Online Space-Time Travel Planning in Dynamic Graphs

Quentin Bramas
University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Jean-Romain Luttringer
University of Strasbourg, ICUBE, CNRS, Strasbourg, France

Sébastien Tixeuil
Sorbonne University, CNRS, LIP6, Institut Universitaire de France, Paris, France

Abstract

We study the problem of traveling in an unknown dynamic graph, to reach a destination with minimum latency. At each step of the execution, an agent can decide to move to a neighboring node if an edge exists at this time instant, wait at the current node in the hope that other links will appear in the future, or move backward in time using an expensive time travel device. A travel that makes use of backward time travel is called a space-time travel. Our aim is to arrive at the destination with zero delay, which always requires the use of backward time travel if no path exists to the destination during the first time instant.

Finding an optimal space-time travel is polynomial when the agent knows the entire dynamic graph (including the future edges), even with additional constraints. However, we consider in this paper that the agent discovers the dynamic graph while it is exploring it, in an online manner.

In this paper, we propose two models that define how an agent learns new knowledge about the dynamic graph during the execution of its protocol: the T-online model, where the agent reaching time t learns about the entire past of the network until t (even nodes not yet visited), and the S-online model, where the agent learns about the past and future about the current node he is located at. We present an algorithm with an optimal competitive ratio of 2 for the T-online model. In the S-online model, we prove a lower bound of $2/3n - 7/4$ and an upper bound of $2n - 3$ on the optimal competitive ratio when the cost function is linear.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Dynamic graphs, online algorithm, space-time travel, treasure hunt

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.7

1 Introduction

We consider the problem of an agent moving in both space and time in a dynamic graph representing a transportation network. The goal of the agent is to reach a destination node in the aforementioned graph with a delay of zero, thanks to backward time-travels. As the dynamic graph evolves, its edges may appear and disappear over time. The agent can wait at a given node for an adjacent edge to appear (thus moving forward in time). However, conversely to most known models, we consider that the agent can also go back in time, to cross an adjacent edge that previously appeared in the past. However, moving backward in time involves a cost that the agent seeks to minimize.

It has been shown by Bramas et al. [4] that finding optimal-delay optimal-costs travels can be computed with a polynomial offline algorithm, even when assuming an upper constraint on the cost. However, the offline setting considered by Bramas et al. [4] implies that the agent knows the entire dynamic graph. For example, an agent at time t may only be aware of the evolution of the dynamic graph up to time t (e.g., if this dynamic graph represents a transportation network, unforeseen problems may arise in the future, while past availability periods have been tracked). Also, the dynamic graph itself may be infinite, which may cause storage issues before running the offline algorithm. Thus, it is important to consider settings where the agent that plans its space-time travel only has limited knowledge.

In this paper, we focus on *online* settings, where agents possess limited initial knowledge of the underlying dynamic graph. We define two settings, referred to as *T-Online* and *S-Online*. In the T-online setting, the agent does *not* know the future of the dynamic graph, but learns everything about the temporal graph up to its current time instant (even the existence of time-edges between nodes that were not yet visited). Thus, the agent has complete spatial knowledge up to its current time instant, but still navigates *T*ime in an *online* fashion. Conversely, in the S-online settings, the agent knows the entire past and future of the nodes it has visited, but has no knowledge about yet unvisited nodes, and thus navigates *S*pace in an *online* fashion. The knowledge acquired by the agent in both settings is illustrated in Fig. 3 and 4.

Related Work. Space-Time routing has been studied, mostly assuming forward time travel, i.e., waiting, is available. Many studies (see *e.g.* Casteigts et al.[7] and references herein) recently revisited popular problems previously studied in static graphs [6, 9, 19] in a dynamic context.

Casteigts et al [8] studied the possibility of discovering a *restless* temporal path between two nodes in a dynamic network with a waiting time constraint: at each step, the traveling agent cannot wait more than c time instants, where c is a given constant. It turns out that computing such paths is NP-Hard. Perhaps surprisingly, Villacis-Llobet et al [20] showed that if one allows going several times through the same node, the obtained restless temporal walk can arrive earlier, and finding it can be done in linear time. As previously mentioned, this line of work only considers *forward* time travel (a temporal path cannot go back in time), and focuses on offline settings.

Multi-criteria path computation problems have been extensively studied within computer networks [10, 16, 17]. In this context, each edge is characterized by a weight vector, comprising both cost and delay. Path computation algorithms thus have to maintain and explore all non-comparable paths, whose number may grow exponentially with respect to the size of the network, leading to the use of approximation schemes or heuristics. However, these works always focus on static graphs and offline settings.

As aforementioned, Bramas et al. [4] have proposed path computation algorithms on dynamic graphs with both forward and backward time-travel (assuming costly backward time-travel). They demonstrated the polynomial solvability of finding the path with minimum delay, even when constraining (or optimizing) the cost. Note that, conversely to us, such travels may not always allow for a delay of 0, if the constraint on the cost is too stringent. However, their study exclusively focuses on offline settings.

Related online problems in graphs include graph exploration and treasure hunting. Online graph exploration has been extensively studied in the literature in models that are similar to our S-online model, i.e., when visiting a node, the agent learns about the identifiers of the neighboring nodes. Algorithms with optimal competitive ratios were found in various classes of graphs such as cycles, tadpole graphs [5], trees [15], and arbitrary graphs [2], in undirected and directed [14] graphs. The case where more than one agent explores the graph has also been investigated [11, 12].

The treasure-hunting problem is equivalent to the problem of reaching a destination node with minimum latency, if considering the destination as the node where the treasure is located. Previous work on treasure hunting only considers static graph [1, 3], usually considering a different model where the agent sees outgoing edges, but lacks visibility of the neighboring nodes identifiers. In [18], the authors considered a model similar to ours, where the agent sees the neighboring nodes, and show that the optimal competitive ratio is $\Theta(n)$. This result implies the same asymptotic bounds in our model when assuming a linear cost function, but the exact bound remains unknown. Moreover, an exact bound in their setting cannot be generalized to our setting as an edge with a given cost requires several nodes to be emulated in our model.

To the best of our knowledge, our paper is the first to consider a problem similar to the online graph exploration and treasure-hunting problem in dynamic graphs.

Contributions. In this paper, we provide the following contributions:

- We introduce the problem of online space-time travel in dynamic networks and formally define several settings. In particular (a) in the T-online setting, an agent learns the past of the entire network when reaching a particular moment in time, and (b) in the S-online setting, an agent learns the past and future interactions involving the node where it is currently located.
- We present a T-online algorithm with an optimal competitive ratio able to compute a space-time travel with lowest delay and having a cost of at most two times the optimal cost.
- We present a lower bound of $2n/3 - 7/3$ for the competitive ratio of S-online algorithms, even if the cost function is the identity. In contrast, we provide a $2n - 3$ competitive S-online algorithm assuming a linear cost function. This algorithm is at most n^2 competitive for arbitrary (but feasible) cost functions.

Our work opens several problems, for instance, how to close the gap between our lower and upper bound regarding the competitive ratio of S-online algorithms.

2 Model

In this section, we define the models and notations used throughout this paper, before formalizing the aforementioned problems.

We represent the dynamic graph as an evolving graph, as introduced by Ferreira [13]: a graph-centric view of the network that maps a dynamic graph as a sequence of static graphs. The *footprint* of the dynamic graph (that includes all nodes and edges that appear at least once during the lifetime of the dynamic graph), is fixed. Furthermore, we assume that the set of nodes is fixed over time, while the set of edges evolves.

More precisely, an evolving graph G is a pair $(V, (E_t)_{t\in\mathbb{N}})$, where V denotes the set of vertices, $\mathbb{N}$ is the set of time instants, and for each $t \in \mathbb{N}$, E_t denotes the set of edges that appears at time t. The *snapshot* of G at time t is the static graph $G(t) = (V, E_t)$, which corresponds to the state, supposedly fixed, of the network in the time interval $t, t+1)$. The *footprint* $\mathcal{F}(G)$ of G is the static graph corresponding the union of all its snapshots, $\mathcal{F}(G) = \left(V, \bigcup_{t\in\mathbb{N}} E_t\right)$. We say $(\{u, v\}, t)$ is a temporal edge of graph G if $\{u, v\} \in E_t$. We say that an evolving graph is *connected* if its footprint is connected.

Space-time Travel. We assume that at each time instant, an agent can travel along any number of adjacent consecutive communication links. However, the graph may not be connected at each time instant, hence it may be that the only way to reach a particular destination node is to travel forward (i.e., wait) or backward in time, to reach a time instant where an adjacent communication link exists. In more detail, an agent travels from a node s to a node d using a *space-time travel* (or simply travel when it is clear from the context).

▶ **Definition 1.** *A* space-time travel *of* length k *is a sequence* $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ *such that*

- $\forall i \in \{0, \dots k\}$, *$u_i \in V$ is a node and $t_i \in \mathbb{N}$ is a time instant,*
- $\forall i \in \{0, \dots k-1\}$, *if $u_i \neq u_{i+1}$, then $t_i = t_{i+1}$ and $\{u_i, u_{i+1}\} \in E_{t_i}$ i.e., there is a temporal edge between u_i and u_{i+1} at time t_i.*

By extension, the *footprint* of a travel is the static graph containing all edges (and their adjacent nodes) appearing in the travel. Now, the *itinerary* of a travel $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ is its projection $(u_0, u_1, \dots, u_k)$ on nodes, while its *schedule* is its projection $(t_0, t_1, \dots, t_k)$ on time instants. Let $\mathcal{T}_G((u, t), (v, t'))$ denote the set of travels in G starting from node u at time t, and arriving at node v at time t'.

▶ **Definition 2.** *A travel $((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ is* simple *if for all $i \in \{2, \dots, k\}$ and $j \in \{0, \dots, i-2\}$, we have $u_i \neq u_j$.*

Intuitively, a travel is simple if its footprint is a line (i.e., a simple path) and contains at most one time-travel per node (as a consequence, no node appears three times consecutively in a simple travel).

▶ **Definition 3.** *The* delay *of a travel $T = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$, denoted delay$(T)$ is defined as $t_k - t_0$.*

The Backward cost of a travel.

▶ **Definition 4.** *The* backward-cost *is the cost of going to the past. The backward-cost function $\mathfrak{f} : \mathbb{N}^* \to \mathbb{R}^+$ returns, for each $\delta \in \mathbb{N}$, the backward-cost $\mathfrak{f}(\delta)$ of traveling δ time instants to the past. As we assume that there is no cost associated to forward time travel (that is, waiting), we extend $\mathfrak{f}$ to $\mathbb{Z}$ by setting $\mathfrak{f}(-\delta) = 0$, for all $\delta \in \mathbb{N}$. In particular, the backward-cost of traveling* 0 *time instants in the past is zero. When it is clear from context, the backward-cost function is simply called the cost function.*

▶ **Definition 5.** *The* backward-cost *(or simply cost) of a travel $T = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$, denoted cost$(T)$ is defined as follows:*

$$\mathit{cost}(T) = \sum_{i=0}^{k-1} \mathfrak{f}(t_i - t_{i+1})$$

▶ **Definition 6.** *Let $T_1 = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k))$ and $T_2 = ((u'_0, t'_0), (u'_1, t'_1), \dots, (u'_{k'}, t'_{k'}))$ be two travels. If $(u_k, t_k) = (u'_0, t'_0)$, then the* concatenated travel $T_1 \oplus T_2$ *is defined as follows:*

$$T_1 \oplus T_2 = ((u_0, t_0), (u_1, t_1), \dots, (u_k, t_k), (u'_1, t'_1), \dots, (u'_{k'}, t'_{k'}))$$

▶ Remark 7. One can easily prove that $\mathit{cost}(T_1 \oplus T_2) = \mathit{cost}(T_1) + \mathit{cost}(T_2)$. In the following, we sometimes decompose a travel highlighting an intermediate node: $T = T_1 \oplus ((u_i, t_i)) \oplus T_2$. Following the definition, this means that T_1 ends with (u_i, t_i), and T_2 starts with (u_i, t_i), so we also have $T = T_1 \oplus T_2$ and $\mathit{cost}(T) = \mathit{cost}(T_1) + \mathit{cost}(T_2)$.

Our notion of space-time travel differs from the classical notion of *journey* found in the literature related to dynamic graphs [13] as we do *not* assume time instants monotonically increase along a travel. As a consequence, some evolving graphs may not allow a journey from A to B yet allow one or several travels from A to B.

We say a travel is cost-optimal, if there does not exist a travel with the same departure and arrival node and times as T having a smaller cost. One can easily prove the following Property.

▶ **Property 1.** *Let T be a cost-optimal travel from node u to node v arriving at time t, and T' a sub-travel of T i.e., a travel such that $T = T_1 \oplus T' \oplus T_2$. Then T' is also cost-optimal.*

In the remaining of this paper, we consider a given evolving graph $G = (V, (E_t)_{t\in\mathbb{N}})$, a given cost function $\mathfrak{f}$, a source nodes s and a destination node d in V.

Problem specification. We consider an agent that travels in the evolving graph, starting from a node s at time 0. When at a node u at time t, the agent can either wait, go back in time, or traverse an edge to a neighboring node v if the temporal edge $(\{u, v\}, t)$ exists. If it waits, it stays in the same node, but the time is incremented by one. If it goes back in time, the new time can be any t', $0 \leq t' < t$, and the cost of this operation is $\mathfrak{f}(t - t')$. Note that when traveling in space the agent can traverse several edges during a single time instant. When time-traveling, the agent may travel back or forward in time to any time instant but will remain on the same node.

The goal of the agent is to reach the destination d with minimal delay, i.e., arriving at time 0. Notice that a backward time-travel is always necessary if no path exists to the destination during the first time instant, and that reaching d at time 0 is always possible if the footprint is connected.

This problem is trivial when the agent knows the entire evolving graph [4] (even when the cost is constrained, and when the backward time travels are limited in amplitude). In this paper, we consider that the agent has a limited initial knowledge of the evolving graph, and it can learn more information by moving in the graph (moving in the topological or temporal sense).

Online Algorithms. An online algorithm A is a function that takes as input tuple (G', t, u) where G' is a sub-graph of G representing the partial information of the agent, t the current time, and u a node where the agent is located. The algorithm outputs the action performed by the agent: wait, go back at time $t' < t$ or traverse an edge. For simplicity, we can consider without loss of generality that the output is a space-time travel T that exists in G'. By doing so, the agent may learn new information about the traversed nodes or they can wait to learn new information about the future. A single action (wait, go back in time, or traverse an edge) can be seen as an elementary space-time travel.

We consider only deterministic algorithms so that executing an online algorithm on a given evolving graph G makes the agent follow a single space-time travel (maybe of infinite length if the agent loops for infinity). On a given evolving graph G, the cost obtained by an online algorithm A is denoted $Cost(A, G)$ and is the cost of the space-time travel performed by the agent on this graph. For comparison, we denote by $Cost(opt, G)$ the optimal cost given by an optimal offline algorithm.

Our goal is to find an algorithm that minimizes the competitive ratio defined as follows.

▶ **Definition 8.** *An online algorithm has competitive ratio ρ if in any evolving graph G, we have*

$$\frac{Cost(A, G)}{Cost(opt, G)} \leq \rho$$

Acquiring new Knowledge. The way the agent learns about the evolving graph depends on the model. We consider two cases. First, in the *T-online* model, when an agent reaches time t, they learns about all the temporal edges $(\{u, v\}, t') \in E$ with $t' \leq t$, for all $u, v \in V$. In other words, they learns about the entire graph up to this time. Second, in the *S-online* setting, when an agent reaches a node u, it learns about all the temporal edges $(\{u, v\}, t') \in E$, for all $t' \geq 0$ and for all $v \in V$. In other words, it learns all the information, past and future, concerning the current node.

Observe that the definition of competitive ratio does not depend on the setting, as the offline optimal algorithm gives the same solution regardless of the setting. The setting just impacts the knowledge of the agent, so in practice, different knowledge should give different algorithms, and it might not be possible to obtain the same competitive ratio in two different settings. Hence, we are also interested in finding lower bounds for the competitive ratio in each setting.

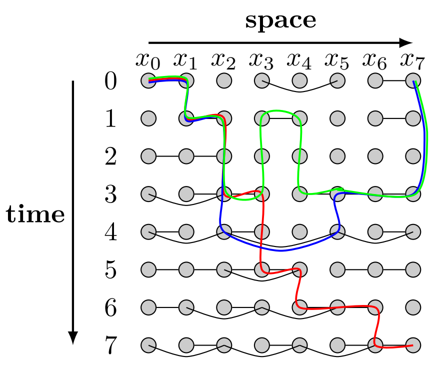

Figure 1 Possible representation of an evolving graph. Possible travels from x_0 to x_7 are shown in red, green, and blue. Note that the blue and green travels require sending an agent to the past (to a previous time instant).

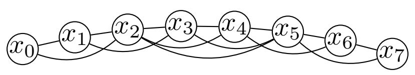

Figure 2 Footprint of the evolving graph represented in Figure 1.

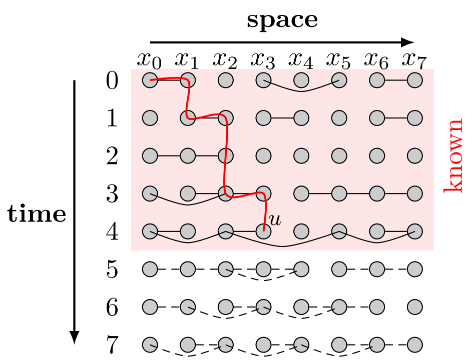

Figure 3 Example of the state of an agent during a T-online travel. The agent at position u does not know about the dashed edges

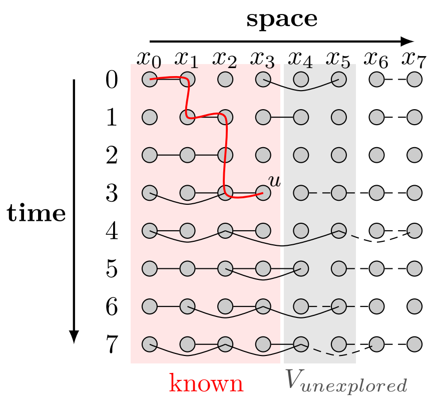

Figure 4 Example of the state of an agent during an S-online travel. The agent at position u does not know about the dashed edges, nor the unknown nodes outside $V_{unexplored}$.

Visual representation of online space-time travels. To help visualize the problem, consider a set of $n + 1$ nodes denoted $x_0, x_1, x_2, \ldots, x_n$. Then, the associated evolving graph can be seen as a vertical sequence of graphs mentioning for each time instant which edges are present. A possible visual representation of an evolving graph can be seen in Fig. 1. One can see the evolution of the topology (consisting of the nodes x_0 to x_7) over time through eight snapshots performed from time instants 0 to 7. Several possible travels are shown in red, green, and blue. The red travel only makes use of forward time travel (that is, waiting) but is the earliest arriving travel in this class (arriving at time 7, while it is possible to arrive at time 4). The green and blue travels both make use of backward time travel and arrive at time 0, so they have minimal travel delay. Similarly, the red travel concatenated with $((x_7, 7), (x_7, 0))$ (i.e., a backward travel to reach x_7 at time 0) also has minimal travel delay. However, if we assume that the cost function is the identity ($\mathfrak{f} : d \mapsto d$) then the green travel has a backward cost of 5, the blue travel has a backward cost of 4, and the concatenated red travel has a backward cost of 7.

The main challenge arises when an agent explores the graph in an online manner,i.e., learns about the graph while it is exploring it. Figure 3 and 4 illustrate the current knowledge of the agent after it traversed the red travel and is currently at node u. In Figure 3, the agent is T-online and knows about the entire past of the graph, i.e., it knows about all the edges that occurred at time 4 or before, regardless of the nodes involved. The agent knows about a possible travel to reach destination x_7, but does not know if it is cost-optimal depending on the cost function (with a cost function $\mathfrak{f} : x \mapsto x$, it knows that the blue travel is optimal).

In contrast, in Figure 4, the agent is S-online and knows about the past and the future of all the visited nodes $x_0, \dots x_3$. In this case, the agent does not know a travel to the destination yet, but it is challenging to decide what node to explore first to minimize the cost.

3 Backward-cost Function Classes

The cost function $\mathfrak{f}$ represents the cost of going back to the past. It has been shown by Bramas et al. [4] that it is necessary for $\mathfrak{f}$ to be non-negative and that it attains its minimum (not just converge to it) on every interval that includes infinity, for an optimal-delay optimal cost travel to exist. These conditions were also shown to be sufficient for an offline algorithm to find an optimal solution.

▶ **Definition 9.** *A cost function $\mathfrak{f}$ is* user optimizable *if it is non-negative, and it attains its minimum when restricted to any interval $[C, \infty)$, with $C > 0$. Let $\mathcal{UO}$ be the set of user optimizable cost functions.*

For simplicity, in this paper, we only consider *user friendly* cost functions as defined by Bramas et al. [4]:

▶ **Definition 10.** *A cost function $\mathfrak{f}$ is* user friendly *if it is user optimizable, non-decreasing, and sub-additive*[1]*. Let $\mathcal{UF}$ be the set of user friendly cost functions.*

Indeed, following the methodology by Bramas et al. [4], the optimal output of an algorithm using a user friendly function can be transformed into an optimal solution assuming the cost function is only user optimizable.

4 T-Online Algorithm with Optimal Competitive Ratio

In this section, we consider the T-online setting. In other words, the future of the evolving graph is unknown to the algorithm: at a time t, only the snapshots at time instants $t' \leq t$ are known. We first prove that there exists no algorithm with a competitive ratio smaller than 2, even if the cost function is the identity. Then we present our T-online Algorithm 1 and we show that it has an optimal competitive ratio.

[1] sub-additive means that for all $a, b \in \mathbb{N}$, $\mathfrak{f}(a + b) \leq \mathfrak{f}(a) + \mathfrak{f}(b)$

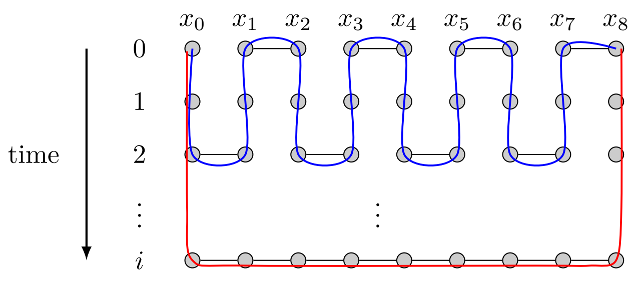

Figure 5 Definition of the evolving graphs G^i, with 9 nodes ($n = 8$). The blue travel T_8 has a backward-cost of 8. The red travel T_i has a backward cost of i.

▶ **Theorem 11.** *Assuming $\mathfrak{f} : x \mapsto x$, if the future is unknown, there exists no T-online algorithm with competitive ratio $2 - \varepsilon$, with $\varepsilon > 0$.*

Proof. Assume for the sake of contradiction that algorithm A is a T-online algorithm and has a competitive ratio of $2-\varepsilon$, with $\varepsilon > 0$. Let n be an even integer greater than $\frac{5}{\varepsilon}$. For any $i > 3$, let G^i be an evolving graph whose footprint is a line with $n + 1$ nodes $x_0, x_1, \ldots, x_n$ defined in the following way:

- $G^i(0)$ is the graph where half of the edges are present:

$$E^i(0) = \{\{x_k, x_{k+1}\} \mid k \in [0, n] \wedge k \equiv 1 \mod 2\}.$$

- $G^i(2)$ is the graph where the other half of the edges are present:

$$E^i(2) = \{\{x_k, x_{k+1}\} \mid k \in [0, n] \wedge k \equiv 0 \mod 2\}.$$

- $G^i(i)$ is a line graph : $E^i(i) = \{\{x_k, x_{k+1}\} \mid k \in [0, n-1]\}$.
- for all $j \notin \{0, 2, i\}$, $G^i(j)$ is a graph with no edge : $E^i(j) = \emptyset$.

It is clear that, in all such graphs G^i, there exists a travel from x_0 to x_n, denoted by T_n, with backward-cost n, using the edges present at time 0 and 2 (the blue travel in Figure 5). In addition, there exists a travel, denoted T_i, of backward-cost i in the evolving graph G^i (the red travel in Figure 5).

If $i > n$, the optimal travel is T_n, and if $i < n$ the optimal travel is T_i.

Let us now run Algorithm A on the evolving graph G^{2n}, with source being x_0 and destination x_n. Clearly, the algorithm cannot wait until time instant $2n$ otherwise the backward cost would be at least $2n$, which is two times more than the backward cost of the optimal path T_n. This implies that Algorithm A cannot distinguish between (hence runs exactly in the same way in) graphs G^i, with $i \geq 2n$. Let $t_{\max}$ be the maximum time instant reached by Algorithm A in G^{2n}. Then we can even say that A cannot distinguish between graphs G^i with $i > t_{\max}$.

Claim 1: *In G^i, with $i > t_{\max}$, Algorithm A outputs a travel with a backward-cost of at least $n + t_{\max} - 2$*

Proof of the Claim: The travel $T = A(G^i)$ that A outputs must contain the same temporal edges as T_n because those are the only edges that exist before time i (recall that $t_{\max} < i$). Let t_j be the time instant reached by Algorithm A at node x_j, for all $j = 0, \ldots, n$. Since at each node the travel either arrives at time 2 or leaves at time 2, then $\forall j \in [0..n], t_j \geq 2$.

To move from node x_j to node x_{j+1} the travel T includes a backward trip of cost t_j, if $j \equiv 1 \mod 2$, and of cost $t_j - 2$, otherwise. Let $t_{j_{\max}} = t_{\max} = \max(t_j)$, we have that

$$cost(T) = \sum_{j \in Odd(n)} t_j + \sum_{j \in Even(n)} t_j - 2 \geq \begin{cases} 2\,|Odd(n)| + t_{\max} - 2 & \text{if } j_{\max} \text{ is odd} \\ 2\,(|Odd(n)| - 1) + t_{\max} & \text{if } j_{\max} \text{ is even} \end{cases}$$

Where $Even(n)$, resp. $Odd(n)$, denotes the set of even, resp. odd, numbers smaller or equal to n, Since $|Odd(n)| = n/2$, we obtain in both case $cost(J) \geq n + t_{\max} - 2$

Claim 2: $t_{\max} \leq n - 4$

Proof of the Claim: Since algorithm A has a competitive ratio of $2 - \varepsilon$, then, if it runs in the evolving graph G^{2n}, it must return a path of backward-cost at most

$$(2 - \varepsilon)Cost(opt, G^{2n}) = (2 - \varepsilon)n < 2n - 5$$

(recall that $n\varepsilon > 5$), so it cannot reach time instant $n - 3$. Indeed, if the algorithm waits until time instant $t_{\max} \geq n - 3$, then, using the previous claim, the backward-cost of the travel would be at least $n + n - 5$.

Now we run Algorithm A on graph $G^{t_{\max}+1}$. Using Claim 1, we know that A returns a travel of cost at least $n + t_{\max} - 2$. However, in $G^{t_{\max}+1}$, since $t_{\max} + 4 \leq n$ (Claim 2), the optimal travel is $T_{t_{\max}+1}$ having a cost of $t_{\max} + 1$. We obtain the following inequality:

$$\begin{aligned} cost(A(G^{t_{\max}+1})) &\geq n + t_{\max} - 2 \geq t_{\max} + 4 + t_{\max} - 2 \geq 2(t_{\max} + 1) \\ &\geq 2Cost(opt, G^{t_{\max}+1}) \end{aligned}$$

This contradicts the fact that A has a competitive ratio of $2 - \varepsilon$. ◀

Our Algorithm 1 works as follows: the agent remains in the initial node and waits to learn more about the network until it finds a space-time travel to the destination. Since it does not know whether this is an optimal travel, it waits until it is sure that this is the case, and then goes back in time and enjoys its trip. Computing the best space-time travel given a sub-graph is possible in polynomial time using the existing offline algorithm [4]. Our algorithm assumes that the cost function tends to infinity when the input goes to infinity. It is easy to see that this assumption is necessary to achieve a constant competitive ratio. For instance, with a constant cost function that is equal to 1, one can create two indistinguishable (up to time n) dynamic graphs where a travel with cost $n - 2$ exists before time $t = n$. Then, the first graph contains no other travel, and the second contains another travel with cost 1 but requires a path available at time $t \gg n$ arbitrarily large. Not knowing which graph it is put in (at time n, the agent's knowledge about the two graphs is identical), the agent either waits indefinitely to see if the second travel exists, or follows the first travel with a competitive ratio of $n - 2$. A similar argument can be constructed when the cost function is upper bounded by some number C when its input goes to infinity.

▶ **Theorem 12.** *For any $\mathfrak{f} \in \mathcal{UF}$ that diverges to infinity, Algorithm 1 is a T-online algorithm with a competitive ratio of 2.*

Proof. Let $T_{\max}$ be the final value of the variable in our algorithm, so it is the space-time travel used by the agent after the agent returns back at time 0. Let

$$t_{\max} = \max\{t \mid \mathfrak{f}(t + 1) < cost(T_{\max})\}.$$

Algorithm 1 T-Online Algorithm.

```
Input :
        $G'$: the known evolving graph
        $time$: the current time
        $u$: the current node (here $u$ is always the starting node $s$)
Let $T_{\max}$ be an optimal space-time travel in $G'$, if it exists, starting at time 0, from node $s$
  to node $d$.
if $T_{\max}$ does not exist or $\mathfrak{f}(time + 1) < cost(T_{\max})$ then
    wait 1 time instant;
else
    go back at time 0, then follow the travel $T_{\max}$;
```

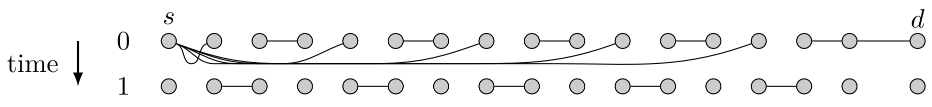

Figure 6 Evolving graph G used to prove that an S-online algorithm has a competitive ratio of at least $2n/3 - 7/3$.

First, we prove that $T_{\max}$ is an optimal offline travel. Indeed, the algorithm reached time $t_{\max}$ so all the other travels that are not discovered by our algorithm require temporal edges appearing after time $t_{\max}$, so their backward-costs are at least $\mathfrak{f}(t_{\max} + 1)$, which is at least $cost(T_{\max})$ by definition. Hence $cost(T_{\max})$ is the optimal backward-cost.

When the algorithm terminates, the travel T that is returned is $((s, 0),(s, t_{\max}),(s, 0)) \oplus T_{\max}$. It has a backward-cost of $\mathfrak{f}(t_{\max}) + cost(T_{\max})$. Since $\mathfrak{f}(t_{\max}) \leq cost(T_{\max})$, the Lemma is proved. ◀

5 S-Online Algorithm

In this section, we study the S-online setting, where an agent knows only about the nodes they have explored. In this case, we show a lower bound of $2n/3 - 7/3$ for the competitive ratio, even when the cost function is the identity. We then present an algorithm that has a competitive ratio of $2n - 3$ when the cost function is linear.

▶ **Theorem 13.** *Assuming a cost function is $\mathfrak{f} : x \mapsto x$, an S-online algorithm cannot have a competitive ratio smaller than $2n/3 - 7/3$, where n denotes the number of nodes in the graph.*

Proof. Assume for the purpose of contradiction that there exists an S-online algorithm A with a competitive ratio c.

Consider an evolving graph G consisting of k paths of length 3 having one node s in common and append a node d to one of the paths, as illustrated in Figures 6 and 7. Links connecting nodes at hop-distance 1 from s and nodes at hop-distance 2 from s appear at time 1, and all the other links appear at time 0. The number of nodes is $n = 3k + 2$.

An agent traveling in this graph initially knows about all the neighbors of s does not know their neighbors. Since all the edges connected to s appear at the same time 0, they are indistinguishable. We can consider without loss of generality that the branches are explored from left to right, direction according to Figure 7 (a branch is explored when the node at hop-distance 3 is visited). Visiting a branch costs $\mathfrak{f}(1)$, and going back to s also costs $\mathfrak{f}(1)$, so when the agent finally visits the last branch that is connected to d, they have paid $2(k-1)+1$.

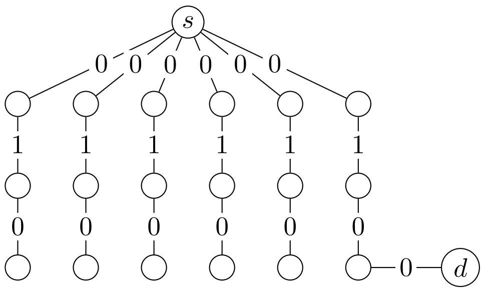

Figure 7 The same evolving graph G shown in Figure 6 as a static graph where the label represents the time where the edge is present.

In the end the travel costs

$$2(k-1)+1 = 2k-1 = \frac{2(n-2)}{3} - 1 = 2n/3 - 7/3$$

while the optimal travel costs 1. So the competitive ratio is $2n/3 - 7/3$. ◀

One important question is whether or not the lower bound is higher when assuming a cost function $\mathfrak{f}$ that is not linear. Interestingly, one can observe that, to create a worst-case using non-linear functions, one must use longer paths because the travel cost of a two-hop travel is the same in one way and the other, so longer travels are required to create travels having higher cost. Moreover, following k smaller backward travels costs at most k times the equivalent single but larger backward travel. So we conjecture that the lower bound is the same for any cost function in $\mathcal{UO}$.

We now present our Algorithm 2. At a given step, the agent is located at a node u at time t and knows a subgraph G' of G. Among the known nodes, some are not yet explored, called $V_{unexplored}$. For a node v in $V_{unexplored}$, the agent does not know its entire neighborhood. In particular, it does not know if it is connected to the destination d. Indeed, d is either unknown or unexplored (otherwise the agent has already reached the destination). The goal of the agent is to find a travel towards the destination that is not too expensive. A possible travel to d is either in G' or goes through a node in $V_{unexplored}$. A travel in G to d that is not in G' must go through a node $v \in V_{unexplored}$, so its cost is at least the cost of a travel towards v at time 0. So the main idea of the algorithm is to explore nodes one by one starting from the one that could potentially be an optimal travel to the destination d i.e., the agent visits first a node $v \in V_{unexplored}$ whose travel from s to v at time 0 is minimal.

To illustrate a step of the algorithm, consider Figure 4. The agent at position u does not know any travel towards d, but it knows that a travel must either go through x_4 or x_5. In the best case, if it goes through x_4, a travel to d costs at least 3 (it costs 2 to reach x_4 using the edge (x_3, x_4) at time 1 plus at best a backward travel to 0 if d is directly connected to x_4. If it goes through x_5, a travel to d costs at least 4. So in this situation, the agent travels towards x_4. When it reaches x_4, the agent realizes that it is not connected to d and that a travel to d passing through x_4 costs at least 5 (the green travel in Figure 1) so the agent decides to explore next x_5. Then it explores x_6 and finally $d = x_7$.

▶ **Theorem 14.** *Assuming the cost function is linear, Algorithm 2 is an S-online algorithm with a competitive ratio of $2n-3$, where n denotes the number of nodes.*
Assuming non-linear $\mathfrak{f} \in \mathcal{UF}$, the competitive ratio of Algorithm 2 is at most n^2.

Proof. Consider first that the cost function is linear. After each iteration of the algorithm, at least one new node v is explored.

Algorithm 2 S-Online Algorithm.

```
Input :
        $G'$: the known evolving graph
        $time$: the current time
        $u$: the current node
Let $V_{unexplored}$ be the set of nodes known but unexplored;
Let $H = \{T_v = \mathcal{T}_{G'}((s,0),(v,0)) | v \in V_{unexplored}\}$;
/* Recall that $\mathcal{T}_{G'}((s,0),(v,0)$ is the set of travels from $s$ to $v$ arriving and
   departing at time 0                                                          */
Let $T_v$ be a space-time travel in $H$ with minimum cost
Follow an optimal space-time travel towards $(v,0)$ in $G'$, from the current location $u$.
```

The cost of the travel from s to v arriving at time 0 is at most the cost $Cost(opt, G)$ of the optimal travel from s to d arriving at time 0. Indeed, if d is unknown, all travels goes through at least one unexplored node, meaning that the cost to reach d is higher or equal than the cost to reach v. If d is known, the travels towards d are included in the set H.

Note that when traveling back to s at time 0 from v at time 0, backward time jumps become waiting, and vice-versa. Hence, an outbound trip requiring several, small waits and a large backward-time jump translate in a return trip with a large wait and several small backward-time jumps, which, intuitively, mais lead to different costs.

However, because we first assume the cost function to be linear (i.e. $f(a+b) = f(a)+f(b)$, the cost of the outbound trip is the same as the cost of the return trip. Thus, to visit the next unexplored node v', the agent has to, in the worst case, go back to s and then travel to v', incurring a cost of $2Cost(opt, G)$. In the worst case, the destination d is the last visited node. The total cost is at most $2(n-2)Cost(opt, G)$ for the first $n-2$ nodes (all except s and d) plus the last travel towards d, i.e., at most $2(n-2)Cost(opt, G) + Cost(opt, G)$ and the competitive ratio is $2n-3$.

Now, suppose the cost function $\mathfrak{f} \in \mathcal{UF}$ is non-linear. In this case, the costs of the outbound and return trip may be different.

Let T be a k-hop travel from s to a node v, arriving and departing at time 0. Let T_r be the associated returned trip. Observe that the sum of the amplitude (denoted Δ in the following) of all backward-time jumps is the same in both directions (i.e., for T and T_r). Also, it is clear that each backward jump performed in T_r has amplitude smaller than Δ and since $\mathfrak{f}$ is non-decreasing, each of these jumps costs at most $\mathfrak{f}(\Delta)$. Since there are at most $k \leq n$ backwards jumps in T_r, we know that, $cost(T_r) \leq k \times \mathfrak{f}(\Delta)$. As $\mathfrak{f}$ is also sub-additive, we know that $\mathfrak{f}(\Delta) \leq cost(T)$. Thus, we have $cost(T_r) \leq k \times cost(T)$.

Hence, using the same proof as in the previous case, we obtain in the worst case, a cost of at most $n \times Cost(opt, G)$ to explore the next node v, which results in a total cost of at most $n^2 Cost(opt, G)$. ◀

6 Discussion and Open Problems

One may notice that the strategies used in Algorithm 2 are similar to some works related to online static graph exploration and treasure hunting. These similarities raise interesting questions regarding possible relations between these problems and possible applications of existing works to our novel model.

Studying this question is challenging due to the many different existing models. For instance, many papers about treasure hunting in static graphs assume that the agent located at a node does not learn about the identifier of the neighboring nodes, but only about the outgoing edges [3, 1]. This results in bounds based on the number of edges, while our algorithm performance only depends on the number of nodes.

One paper by Komm et al. [18] considers that an agent learns about the identifier of its neighbors (a model dubbed *fixed graph*). Interestingly, our models align with theirs under the assumption of a linear cost function. We, however, give a more refined bound about the competitive ratio, implying perhaps a more precise bound for the model given by Komm et al. [18] (which briefly mentions an asymptotically linear competitive ratio before focusing on the impact of advice fed to the agent).

When assuming a *non-linear* cost function, our problem seems to exhibit similarities with the treasure-hunting problem in directed graphs when agents see the neighboring nodes (a problem that, to our knowledge, is unexplored in the literature).

Our work in the S-online setting can thus be seen as the first generalization of the treasure-hunting problem in dynamic graphs. It is interesting to see that such generalizations bear similarities to their static counterparts. However, these outcomes depend on the agent's ability to engage in time travel. In the absence of such capabilities, certain assumptions about the graph may be needed to make up for the absence of backward time travel. For instance, one might consider assumptions such as periodicity, or the presence of bounded-recurrent edges, allowing for the traversal of disappearing edges by waiting for their reappearance, instead of traveling back in time. The study of relations between the settings studied in this paper and more general ones, as well as the study of associated complexity results, are open.

7 Conclusion

We presented the first online solutions to the delay-optimal cost-optimal space-time travel problem in dynamic networks.

We first showed that, when the future is unknown, even assuming an identity cost function, no online algorithm can exhibit a competitive ratio of less than two, and we present a very simple online algorithm with a competitive ratio of two, for a larger class of cost functions.

Then, when the graph itself is unknown and has to be explored to gain connectivity knowledge, we showed that there exists a linear (in the size of the graph) lower bound on the competitive ratio, even when the cost function is the identity, and we present an algorithm with a linear (in the size of the graph) competitive ratio assuming any linear cost function. Refining the constants between our lower and upper bound is left for future work.

References

1 Adri Bhattacharya, Barun Gorain, and Partha Sarathi Mandal. Treasure hunt in graph using pebbles. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 99–113. Springer, 2022.

2 Alexander Birx, Yann Disser, Alexander V Hopp, and Christina Karousatou. An improved lower bound for competitive graph exploration. *Theoretical Computer Science*, 868:65–86, 2021.

3 Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Almost-optimal deterministic treasure hunt in unweighted graphs. *ACM Transactions on Algorithms*, 19(3):1–32, 2023.

4 Quentin Bramas, Jean-Romain Luttringer, and Sébastien Tixeuil. Offline constrained backward time travel planning. In Shlomi Dolev and Baruch Schieber, editors, *Stabilization, Safety, and*

Security of Distributed Systems - 25th International Symposium, SSS 2023, Jersey City, NJ, USA, October 2-4, 2023, Proceedings, volume 14310 of *Lecture Notes in Computer Science*, pages 466–480. Springer, 2023. doi:10.1007/978-3-031-44274-2_35.

5 Sebastian Brandt, Klaus-Tycho Foerster, Jonathan Maurer, and Roger Wattenhofer. Online graph exploration on a restricted graph class: Optimal solutions for tadpole graphs. *Theoretical Computer Science*, 839:176–185, 2020.

6 Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. Shortest, fastest, and foremost broadcast in dynamic networks. *Int. J. Found. Comput. Sci.*, 26(4):499–522, 2015. doi:10.1142/S0129054115500288.

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

8 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.

9 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *J. Comput. Syst. Sci.*, 121:1–17, 2021. doi:10.1016/j.jcss.2021.04.004.

10 Shigang Chen and Klara Nahrstedt. An overview of qos routing for the next generation high-speed networks: Problems and solutions. *Network, IEEE*, 12:64–79, December 1998. doi:10.1109/65.752646.

11 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Pająk, and Przemysław Uznański. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015.

12 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Škorić, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theoretical Computer Science*, 811:70–78, 2020.

13 Afonso Ferreira. On models and algorithms for dynamic communication networks: The case for evolving graphs. In *Quatrièmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (ALGOTEL 2002)*, pages 155–161, Mèze, France, May 2002. INRIA Press.

14 Klaus-Tycho Foerster and Roger Wattenhofer. Lower and upper competitive bounds for online directed graph exploration. *Theoretical Computer Science*, 655:15–29, 2016.

15 Robin Fritsch. Online graph exploration on trees, unicyclic graphs and cactus graphs. *Information Processing Letters*, 168:106096, 2021.

16 Rosario G. Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, December 2010. doi:10.1016/j.comnet.2010.05.017.

17 Jochen W. Guck, Amaury Van Bemten, Martin Reisslein, and Wolfgang Kellerer. Unicast qos routing algorithms for sdn: A comprehensive survey and performance evaluation. *IEEE Communications Surveys & Tutorials*, 20(1):388–415, 2018. doi:10.1109/COMST.2017.2749760.

18 Dennis Komm, Rastislav Královič, Richard Královič, and Jasmin Smula. Treasure hunt with advice. In Christian Scheideler, editor, *Structural Information and Communication Complexity*, pages 328–341, Cham, 2015. Springer International Publishing.

19 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*, pages 987–997. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00098.

20 Juan Villacis-Llobet, Binh-Minh Bui-Xuan, and Maria Potop-Butucaru. Foremost non-stop journey arrival in linear time. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 283–301. Springer, 2022. doi:10.1007/978-3-031-09993-9_16.