 2 *There exists a polynomial-time $O(1)$-approximation algorithm for* F- TEXP *on always-connected* 1*-edge-deficient temporal graphs.*

5 Lower bound

To complement the upper bounds from Sects. 3 and 4, we also present a lower bound on the worst-case exploration time of k-edge-deficient temporal graphs. For this, we adapt the construction of a lower bound of $\Omega(n \log n)$ on the exploration time of temporal graphs with underlying planar graphs of maximum degree 4 from [6, Theorem 4.1]. That construction

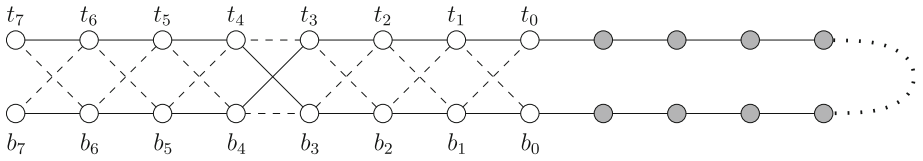

Fig. 2 Underlying graph of the temporal graph constructed in the proof of Theorem 7 for $k = 14$. Edges present from time $n/2+1$ to n are drawn solid (with the dotted line representing a longer path of solid edges), the remaining edges are drawn dashed

has a time-varying part (in which $n/2$ edges are missing in each time step) and a fixed part (a static path of $n/2$ edges). By reducing the size of the time-varying part and increasing the size of the static part, we can show the following theorem.

Theorem 7 *For arbitrarily large n and every k with $2 \leq k \leq \frac{n}{2} - 2$, there is an always-connected k-edge-deficient temporal graph with n vertices and lifetime $(n-1)^2$ for which an optimal exploration takes $\Omega(n \log k)$ steps.*

Proof Assume without loss of generality that $k+2$ is a power of two (otherwise, consider the next smaller value of k that satisfies this condition) and that n is even. Consider the following underlying graph $G = (V, E)$ (see Fig. 2): the vertex set V contains $k+2$ vertices $V_0 = \{t_i, b_i \mid 0 \leq i \leq k/2\}$, edges $\{t_i, t_{i+1}\}$, $\{b_i, b_{i+1}\}$, $\{t_i, b_{i+1}\}$ and $\{b_i, t_{i+1}\}$ for $0 \leq i < k/2$, and a path P with $n-(k+2)$ new internal vertices (drawn solid in Fig. 2) that connects t_0 and b_0.

Let $h = 1+k/2$. For $1 \leq i < h$, we refer to the edges $\{t_{i-1}, t_i\}$ and $\{b_{i-1}, b_i\}$ as *horizontal* edges of column i, and to the edges $\{t_{i-1}, b_i\}$ and $\{b_{i-1}, t_i\}$ as *cross* edges of column i.

We describe the first $\frac{n}{2} \log_2 h = \Theta(n \log k)$ steps of a temporal realization of G and show that it cannot be explored within these steps. For the remaining time steps of the temporal realization of G, we let each layer be equal to the layer in step $\frac{n}{2} \log_2 h$.

Consider the following construction of the first $\frac{n}{2} \log_2 h$ time steps of a temporal realization of G, consisting of $\log_2 h$ *phases*, each of which is $n/2$ time steps long and consists of $n/2$ identical layers: the path P is always present. In the first phase of $n/2$ time steps, the graph additionally contains the horizontal edges of all columns. At the start of the next phase of $n/2$ time steps, the horizontal edges of column $h/2$ are replaced by the cross edges (this is the state shown in Fig. 2), and this replacement is permanent until the end of the construction. At the start of the next phase of $n/2$ time steps, the horizontal edges of columns $h/4$ and $3h/4$ are replaced (permanently) by the cross edges. This continues for a total of $\log_2 h$ phases as follows: at the start of each phase, the horizontal edges of the middle column in each stretch of consecutive horizontal edges are permanently replaced by the cross edges. This implies that, throughout the steps of the last of the $\log_2 h$ phases, all cross edges but none of the horizontal edges are present.

Let $V_0' = V_0 - \{t_0, b_0\}$. In each phase, we say that a vertex of V_0' is *on the t_0-side* if it is in the same connected component as t_0 in the subgraph induced by V_0 in the layers of that phase, and *on the b_0-side* otherwise. Observe that in each of the phases, any algorithm can explore either vertices in V_0' on the t_0-side or vertices in V_0' on the b_0-side, but not both (because the path P has $n-(k+2)+1 \geq \frac{n}{2}+1$ edges). If the algorithm visits any vertex of V_0' in a phase and if that vertex is on the t_0-side, we assume that the algorithm visits all vertices of V_0' on the t_0-side in that phase, and similarly if it visits a vertex of V_0' on the b_0-side. This assumption can only decrease the exploration time. No matter which of the two sets of vertices the algorithm visits in a phase, in the next phase half of the unvisited vertices

in V_0' will be on the t_0-side and half on the b_0-side. Therefore, each phase halves the number of unvisited vertices. At the end of phase $\log_2 h$, the number of unvisited vertices in V_0' is at least $2k/2^{\log_2 h} = 2k/(k + \frac{1}{2}) > 1$. Hence, the exploration time is $\Omega(n \log k)$, independent of the start vertex.

Furthermore, it is easy to see that the constructed temporal graph is k-edge-deficient: in each column i, $1 \leq i < h$, two edges from the underlying graph are missing in each step, so the total number of missing edges in each step is $2(h-1) = k$. □

6 Conclusion

We have shown that always-connected k-edge-deficient temporal graphs admit an exploration schedule with arrival time $O(kn \log n)$; if $k = 1$, the arrival time improves to $O(n)$. The proofs are constructive, yielding polynomial-time algorithms for computing such exploration schedules. As $n - 1$ steps are necessary to explore any graph, our results also yield $O(k \log n)$ and $O(1)$-approximation algorithms for F- TEXP for the $k \in \mathbb{N}$ and $k = 1$ cases, respectively, as well as an $O(\log n)$-approximation if $k = O(1)$. Furthermore, we have given an infinite family of k-edge-deficient temporal graphs that require $\Omega(n \log k)$ time steps to be explored. It would be interesting to close the gap between the lower and upper bounds. In particular, an interesting question is whether always-connected k-edge-deficient graphs for $k = O(1)$ can be explored in $O(n)$ steps. For $k = 1$, there is the question of whether (and by how much) the constant factor in our upper bound of $51n$ on the exploration time can be improved. It is known that there are 1-deficient temporal graphs whose underlying graph is a cycle with n nodes that require $2n - 3$ steps for exploration [6], which implies that the constant factor in the upper bound for arbitrary always-connected 1-edge-deficient temporal graphs must be at least 2.

References

1. Akrida, E.C., Mertzios, G.B., Spirakis, P.G.: The temporal explorer who returns to the base. In: P. Heggernes (ed.) 11th International Conference on Algorithms and Complexity (CIAC 2019), Lecture Notes in Computer Science, vol. 11485, pp. 13–24. Springer (2019). https://doi.org/10.1007/978-3-030-17402-6_2
2. Bodlaender, H.L., van der Zanden, T.C.: On exploring always-connected temporal graphs of small pathwidth. Inf. Process. Lett. **142**, 68–71 (2019). https://doi.org/10.1016/j.ipl.2018.10.016
3. Brodén, B., Hammar, M., Nilsson, B.J.: Online and offline algorithms for the time-dependent TSP with time zones. Algorithmica **39**(4), 299–319 (2004). https://doi.org/10.1007/s00453-004-1088-z
4. Bui-Xuan, B., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. Int. J. Found. Comput. Sci. **14**(2), 267–285 (2003). https://doi.org/10.1142/S0129054103001728
5. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. Int. J. Parallel Emergent Distrib. Syst. **27**(5), 387–408 (2012). https://doi.org/10.1080/17445760.2012.668546

6. Erlebach, T., Hoffmann, M., Kammer, F.: On temporal graph exploration. J. Comput. Syst. Sci. **119**, 1–18 (2021). https://doi.org/10.1016/j.jcss.2021.01.005
7. Erlebach, T., Kammer, F., Luo, K., Sajenko, A., Spooner, J.T.: Two Moves per Time Step Make a Difference. In: C. Baier, I. Chatzigiannakis, P. Flocchini, S. Leonardi (eds.) 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, pp. 141:1–141:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.141
8. Erlebach, T., Spooner, J.T.: Faster Exploration of Degree-Bounded Temporal Graphs. In: I. Potapov, P. Spirakis, J. Worrell (eds.) 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), Leibniz International Proceedings in Informatics (LIPIcs), vol. 117, pp. 36:1–36:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.MFCS.2018.36
9. Erlebach, T., Spooner, J.T.: Non-strict Temporal Exploration. In: A.W. Richa, C. Scheideler (eds.) 27th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2020), Lecture Notes in Computer Science, vol. 12156, pp. 129–145. Springer (2020). https://doi.org/10.1007/978-3-030-54921-3_8
10. Erlebach, T., Spooner, J.T.: Exploration of k-edge-deficient temporal graphs. In: A. Lubiw, M.R. Salavatipour (eds.) 17th International Symposium on Algorithms and Data Structures (WADS 2021), Lecture Notes in Computer Science, vol. 12808, pp. 371–384. Springer (2021). https://doi.org/10.1007/978-3-030-83508-8_27
11. Fan, G.: Covering graphs by cycles. SIAM J. Discret. Math. **5**(4), 491–496 (1992). https://doi.org/10.1137/0405039
12. Frederickson, G.N., Johnson, D.B.: Finding k-th paths and p-centers by generating and searching good data structures. J. Algorithms **4**(1), 61–80 (1983). https://doi.org/10.1016/0196-6774(83)90035-4
13. Gallai, T.: Elementare Relationen bezüglich der Glieder und trennenden Punkte von Graphen. Magyar Tud. Akad. Mat. Kutato Int. Kozl **9**, 235–236 (1964)
14. Gotoh, T., Flocchini, P., Masuzawa, T., Santoro, N.: Tight Bounds on Distributed Exploration of Temporal Graphs. In: Felber, P., Friedman, R., Gilbert, S., Miller, A. (eds.) 23rd International Conference on Principles of Distributed Systems (OPODIS 2019), Leibniz International Proceedings in Informatics (LIPIcs), vol. 153, pp. 22:1–22:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.OPODIS.2019.22
15. Gotoh, T., Sudo, Y., Ooshita, F., Masuzawa, T.: Dynamic ring exploration with (H, S) view. Algorithms (2020). https://doi.org/10.3390/a13060141
16. Harary, F., Prins, G.: The block-cutpoint-tree of a graph. Publ. Math. Debrecen **13**(103–107), 19 (1966)
17. Ilcinkas, D., Klasing, R., Wade, A.M.: Exploration of Constantly Connected Dynamic Graphs Based on Cactuses. In: Halldórsson, M.M. (ed.) 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO 2014), Lecture Notes in Computer Science, vol. 8576, pp. 250–262. Springer (2014). https://doi.org/10.1007/978-3-319-09620-9_20
18. Ilcinkas, D., Wade, A.M.: Exploration of the T-interval-connected dynamic graphs: The case of the ring. In: Moscibroda, T., Rescigno, A.A. (eds.) 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2013), Lecture Notes in Computer Science, vol. 8179, pp. 13–23. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03578-9_2
19. Kempe, D., Kleinberg, J.M., Kumar, A.: Connectivity and inference problems for temporal networks. J. Comput. Syst. Sci. **64**(4), 820–842 (2002). https://doi.org/10.1006/jcss.2002.1829
20. Michail, O.: An introduction to temporal graphs: An algorithmic perspective. In: Zaroliagis, C., Pantziou, G., Kontogiannis, S. (eds.) Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday, pp. 308–343. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-24024-4_18
21. Michail, O., Spirakis, P.G.: Traveling salesman problems in temporal graphs. Theor. Comput. Sci. **634**, 1–23 (2016). https://doi.org/10.1016/j.tcs.2016.04.006
22. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7**(5&6), 583–596 (1992). https://doi.org/10.1007/BF01758778
23. Shannon, C.E.: Presentation of a maze-solving machine. In: Sloane, N.J.A., Wyner A.D. (eds.) Claude Elwood Shannon – Collected Papers, pp. 681–687. IEEE Press (1993)
24. Zschoche, P., Fluschnik, T., Molter, H., Niedermeier, R.: The complexity of finding small separators in temporal graphs. J. Comput. Syst. Sci. **107**, 72–92 (2020). https://doi.org/10.1016/j.jcss.2019.07.006