Temporal Connectivity: Coping with Foreseen and Unforeseen Delays

Eugen Füchsle
Faculty IV, Algorithmics and Computational Complexity, TU Berlin, Germany

Hendrik Molter
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Rolf Niedermeier
Faculty IV, Algorithmics and Computational Complexity, TU Berlin, Germany

Malte Renken
Faculty IV, Algorithmics and Computational Complexity, TU Berlin, Germany

Abstract

Consider planning a trip in a train network. In contrast to, say, a road network, the edges are *temporal*, i.e., they are only available at certain times. Another important difficulty is that trains, unfortunately, sometimes get delayed. This is especially bad if it causes one to miss subsequent trains. The best way to prepare against this is to have a connection that is *robust* to some number of (small) delays. An important factor in determining the robustness of a connection is how far in advance delays are announced. We give polynomial-time algorithms for the two extreme cases: delays known before departure and delays occurring without prior warning (the latter leading to a two-player game scenario). Interestingly, in the latter case, we show that the problem becomes PSPACE-complete if the itinerary is demanded to be a path.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness; Mathematics of computing → Discrete mathematics

Keywords and phrases Paths and walks in temporal graphs, static expansions of temporal graphs, two-player games, flow computations, dynamic programming, PSPACE-completeness

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.17

Related Version *Full Version*: `https://arxiv.org/abs/2201.05011` [15]

Funding *Hendrik Molter*: Supported by the ISF, grant No. 1070/20.
Malte Renken: Supported by the DFG, project MATE (NI 369/17).

1 Introduction

Computing *temporal paths* is one of the back-bone algorithmic problems in the context of temporal graphs, that is, graphs whose edges are present only at certain, known points in time [3, 26]. Temporal graphs are specified by a set V of vertices and a set E of time arcs, where each time arc $(v, w, t, \lambda) \in E$ consists of a *start vertex* v, an *end vertex* w, a *time label* t, and a *traversal time* λ; then there is a (direct) connection from v to w starting at time t and arriving at time $t + \lambda$. Temporal graphs model numerous real-world scenarios [16, 17, 5, 20]: Social, communication, transportation, and many other networks are usually not static but vary over time.

The added dimension of time causes many aspects of connectivity to behave quite differently from static (i.e., non-temporal) graphs. Thus, the flow of items through a temporal network has to be time-respecting. More specifically, it follows a *temporal walk* (or *path*, if every vertex is visited at most once), i.e., a sequence of time arcs $(v_i, w_i, t_i, \lambda_i)_{i=1}^{\ell}$

where $v_{i+1} = w_i$ and $t_{i+1} \geq t_i + \lambda_i$ for all $i < \ell$. While inheriting many properties from static walks, temporal walks exhibit certain characteristics that add a further level of difficulty to algorithmic problems centered around them. For example, temporal connectivity is not transitive: the existence of a temporal walk from vertex u to v and a temporal walk from v to w does not imply the existence of a temporal walk from u to w. Moreover, the temporal setting naturally leads to several notions of what an "optimal" temporal path could be [2, 3].

Computation of temporal paths and walks has already been studied intensively [26, 3], including specialized settings that are novel to temporal graphs: For example, Bentert et al. [2] and Casteigts et al. [6] studied temporal walks and paths that are only allowed to have limited waiting time at any vertex.

In this work, we investigate another natural, inherently temporal connectivity problem. It addresses *delays*. In many real-world temporal networks such as transport networks (e.g. trains, shipping routes), individual edges may get delayed for various reasons. Thus it is an important question whether connectivity between a start and a target node is *fragile*, i.e., easily disrupted by delays, or whether it is *robust*.

An important aspect in this matter is the time at which delays become known. The earlier they are announced, the easier one can still adapt the chosen route. In this work, we study the two endpoints of this spectrum: one where all delays are known up front, and one where delays occur without any prior warning.

We now briefly describe our models for these two problems, beginning with the problem variant in which all delays are announced before the start of the journey, and the question is whether the designated target remains reachable even in a worst-case scenario. Herein, a *D-delayed temporal path* refers to a temporal path that remains valid when the time arcs in $D \subseteq E$ have been delayed by some fixed amount δ each. A more formal definition will be given in Section 2.

Delay-Robust Connection
Input: A temporal graph $\mathcal{G} = (V, E)$, two vertices $s, z \in V$, and $x, \delta \in \mathbb{N}$.
Question: Is there, for every delay set $D \subseteq E$ of size $|D| \leq x$, a D-delayed temporal path from s to z in $\mathcal{G}$?

In contrast, if the delays occur during the journey without prior warning, then the resulting problem is best modeled as a two-player game, which we call the Delayed-Routing Game. The first player (the *traveler*) starts at vertex s and has to decide at each turn which time arc they want to traverse next. The other player (the *adversary*) then gets to decide whether that time arc is delayed or not. As before, there is a bound x on the overall number of time arcs that can be delayed. The traveler wins if they reach the target vertex z. A *winning* strategy for the traveler is a strategy that guarantees that they will reach their target.

Delayed-Routing Game
Input: A temporal graph $\mathcal{G} = (V, E)$, two vertices $s, z \in V$, and $x, \delta \in \mathbb{N}$.
Question: Does the traveler have a winning strategy for the Delayed-Routing Game?

The difference between the two models Delay-Robust Connection and Delayed-Routing Game is illustrated in Figure 1.

Finally, we want to consider a variant of Delayed-Routing Game, in which the traveler may not visit any vertex more than once. We refer to this as Delayed-Routing Path Game.

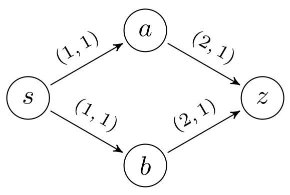


Figure 1 An example temporal graph, time arcs are labeled by $(\mathrm{t}(e), \lambda(e))$. For $x = 1$ and $\delta = 1$, this is a yes-instance of Delay-Robust Connection: If the time arc from s to a is delayed then there is a temporal path from s to z via b. Otherwise, if that time arc is not delayed, then the temporal path via a is available. However, the same setting is a no-instance of Delayed-Routing Game: If the traveler picks the time arc from s to a, then the adversary will delay it, rendering the traveler stuck at a since they reach it at time 3. If the traveler picks the time arc from s to b, then the situation is analogous.

Related Work. There has been extensive research on many other connectivity-related problems on temporal graphs [4, 11, 19, 21, 9, 10, 12, 27, 18]. Delays in temporal graphs have been considered in terms of manipulating reachability sets [7, 22]. An individual delay operation considered in the mentioned work delays a single time arc and is similar to our notion. Typically, the computational problems in this context are NP-hard and can be also considered as computing "robustness measures" for the connectivity in temporal graphs.

In companion work [14] we investigate a problem located somewhat "between" Delay-Robust Connection and Delayed-Routing Game in which the delays become known after the sequence of vertices to be traversed from s to z is fixed, but before the exact time arcs to be traversed are chosen. There, we show that this problem is NP-hard and further study its parameterized complexity.

Our Contribution. We introduce two computational problems related to testing connectivity between two terminal vertices in the presence of delays. We give polynomial-time algorithms for Delay-Robust Connection (Section 3) and Delayed-Routing Game (Section 4.1), but prove PSPACE-completeness for Delayed-Routing Path Game (Sections 4.2 and 4.3), the variant of the second problem in which no vertex may be visited twice. Due to space constraints, the PSPACE-hardness proof for Delayed-Routing Path Game is partially deferred to a full version [15].

2 Preliminaries

We abbreviate $\{1, 2, \ldots, n\}$ as $[n]$ and $\{n, n+1, \ldots, m\}$ as $[n, m]$. The Iverson bracket $[P]$ is 1 if property P holds and 0 otherwise. For a time arc $e = (v, w, t, \lambda_e)$, we denote the starting and ending vertices as $\mathrm{start}(e) = v$ and $\mathrm{end}(e) = w$, the time label as $\mathrm{t}(e) = t$, and the traversal time as $\lambda(e) = \lambda_e$. For a vertex v, τ_v denotes the set of time steps where v has incoming or outgoing time arcs.

Delays. When a time arc e gets delayed, then its traversal time $\lambda(e)$ is increased by some fixed amount δ. For a given set $D \subseteq E$ of *delayed arcs*, a sequence of time arcs $(v_i, w_i, t_i, \lambda_i)_{i=1}^{\ell}$ is called a *D-delayed temporal walk* if it is a temporal walk in the temporal graph obtained from $\mathcal{G}$ by applying delays to all time arcs in D. (We omit D when it is clear from the context.)

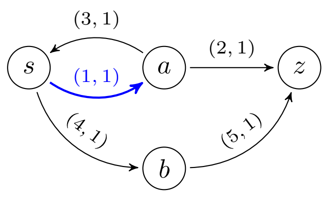


Figure 2 A temporal graph which, for $x = \delta = 1$, forms a yes-instance of Delayed-Routing Game but a no-instance of Delayed-Routing Path Game. In the former, a winning strategy for the traveler is to take the thick blue time arc to a. If it is not delayed, then they can directly go to z. If it is delayed, then there are no remaining delays. Thus, after returning to s, the target vertex z can be reached via b.

As an example consider the temporal walk $(a, b, 1, 1), (b, c, 3, 1)$:

$$\begin{tikzcd} a \arrow[r, "{(1,1)}"] & b \arrow[r, "{(3,1)}"] & c \end{tikzcd}$$

When delaying the first time arc by 1, i.e. having $\delta = 1$ and $D = \{(a, b, 1, 1)\}$, then this is also a delayed temporal walk: Due to the delay, the first time arc arrives in b at time step $2 + \delta = 3$ which is still not later than the departure of the second time arc. However, if we instead set $\delta = 2$ and $D = \{(a, b, 1, 1)\}$, then it is no longer a delayed temporal walk, because the first time arc only reaches b at time 4.

Clearly, from any temporal walk one can obtain a temporal path by eliminating all circular subwalks. Thus, for any delay set D, if there is a D-delayed temporal walk from s to z, then there is also a D-delayed temporal path. This is the reason why we did not define a separate version of Delay-Robust Connection for temporal walks.

Note that this equivalence does not extend to Delayed-Routing Game as Figure 2 proves.

Static expansion. Sometimes, a problem on a temporal graph $\mathcal{G}$ is transformed to problems on a non-temporal "time-expanded" graph, a *static expansion* of $\mathcal{G}$. The idea is to model each temporal occurrence of every vertex in the temporal graph as a distinct vertex of the static expansion.

Formally, we say that a digraph $H = (W, A)$ is a *static expansion* of the temporal graph $\mathcal{G} = (V, E)$ if

(i) $\bigcup_{v \in V} (\{v\} \times \tau_v) \subseteq W \subseteq V \times \mathbb{N}$, and

(ii) $A = A_1 \cup A_2$ with

$$A_1 = \left\{ (v, t), (v, t') \;\middle|\; (v, t) \in W \wedge t' = \arg\min_{t'' > t} \{(v, t'') \in W\} \right\},$$
$$A_2 = \{((v, t), (w, t + \lambda)) \mid (v, w, t, \lambda) \in E\}.$$

The arcs in A_1 are often called *waiting arcs* while the arcs in A_2 are in one-to-one correspondence to the time arcs of $\mathcal{G}$. We call the arcs in A_2 arcs *corresponding* to E. The static expansion with the minimal set of vertices is called the *reduced static expansion*.

The main virtue of static expansions is that they model temporal walks as (non-temporal) paths. More precisely, we have the following lemma.

▶ **Lemma 1.** *If (v,t) and (w,u) are two vertices of a static expansion H of $\mathcal{G}$, then there is a path from (v,t) to (w,u) if and only if $\mathcal{G}$ contains a temporal walk from v to w which starts at time t or later and arrives at time u or earlier.*

The proof of Lemma 1 is folklore; we omit it, as well as the proof of the following easy observation.

▶ **Observation 2.** *If $H = (W, A)$ is a static expansion of $\mathcal{G} = (V, E)$, $E' \subseteq E$ is a set of time arcs and $A' \subseteq A$ the set of arcs corresponding to E', then $(W, A \setminus A')$ is a static expansion of $(V, E \setminus E')$.*

3 Delay-Robust Connection

In this section, we present an algorithm (Algorithm 1) which solves Delay-Robust Connection in polynomial time. The core idea is to reduce the problem to the computation of a maximum flow problem in a static expansion. There are three steps in this algorithm. First, we construct a new temporal graph $\mathcal{G}^*$ in which the removal of a time arc has the same effect as delaying the respective time arc in the original input graph $\mathcal{G} = (V, E)$. Second, we construct a static expansion H of $\mathcal{G}^*$. Finally, we compute the maximum flow from the start vertex s to the target vertex z in H. We will show that the value of this flow equals the number of delays required to break temporal connectivity between s and z in $\mathcal{G}$.

For the first step, define $\mathcal{G}^* = (V, E \cup E^*)$ where $E^* = \{(v, w, t, \lambda + \delta) \mid (v, w, t, \lambda) \in E\}$. Then we can observe the following property of $\mathcal{G}^*$.

▶ **Lemma 3.** *Let $\mathcal{G} = (V, E)$ be a temporal graph, $s, z \in V$, $D \subseteq E$, and $\mathcal{G}^* = (V, E \cup E^*)$ as defined above. There is a D-delayed temporal (s,z)-walk in $\mathcal{G}$ if and only if there is a temporal (s,z)-walk in $\mathcal{G}^*_D := (V, (E \setminus D) \cup E^*)$.*

Proof.
($\Rightarrow$): Let $W = (e_1, e_2, \ldots, e_k)$ be a D-delayed walk from s to z in $\mathcal{G}$ with $e_i = (v_i, w_i, t_i, \lambda_i)$ for $i \in [k]$. This means that $v_1 = s$, $w_k = z$, and for all $\ell \in [k-1]$ it holds that $w_\ell = v_{\ell+1}$ and $t_\ell + \lambda_\ell + [e_\ell \in D] \cdot \delta \leq t_{\ell+1}$. We construct the temporal walk $\hat{W} = (\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_k)$ in $\mathcal{G}^*_D$ with $\hat{e}_i = (v_i, w_i, t_i, \lambda_i + [e_i \in D] \cdot \delta)$ for $i \in [k]$. Then $t(\hat{e}_\ell) + \lambda(\hat{e}_\ell) \leq t(\hat{e}_{\ell+1})$ holds for all $\ell \in [k-1]$, thus $\hat{W}$ is a temporal walk from s to z in $\mathcal{G}^*_D$.

($\Leftarrow$): Let $W = (e_1, e_2, \ldots, e_k)$ be a temporal (s,z)-walk in $\mathcal{G}^*_D$ with $e_i = (v_i, w_i, t_i, \lambda_i)$. This means that $v_1 = s, w_k = z$ and for all $\ell \in [k-1]$, it holds that $w_\ell = v_{\ell+1}$ and $t_\ell + \lambda_\ell \leq t_{\ell+1}$. We construct a delayed temporal walk $\hat{W} = (\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_k)$ in $\mathcal{G}$ for the delay set D with

$$\hat{e}_i = \begin{cases} e_i, & \text{if } e_i \in E \setminus D \\ (v_i, w_i, t_i, \lambda_i - \delta), & \text{otherwise} \end{cases}$$

for $i \in [k]$. Note that $\hat{e}_i \in E$: If $e_i \notin E \setminus D$, then $e_i \in E^*$, and thus $\hat{e}_i = (v_i, w_i, t_i, \lambda_i - \delta) \in E$ by construction of E^*. We then have for all $\ell \in [k-1]$ that

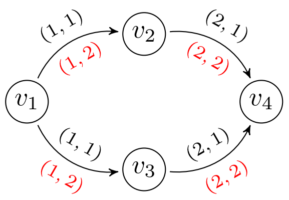
$$t(\hat{e}_\ell) + \lambda(\hat{e}_\ell) + [\hat{e}_\ell \in D] \cdot \delta = t(e_\ell) + \lambda(e_\ell) \leq t(e_{\ell+1}) = t(\hat{e}_{\ell+1})$$

holds, hence $\hat{W}$ is a D-delayed temporal walk in $\mathcal{G}$. ◀

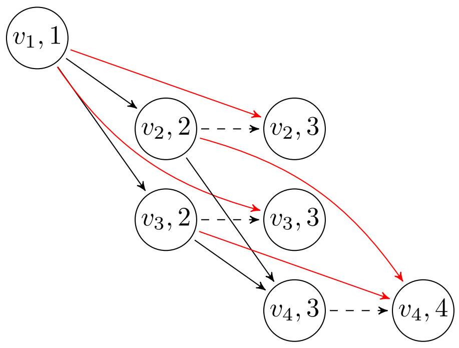
The algorithm for Delay-Robust Connection is given as pseudo-code in Algorithm 1. See also Figure 3 for an illustration of the graph H. We now show the correctness of the algorithm.

Algorithm 1 Delay-Robust Connection.

```
Input: $\mathcal{G} = (V, E), s, z \in V, x, \delta \in \mathbb{N}$
Output: Is $(\mathcal{G}, s, z, x, \delta)$ a yes-instance of Delay-Robust Connection?
1: $H \leftarrow$ reduced static expansion of $\mathcal{G}^*$
2: Define $c : E(H) \to \{1, \infty\}$ by $c(e) = \begin{cases} 1 & \text{if } e \text{ corresponds to a time arc in } E \\ \infty & \text{otherwise} \end{cases}$
3: $T \leftarrow \arg\max_t \{(z, t) \in V(H)\}$
4: Compute the value $f$ of a maximum flow from $(s, 0)$ to $(z, T)$ in $H$ with capacities $c$
5: if $f \leq x$ then
6:     return "NO"
7: else
8:     return "YES"
```



(a) The temporal graph $\mathcal{G}^*$. The black part of the figure shows $\mathcal{G}$, while the red time arcs are in E^*.



(b) The reduced static expansion of $\mathcal{G}^*$. The dashed arcs are waiting arcs and the black part forms a static expansion of $\mathcal{G}$. Note that the capacity function c assigns 1 exactly to the solid black edges.

Figure 3 Example depiction of $\mathcal{G}^*$ and its reduced static expansion ($\delta = 1$).

▶ **Lemma 4.** *Algorithm 1 solves* Delay-Robust Connection.

Proof. By Lemma 3, the given instance of Delay-Robust Connection is a no-instance if and only if there exists a set $D \subseteq E$ of size $|D| \leq x$ such that $\mathcal{G}^*_D$ contains no temporal (s, z)-walk. By Lemma 1 and Observation 2, this is equivalent of there being a set $D' \subseteq E(H)$ of edges in the static expansion H of $\mathcal{G}^*$ such that

(i) $|D'| \leq x$ and the edges in D' all correspond to edges in E, and

(ii) $H - D'$ contains no walk from $(s, 0)$ to (z, T) where T is the largest integer for which H contains the vertex (z, T).

Note that ii is equivalent to D' forming a cut set that separates $(s, 0)$ and (z, T). Also, by definition of the capacity function c, i is equivalent to the total capacity $\sum_{e \in D'} c(e)$ being at most x.

Therefore, by the Max-Flow-Min-Cut-Theorem [13, 8], the given instance is a no-instance if and only if the maximum flow from $(s, 0)$ to (z, T) in the graph H with edge capacities c is at most x. ◀

Next, we analyze Algorithm 1's running time.

▶ **Lemma 5.** *Algorithm 1 has a running time of $\mathcal{O}(|E|^2)$.*

Proof. In the first step, the algorithm constructs the reduced static expansion H of the temporal graph $\mathcal{G}^*$. The size of H is in $\mathcal{O}(|E|)$ and the construction can be done in time linear to its size. Constructing c and T can also be done in $\mathcal{O}(|E|)$ time.

Next, we need to compute the value of a maximum flow in H. Actually, it suffices to only test whether that value exceeds x (and we may assume $x \leq |E|$). This test is possible in $\mathcal{O}(|E|^2)$ time, for example by using the classic method of Ford & Fulkerson [13]. ◀

Finally, Lemmas 4 and 5 give us the following theorem.

▶ **Theorem 6.** *Delay-Robust Connection can be solved in $\mathcal{O}(|E|^2)$ time.*

4 Delayed-Routing Games

In this section, we analyze the problems Delayed-Routing Game and Delayed-Routing Path Game, which ask whether a *traveler* can reach their destination when an *adversary* can delay time arcs while the traveler is traversing them. More formally, the traveler and the adversary are players in a given game instance of the two-player game Delayed-Routing Game or Delayed-Routing Path Game, respectively, and we ask whether the traveler has a winning strategy. Starting at a start vertex s at time step 0, the traveler selects an out-going time arc from the current vertex, while the adversary can then delay a selected time arc by δ time units. However, the number of delays of the adversary is limited to x, thus they cannot always apply a delay. The traveler wins when they reach the target vertex z. In the Delayed-Routing Path Game the traveler can visit each vertex at most once, whereas no such restriction applies in the Delayed-Routing Game.

We present a dynamic program to solve Delayed-Routing Game in $O(|V| \cdot |E| \cdot x)$ time. We later use a slightly modified version of the algorithm to show that Delayed-Routing Path Game is in PSPACE. Furthermore, using a polynomial-time many-one reduction from QBF Game we prove that Delayed-Routing Path Game is PSPACE-hard. Hence, we can conclude that Delayed-Routing Path Game is PSPACE-complete.

4.1 A dynamic program for Delayed-Routing Game

A key observation for our dynamic program is that there are only polynomially many game states in Delayed-Routing Game. Furthermore, the available moves from a node of the game tree[1] and the determination of the winner only depend on the current node/game state and not on its predecessors. Hence, once we have computed whether the traveler has a winning strategy for any given game state, we can save this information in a dynamic programming table.

Let $I = (\mathcal{G} = (V, E), s, z \in V, \delta, x \in \mathbb{N})$ be an instance of Delayed-Routing Game. Starting at vertex s at time step 0 and with the adversary having a budget of x delays, the goal for the traveler is to reach the target vertex z. On the traveler's turn, they can select a time arc incident to the current vertex that occurs at the current time step or later. The adversary can then decide whether they delay this time arc by δ, thus reducing their number of remaining delays by 1. Once the current vertex is the target vertex z, the game ends with the traveler as the winner. If at any point there are no available time arcs, then the game ends with the adversary as the winner. If the game runs indefinitely, then the adversary also wins the game (since $\mathcal{G}$ is finite, this can only occur if the traveler follows a cycle with traversal time 0).

[1] The *game tree* consists of all possible game states linked by the successor relationship [25].

At the traveler's turn, the game state can be fully described by the 3-tuple (v, t, y), where $v \in V$ is the current vertex, $t \in \mathbb{N}$ is the current time step, and $y \in [0, x]$ is the number of remaining delays. We may take t to be from the set $T := \{1, \mathrm{t}(e)+\lambda(e), \mathrm{t}(e)+\lambda(e)+\delta \mid e \in E\}$. The starting game state is $(s, 1, x)$.

We define a dynamic programming table $F : V \times T \times [-1, x] \to \{\texttt{true}, \texttt{false}\}$ where $F(v, t, y) = \texttt{true}$ if the traveler has a winning strategy from the game state (v, t, y). Note that we allow the delay budget to reach -1 for technical reasons, but we will define $F(v, t, -1) = \texttt{true}$ for all $v \in V$, $t \in T$. This can be interpreted as allowing the adversary to "cheat" by exceeding their budget of delays, at the cost of immediately losing. Since this option is never beneficial for the adversary, providing it does not change the game in any significant way.

Denote by $E_t(v) := \{(v, w, t', \lambda) \in E \mid t' \geq t\}$ the set of all time arcs that are available at $v \in V$ at time t or later. Then we have the following.

▶ **Lemma 7.** *For all $v \in V \setminus \{z\}$, $t \in T$, and $y \in [0, x]$ it holds that*

$$F(z, t, y) = \texttt{true} \tag{1}$$

$$F(v, t, -1) = \texttt{true} \tag{2}$$

$$F(v, t, y) = \bigvee_{e \in E_t(v)} \big(F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e), y) \wedge F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e) + \delta, y - 1)\big) \tag{3}$$

where the empty disjunction evaluates to $\texttt{false}$*.*

Proof. Equation (1) is trivially correct, since the traveler has reached their destination vertex z. Equation (2) holds by definition as noted above.

It remains to prove (3). By the rules of the game, the traveler may choose any time arc $e \in E_t(v)$ when they are in game state (v, t, y). If the adversary opts to delay that time arc, then the resulting game state is $F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e) + \delta, y - 1)$. Otherwise, the resulting game state is $F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e), y)$. If, for some $e \in E_t(v)$, the traveler has winning strategies for both of these game states, then they can win from (v, t, y) by picking e and then proceeding from either of the two resulting game states according to their respective winning strategy. Conversely, if all of the time arcs in $E_t(v)$ lead to a game state from which the adversary has a winning strategy, then the traveler clearly cannot win. ◀

In principle, we would like to use Lemma 7 to compute all values of F. However, in the presence of arcs with traversal time zero, Lemma 7 might not suffice to completely determine all values of F. Consider the example instance given in Figure 4. For all $v \in V$, $t \in T$, and $y \in \{0, 1\}$ we clearly have

$$F(b, t, y) = [t \leq 2] \quad \text{and} \quad F(a, t, 0) = [t \leq 2]$$

by (3). Note that we can compute

$$F(a, 2, 0) = F(b, 2, 0) \vee F(s, 2, 0) = \texttt{true} \vee F(s, 2, 0) = \texttt{true}$$

without needing the value of $F(s, 2, 0)$. However, $F(s, 1, 1)$ and $F(a, 1, 1)$ depend on each other through (3):

$$\begin{aligned} F(s, 1, 1) &= \big(F(a, 1, 1) \wedge F(a, 2, 0)\big) \vee \big(F(a, 2, 1) \wedge F(a, 3, 0)\big) = F(a, 1, 1) \quad \text{and} \\ F(a, 1, 1) &= \big(F(s, 1, 1) \wedge F(s, 2, 0)\big) \vee \big(F(s, 2, 1) \wedge F(s, 3, 0)\big) \vee \big(F(b, 2, 1) \wedge F(b, 3, 0)\big) \\ &= F(s, 1, 1). \end{aligned}$$

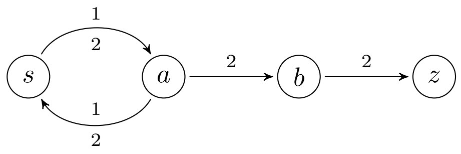


Figure 4 A Delayed-Routing Game instance in which the traveler cycles between s and a forever. All traversal times are 0 (omitted in the figure), and $\delta = x = 1$. Edges with multiple labels occur at multiple times.

If the value of a table entry is not determined through (1)–(3), then we call this value *hung*. As we have seen, it can occur that a subset of all table entries remains hung, even when all other entries have been computed. This is precisely the case when in each clause appearing in the disjunction (3) at least one of the two referenced entries is is either `false` or hung itself. In other words, from a hung state, the traveler only has the options to move to a losing state or to another hung state. In particular, the traveler can play on forever but never reach a winning state. Thus, in accordance with our rule that the adversary shall win if the game continues forever, we may set all hung table entries to `false` in this case.

We summarize this in the following lemma.

▶ **Lemma 8.** *If every value of F has either been computed or depends, through* (3)*, on another still uncomputed value, then the traveler does not have a winning strategy from any of the game states whose values are still uncomputed.*

Next, we determine the time required to fill the dynamic programming table.

▶ **Lemma 9.** *All entires of F can be computed in $O(|V| \cdot |E| \cdot x)$ time.*

Proof. The number of table entries is $N := |V| \cdot |T| \cdot (x+2) \in \mathcal{O}(|V| \cdot |E| \cdot x)$. For any $t \in T$, denote by t^+ the smallest element of T strictly larger than t. Begin by observing that (3) can be replaced with the following equivalent formula.

$$F(v,t,y) = F(v,t^+,y) \lor \bigvee_{e \in E_t(v) \setminus E_{t^+}(v)} \big(F(\text{end}(e), \text{t}(e)+\lambda(e), y) \land F(\text{end}(e), \text{t}(e)+\lambda(e)+\delta, y-1)\big) \tag{4}$$

We compute the table entries using Lemma 7 as follows. Start with the entries given by (1). Whenever an entry $F(v,t,y)$ is set to `true`, check whether any of the entries directly depending on $F(v,t,y)$ through (4) can be also computed (i.e., set to `true`, as (4) contains no negations). Note that there are three ways of how another entry $F(v',t',y')$ can directly depend on $F(v,t,y)$:
(i) $v = v'$, $y = y'$, and $t = t'^+$,
(ii) $y' = y$ and there is a time arc $(v', v, \hat{t}, \lambda)$ with $t' \le \hat{t} < t'^+$ and $\hat{t} + \lambda = t$, or
(iii) $y' = y + 1$ and there is a time arc $(v', v, \hat{t}, \lambda)$ with $t' \le \hat{t} < t'^+$ and $\hat{t} + \lambda + \delta = t$.
In particular, each time arc causes a direct dependency only between about $2x$ pairs of entries through (ii) and (iii). The number of dependencies through (i) is clearly at most N. Thus, the overall number of checks to be performed is at most $2x \cdot |E| + N$ and each of these checks can be done in constant time.

Afterwards, all remaining entries must be either `false` or hung: Since (4) contains no negations, setting entries to `false` can never cause other entries to become `true`. By Lemma 8, we can thus set all remaining entries to `false`. Hence, the entire table can be filled in $\mathcal{O}(x \cdot |E| + N) \subseteq \mathcal{O}(|V| \cdot |E| \cdot x)$ time. ◀

To solve Delayed-Routing Game, we can now evaluate F and check whether $F(s, 1, x) =$ `true`. Hence, Lemma 9 gives us the following.

▶ **Theorem 10.** *Delayed-Routing Game can be solved in $O(|V| \cdot |E| \cdot x)$ time.*

4.2 PSPACE-hardness of Delayed-Routing Path Game

We now present a polynomial-time reduction from the PSPACE-complete QBF Game to Delayed-Routing Path Game. QBF Game is a game formulation of the problem QBF that asks whether a given quantified boolean formula is true. In the game variant, Player 1 and Player 2 choose truth assignments for existentially and universally quantified variables, respectively. Player 1 wins when the formula is satisfied, otherwise Player 2 wins. If Player 1 has a winning strategy, then it is a yes-instance. QBF and QBF Game are equivalent and known to be PSPACE-complete [1].

In QBF Game, we are given a quantified boolean formula $\Phi = Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . \varphi$ with $Q_i \in \{\exists, \forall\}$ and φ being a boolean formula. The game then consists of n rounds. In the i-th round, if $Q_1 = \exists$, then Player 1 selects a truth value for x_i. Else if $Q_1 = \forall$, then Player 2 selects a truth value for x_i. If after the n-th round φ is satisfied under the selected truth assignment, then Player 1 wins, otherwise Player 2 is the winner.

QBF Game

Input: A quantified boolean formula $\Phi = Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . \varphi$ with $Q_i \in \{\exists, \forall\}$.

Question: Is there a winning strategy for Player 1?

Given a QBF Game-instance $\Phi = Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . \varphi$, we construct an instance $(\mathcal{G} = (V, E), s, z \in V, x, \delta \in \mathbb{N})$ of Delayed-Routing Path Game, so that Player 1 has a winning strategy for Φ if and only if the traveler has a winning strategy for the Delayed-Routing Path Game-instance. Without loss of generality, we assume that φ is in conjunctive normal form and has three literals per clause.

The main idea for our reduction is the following. The temporal graph $\mathcal{G}$ we create in our reduction consists of n chained selection gadgets, one for each quantified variable, and $m - 1$ validation gadgets where m is the number of clauses of φ. In a selection gadget for a variable quantified by an existential quantifier, the traveler can freely choose one of two paths, corresponding to a truth assignment of this variable. A delay by the adversary has no effect in this gadget. In a selection gadget for a variable quantified by a universal quantifier, a delay of the adversary forces the traveler to take a specific path, corresponding to a truth assignment of this variable. If not taking this enforced path, the traveler gets immediately stuck and loses the game. The gadget is constructed in a way that the adversary needs to use exactly one delay per universal quantified variable. Using no delay lets the traveler immediately win, while using more than one delay is no better for the adversary than a single delay.

In the validation gadgets, the adversary can force the traveler to take one of two paths, one corresponding to selecting the corresponding clause of φ, the other leading to the next validation gadget. In this way, the adversary can select one clause of φ. After traversing either all validation gadgets or selecting a specific clause, the adversary has no remaining delays.

Finally, if there is a literal in the clause which is satisfied under the truth assignment corresponding to the path taken in the selection gadgets, then the traveler can traverse back to a vertex in the selection gadget. From this vertex, the traveler can then reach the target vertex. Otherwise, if all literals in the clause are unsatisfied, then all vertices reachable with a time arc have already been visited when traversing the selection gadgets. Thus the traveler becomes stuck.

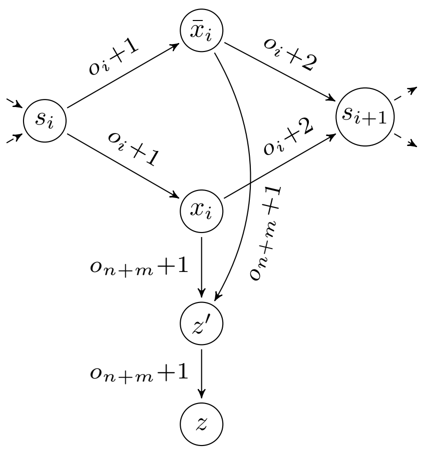


Figure 5 Selection gadget for the existentially quantified variable x_i. The traveler can choose freely whether the upper or lower path to s_{i+1} is taken. The adversary has no incentive to apply any delays. Dwelling in x_i or $\bar{x}_i$ to get to z' is no option for the traveler as long as there are delays left.

Now we describe the reduction more formally. All time arcs in the constructed temporal graph $\mathcal{G}$ have a traversal time of 0, thus we write time arcs as 3-tuples $(v, w, t) \in E$ and omit the traversal time in figures. We set the number of delays $x := n' + m - 1$, where $n' \leq n$ is the number of universal quantifiers in Φ. Furthermore, we set $\delta := 1$. The start and target vertices of the game are s_1 and z, respectively, which are added during the construction of the temporal graph. The gadgets use an offset o_i, starting with $o_1 = 0$. The other offsets are computed while constructing the gadgets. Initially, we add the vertices $s_1, s_2, \dots, s_{n+1}$, z', and z to V, and we add the time arc

$$z' \xrightarrow{o_{n+m}+1} z.$$

The time step $o_{n+m} + 1$ is the largest time step of the constructed temporal graph.

Selection Gadgets

The selection gadget is used to assign a truth value to the quantified variable $Q_i x_i$. The gadget depends on the type of quantifier:

Case 1. $Q_i x_i = \exists x_i$; the variable x_i is existentially quantified.

We add the vertices x_i and $\bar{x}_i$ to V. Furthermore, we add the following time arcs

$$s_i \xrightarrow{o_i+1} x_i \xrightarrow{o_i+2} s_{i+1} \quad \text{and} \quad s_i \xrightarrow{o_i+1} \bar{x}_i \xrightarrow{o_i+2} s_{i+1}.$$

corresponding to assigning x_i to true and false, respectively. The traveler can choose whether to reach s_{i+1} over the vertex x_i or $\bar{x}_i$. A delay of the adversary will have no effect.

Additionally, we add the time arcs

$$x_i \xrightarrow{o_{m+n}+1} z' \quad \text{and} \quad \bar{x}_i \xrightarrow{o_{m+n}+1} z';$$

however, when traversing this gadget, if the traveler dwells in x_i or $\bar{x}_i$ to take the time arc to z', then the adversary can use a delay that makes the only outgoing time arc to z at time step $o_{m+n} + 1$ unavailable.

We set the offset $o_{i+1} := o_i + 2$. An example of a selection gadget for existentially quantified variables can be seen in Figure 5.

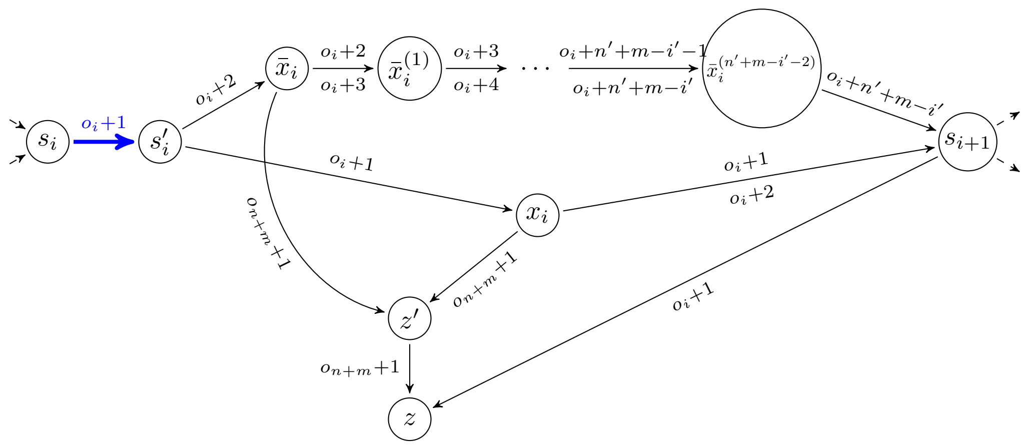


Figure 6 Selection gadget for the universally quantified variable x_i. Let i' be the number of universally quantified variables before the i-th variable. The traveler enters with $n' + m - i' - 1$ delays left. If the adversary delays the first (thick, blue) time arc, only the upper path to s_{i+1} is available for the traveler. The remaining $n' + m - i' - 2$ delays are not enough to make the traveler stuck, and the adversary has no incentive to use any further delays. If the adversary does not delay the first (thick, blue) time arc, then the traveler is forced to take the lower path to s_{i+1}, since if the adversary applies all $n' + m - i' - 1$ delays on the upper time arcs, then the traveler gets stuck. The adversary will have to apply one delay on the lower time arcs, otherwise the traveler can take the time arc from s_{i+1} to z.

Case 2. $Q_i x_i = \forall x_i$; the variable x_i is universally quantified.

Let i' be the number of universally quantified variables before the i-th variable. We add the vertices $s'_i, x_i, \bar{x}_i$, and $\bar{x}_i^{(1)}, \bar{x}_i^{(2)}, \dots, \bar{x}_i^{(n'+m-i'-2)}$ to V. Furthermore, we add a time arc $s_i \xrightarrow{o_i+1} s'_i$, and the three time arcs

$$s'_i \xrightarrow{o_i+1} x_i \xrightarrow[o_i+2]{o_i+1} s_{i+1},$$

corresponding to setting x_i to true, and

$$s_i \xrightarrow{o_i+2} \bar{x}_i \xrightarrow[o_i+3]{o_i+2} \bar{x}_i^{(1)} \xrightarrow[o_i+4]{o_i+3} \bar{x}_i^{(2)} \dots \xrightarrow[o_i+n'+m-i']{o_i+n'+m-i'-1} \bar{x}_i^{(n'+m-i'-2)} \xrightarrow{o_i+n'+m-i'} s_{i+1},$$

corresponding to setting x_i to false. Finally, we add a time arc $s_{i+1} \xrightarrow{o_i+1} z$ directly to the end vertex z.

By not delaying the time arc $s_i \xrightarrow{o_i+1} s'_i$, the adversary forces the traveler to take the path through vertex x_i, since the path $s_i \to \bar{x}_i \to \bar{x}_i^{(1)} \to \dots \to \bar{x}_i^{(n'+m-i'-2)} \to s_{i+1}$ can be broken by applying all remaining $n' + m - i' - 1$ delays. The adversary is still enforced to apply one delay in $s'_i \to x_i \to s_{i+1}$, otherwise the traveler can take $s_{i+1} \xrightarrow{o_i+1} z$ and directly wins. By delaying the time arc $s_i \xrightarrow{o_i+1} s'_i$, the adversary forces the traveler to take the path through vertex $\bar{x}_i$, since the time arc $s_i \xrightarrow{o_i+1} x_i$ becomes unavailable. However, the remaining $n' + m - i'$ delays are not enough to break the path $s_i \to \bar{x}_i \to \bar{x}_i^{(1)} \to \bar{x}_i^{(2)} \dots \bar{x}_i^{(n'+m-i')} \to s_{i+1}$.

Additionally, we add the time arcs

$$x_i \xrightarrow{o_{m+n}+1} z' \quad \text{and} \quad \bar{x}_i \xrightarrow{o_{m+n}+1} z',$$

however when traversing this gadget, if the traveler dwells in x_i or $\bar{x}_i$ to take the time arc to z', then the adversary can use a delay that makes the only outgoing time arc to z at time step $o_{m+n} + 1$ unavailable.

We set the offset $o_{i+1} := o_i + n' + m - i'$. An example of a selection gadget for universally quantified variables can be seen in Figure 6.

Validation Gadgets

The validation gadgets are used to check whether the formula φ is satisfied for the truth assignment chosen in the selection gadgets. By using all $m-1$ remaining delays, the adversary can force the traveler to visit a vertex corresponding to a specific clause of φ. For the clauses $c_1, c_2, \dots, c_{m-1}$ there is a validation gadget. By placing a single delay, the adversary can force the traveler to take one of two junctions, where one corresponds to selecting the clause, and the other leads to the next validation gadget. (For the $m-1$-st validation gadget the other junction corresponds to the m-th clause.) From there, the traveler can reach z only if there is a satisfied literal in the clause.

For each clause $c_i \in \{c_1, c_2, \dots, c_{m-1}\}$, we add the vertices v_i, $v_i^{(l,k)}$ for $k \in [m-i]$, and $v_i^{(r,k)}$ for $k \in [m-i]$. For $i \in [2, m-1]$, we add the time arc

$$v_{i-1}^{(r,m-(i-1))} \xrightarrow{o_{n+i}+1} v_i,$$

connecting the i-th validation gadget with the $(i-1)$-st validation gadget. For $i = 1$, we add the time arc

$$s_{n+1} \xrightarrow{o_{n+1}+1} v_i,$$

connecting the last selection gadget with the first validation gadget.

For the branch corresponding to selecting the i-th clause, we add the time arcs

$$v_i \xrightarrow{o_{n+i}+1} v_i^{(l,1)} \xrightarrow{o_{n+i}+2} v_i^{(l,2)} \xrightarrow{o_{n+i}+3} \dots \xrightarrow{o_{n+i}+m-i} v_i^{(l,m-i)}.$$

Furthermore, for all $v_i^{(l,k)}$ with $k \in [m-i]$, we add a time arc

$$v_i^{(l,k)} \xrightarrow{o_{n+i}+k} z,$$

which enforces the adversary to place delays on all time arcs above, so the traveler cannot directly go to vertex z and win the game. This ensures that all delays are used after reaching $v_i^{(l,m-i)}$, which corresponds to selecting the i-th clause.

For the branch corresponding to not selecting the i-th clause, we add the time arc

$$v_i \xrightarrow{o_{n+i}+2} v_i^{(r,1)},$$

the time arcs

$$v_i^{(r,k)} \xrightarrow[o_{n+i}+k+2]{o_{n+i}+k+1} v_i^{(r,k+1)},$$

for $k \in [m-i-1]$, and the time arc

$$v_i^{(r,m-i)} \xrightarrow{o_{n+i}+m-i+1} v_i^{(r,m-i+1)}.$$

Delaying all $m-i$ time arcs from v_i to $v_i^{(r,m-i)}$ will break the connection from $v_i^{(r,m-i)} \to v_i^{(r,m-i+1)}$; however, if there is one delay less, then the adversary does not get any better by using delays. We set the next offset $o_{n+i+1} := o_{n+i} + m - i + 2$.

Finally, we add time arcs for the literals in the clauses. Let the i-th clause be

$$(l_i^{(1)} \vee l_i^{(2)} \vee l_i^{(3)}).$$

For all $i \in [m-1]$ and all $j \in [3]$, if $l_i^{(j)} = x_k$, then we add the time arc

$$v_i^{(l,m-i)} \xrightarrow{o_{n+m}+1} \bar{x}_k,$$

and if $l_i^{(j)} = \bar{x}_k$, then we add the time arc

$$v_i^{(l,m-i)} \xrightarrow{o_{n+m}+1} x_k,$$

where $k \in [n]$. For the last clause c_m and for all $j \in [3]$, if $l_i^{(j)} = x_k$, we add the time arc

$$v_{m-1}^{(r,2)} \xrightarrow{o_{n+m}+1} \bar{x}_k,$$

and if $l_i^{(j)} = \bar{x}_k$, then we add the time arc

$$v_{m-1}^{(r,2)} \xrightarrow{o_{n+m}+1} x_k,$$

where $k \in [n]$. At this point there are no delays remaining, and for all $k \in [n]$, the time arcs

$$x_k \xrightarrow{o_{n+m}+1} z' \xrightarrow{o_{n+m}+1} z \quad \text{or} \quad \bar{x}_k \xrightarrow{o_{n+m}+1} z' \xrightarrow{o_{n+m}+1} z,$$

which have been added previously in the selection gadgets, can be traversed.

Due to space constraints, we defer a visualization of the validation gadget and the remaining details of the proof which leads to the following theorem to the full version.

▶ **Theorem 11.** *Delayed-Routing Path Game is* PSPACE*-hard.*

4.3 PSPACE-containment of Delayed-Routing Path Game

Complementing the PSPACE-hardness of Delayed-Routing Path Game from the previous section, we now show that Delayed-Routing Path Game is containted in PSPACE, which lets us conclude PSPACE-completeness of Delayed-Routing Path Game.

We show this by modifying the dynamic program for Delayed-Routing Game (Section 4.1) to also save the set of vertices that already has been visited for every state. This will cause the dynamic programming table to have exponential size; however, we can evaluate it recursively, that is, recomputing every entry when needed. In this way we only require polynomial space. Formally, we adapt the recursive formula as follows.

Let $I = (G = (V, E), s, z \in V, \delta, x \in \mathbb{N})$ be an instance of Delayed-Routing Path Game. A game state can be fully described by the 4-tuple (v, t, y, V'), where $v \in V$ is the current vertex, $t \in \mathbb{N}$ is the current time step, $y \in [0, x]$ is the number of remaining delays, and $V' \subseteq V$ the set of visited vertices. We may take t to be from the set $T := \{1, \mathrm{t}(e) + \lambda(e), \mathrm{t}(e) + \lambda(e) + \delta \mid e \in E\}$. The starting game state is $(s, 1, x, \emptyset)$.

We define $F : V \times T \times [-1, x] \times 2^V \to \{\texttt{true}, \texttt{false}\}$ to indicates for each game state whether the traveler has a winning strategy from that game state. Denote by $E_t(v) := \{(v, w, t', \lambda) \in E \mid t' \geq t\}$ the set of all time arcs that are available at $v \in V$ at time t or later. Then, for all $v \in V \setminus \{z\}$, $t \in T$, $y \in [0, x]$, and $V' \subseteq V \setminus \{z\}$ the following holds:

$$F(z, t, y, V') = \texttt{true}, \tag{5}$$
$$F(v, t, -1, V') = \texttt{true}, \tag{6}$$
$$F(v, t, y, V') = \bigvee_{e \in E_t(v)} \big(\mathrm{end}(e) \notin V' \wedge F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e), y, V' \cup \{v\}) \wedge F(\mathrm{end}(e), \mathrm{t}(e) + \lambda(e) + \delta, y - 1, V' \cup \{v\})\big), \tag{7}$$

where the empty disjunction evaluates to `false`.

Using (5)–(7), we get the following result by evaluating it in a depth-first-search fashion from the starting configuration.

▶ **Proposition 12.** *Delayed-Routing Path Game is contained in* PSPACE.

Proof. The correctness of our approach can be shown in a way analogous to Lemma 7. Note that since the set V' of visited vertices is growing with each move of the traveler, the game cannot run infinitely. Thus, all entries can be computed by means of (5)–(7).

Instead of storing all (exponentially many) entries, we evaluate F in a depth-first-search fashion from the starting configuration $(s, 1, x, \emptyset)$. This only requires us to keep the current branch of the search tree in memory. Since each game state requires polynomial space and there are at most $O(|V|)$ moves in a game of Delayed-Routing Path Game (every vertex can be visited at most once), we only require polynomial space. ◀

From Theorem 11 and Proposition 12 we can now conclude that Delayed-Routing Path Game is PSPACE-complete.

▶ **Corollary 13.** *Delayed-Routing Path Game is* PSPACE*-complete.*

5 Conclusion and Outlook

On the spectrum of delay-related routing problems, we have studied two extreme (but natural) cases in terms of when information about the delays is made available. Interestingly, both are polynomial-time solvable, whereas a “middle ground” case studied in companion work turned out NP-hard.

It might also seem surprising that Delayed-Routing Game is efficiently solvable while Delayed-Routing Path Game is PSPACE-complete. However, this situation is not unprecedented. For example, deciding whether a temporal path under waiting time constraints exists (Δ-Restless Temporal Path) is NP-complete [6], while finding temporal walks under waiting time constraints can be done in polynomial time [2]. Similarly, counting foremost temporal paths is #P-hard [23], while counting of foremost temporal walks can be done in polynomial time [24].

We remark that instead of delaying edges by increasing their traversal time, it is also sensible to instead delay their time label. It can be shown that our results on Delay-Robust Connection transfer also to this modified version. For Delayed-Routing Game the situation is more complicated, we leave this open for future work.

Even more different notions of delays could also be explored. While in our definitions up to $x \in \mathbb{N}$ time arcs can be delayed by a fixed integer δ each, one could also define an overall “budget” Δ which can be distributed among all time arcs. Thus, a time arc could be delayed by more than δ or more than x time arcs could be delayed by less than δ each.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

2 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020. doi:10.1007/s41109-020-00311-0.

3 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003. doi:10.1142/S0129054103001728.

4 Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. Algorithmic aspects of temporal betweenness. In *Proceedings of the 26th SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2084–2092, 2020. doi:10.1145/3394486.3403259.

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

6 Arnaud Casteigts, 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.

7 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. In *Proceedings of the 34th Conference on Artificial Intelligence (AAAI)*, pages 9810–9817, 2020. doi:10.1609/aaai.v34i06.6533.

8 Peter Elias, Amiel Feinstein, and Claude E. Shannon. A note on the maximum flow through a network. *IRE Transactions on Information Theory*, 2(4):117–119, 1956. doi:10.1109/TIT.1956.1056816.

9 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/j.jcss.2021.01.007.

10 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021. doi:10.1016/j.jcss.2020.08.001.

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

12 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020. doi:10.1016/j.tcs.2019.03.031.

13 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

14 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, 2022. doi:10.4230/LIPIcs.STACS.2022.30.

15 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Temporal connectivity: Coping with foreseen and unforeseen delays, 2022. arXiv:2201.05011.

16 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015. doi:10.1140/epjb/e2015-60657-4.

17 Petter Holme and Jari Saramäki, editors. *Temporal Network Theory*. Springer, 2019. doi:10.1007/978-3-030-23495-9.

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

19 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4090–4096, 2021. doi:10.24963/ijcai.2021/563.

20 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1):61, 2018. doi:10.1007/s13278-018-0537-7.

21 George B Mertzios, Othon Michail, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/s00453-018-0478-6.

22 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 76:1–76:15, 2021. doi:10.4230/LIPIcs.MFCS.2021.76.

23 Amir Afrasiabi Rad, Paola Flocchini, and Joanne Gaudet. Computation and analysis of temporal betweenness in a knowledge mobilization network. *Computational Social Networks*, 4(1):1–22, 2017. doi:10.1186/s40649-017-0041-7.

24 Maciej Rymar, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Towards classifying the polynomial-time solvability of temporal betweenness centrality. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 219–231, 2021. doi:10.1007/978-3-030-86838-3_17.

25 Aaron N. Siegel. *Combinatorial game theory*. American Mathematical Society, 2013.

26 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016. doi:10.1109/TKDE.2016.2594065.

27 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020. doi:10.1016/j.jcss.2019.07.006.