Network Navigation with Online Delays is PSPACE-complete

Thomas Depian[1] Christoph Kern[1] Sebastian Röder[1] Soeren Terziadis[2] Markus Wallinger[2]

Abstract: In public transport networks disruptions may occur and lead to travel delays. It is thus interesting to determine whether a traveler can be resilient to delays that occur unexpectedly, ensuring that they can reach their destination in time regardless. We model this as a game between the traveler and a delay-introducing adversary. We study the computational complexity of the problem of deciding whether the traveler has a winning strategy in this game. Our main result is that this problem is PSPACE-complete.

Keywords: temporal paths; network navigation; robust connections

1 Introduction

According to Destatis, the total distance traveled by individuals in Germany using public transport in 2022 amounted to 99 billion kilometers [22]. Finding the best public transport route between a starting point and a destination is a well-researched topic and there are well-known algorithms and datastructures for computing such routes [Ba16]. Once the route has been determined, however, the traveler may encounter additional challenges. According to rail company Deutsche Bahn, more than 13 % of the stops of their long-distance trains were not within 15 minutes of the schedule in April 2023 [23]. Travelers thus regularly need to board delayed trains, perhaps causing them to miss connecting trains later on. Therefore, it is an interesting problem to find out whether a traveler can be resilient to such delays. The problem formulation we are interested in here models the following question: Starting from point s, is it possible to reach point z in time t where a delay of a connecting train may occur unexpectedly at any changeover?

The above question can be modeled as a game between a traveler and a public transport company. In each round of the game, the traveler arrives at some station in the public transport network. Then the public transport company announces delays of the connections. The traveler then decides on which connection to take next and so on, until either the traveler reaches the destination z or the time is up. To model realistic scenarios we impose a delay-budget constraint on the public transport company, that is, the announced delays may sum up to at most some fixed budget. Whether the traveler is resilient to delays is then equivalent to whether they have a winning strategy, that is, whether it is possible to reach the destination in time regardless of which delays are being announced in each round. We call the resulting decision problem Robust Connection Game.

[1] [e11807882|e11904675|sebastian.roeder]@student.tuwien.ac.at, TU Wien, Austria
[2] [sterziadis|mwallinger]@ac.tuwien.ac.at, Algorithms & Complexity Group, TU Wien, Austria

doi:10.18420/SKILL2023_16

For very important appointments it may be useful to check beforehand whether they can be reached even with potential delays. That is, we want to decide Robust Connection Game computationally. Thus it is interesting to know its computational complexity. That is, how complex is it to decide, given the schedule of the network, the start and destination as well as the arrival time, whether the traveler has a winning strategy?

We show that Robust Connection Game can be solved with space bounded polynomially in the input length, that is, Robust Connection Game is contained in PSPACE (see Sect. 3). This in particular implies that it can be solved in time exponential in the input length. However, we also prove that Robust Connection Game is PSPACE-hard, that is, every problem in PSPACE can be reduced to Robust Connection Game in polynomial time (see Sect. 4). This makes it unlikely that the problem can be solved efficiently in general. While a negative result, our reduction highlights several features of the networks that we exploit in showing hardness, see the conclusion in Sect. 5. It may be worthwhile to check to which extent these features occur in real-world networks and, if not, whether their absence can be exploited to obtain efficient algorithms. Due to space constraints, we defer proofs for results marked by ★ to a full version of the paper, see Ref. [De23].

Related work. We model Robust Connection Game on so-called temporal graphs, that is, graphs in which edges are equipped with time information such as their starting time and traversal time. Routing on temporal graphs was to our knowledge first explored by Berman [Be96] and has in recent years gained considerable attention. Robustness of temporal connectivity was herein mostly studied with respect to deletion of a bounded number of time arcs or vertices, see, e. g., the overview by Füchsle et al. [Fü22a]. In terms of delays in our context, we are aware of two works:

First, Füchsle et al. [Fü22a] study the problem of finding one route that is robust to a bounded number of unit delays, regardless of when they occur. That is, checking whether there *exists* a route that is feasible *for all* delays. In this model, the authors assume that the traveler never reconsiders the rest of the route. The traveler thus potentially foregoes better connections that open up after they experienced some delays already. The authors then study the complexity of finding such routes and whether efficient algorithms can be obtained if the number of delays or the network topology is restricted.

In the second work, Füchsle et al. [Fü22b] study finding *for all* possible delays whether there *exists* a delay-tolerant route. In particular, they study the case where the delays may occur unexpectedly and model the resulting problem as a game similar to what we do here. However, there is a crucial difference: In Füchsle et al.'s model a delay of a connection may be announced only *during* the time in which the traveler takes the connection. In contrast, in our model the delays are announced *before* the traveler decides on the next connection to take. We would argue that both models are relevant and thus we close a gap in the literature. It also turns out that this seemingly small difference has an immense effect on the computational complexity: In Füchsle et al.'s game model, deciding whether there is a winning strategy is polynomial-time solvable and only becomes PSPACE-hard if we require the route taken by the traveler to be a path (that is, no vertex is traversed twice). In contrast, in our model the problem is PSPACE-hard without additional requirements on the route.

2 Preliminaries

In this paper, we work with temporal graphs, which build on top of static graphs. A *static graph* is a (directed) graph $G_s = (V_s, A_s)$ consisting of a set of *vertices* V_s and *arcs* $A_s \subseteq V_s \times V_s$. We denote an *arc* a as a tuple $a = (u, v)$, and call $\text{tail}(a) = u$ the *tail* vertex, and $\text{head}(a) = v$ the *head* vertex of a. The graph G_s contains *multi-arcs*, if there are two distinct arcs $a = (u, v), a' = (u, v) \in A_s$ that have the same vertices u and v as their head and tail. We allow multi-arcs but we prohibit self-loops, i. e., arcs of the form (u, u) for $u \in G_s$. A *walk* $W = (v_1, \ldots, v_k)$ in G_s is a sequence of k, not necessarily pairwise distinct, vertices such that we have $(v_i, v_{i+1}) \in A_s$ for $1 \leq i < k$. A walk is a *path* if the vertices are pairwise distinct.

Temporal Graphs. A *temporal graph* $G = (V, E)$ is a static directed graph where we replace arcs with temporal arcs. We denote a *temporal arc* e as a tuple $e = (u, v, t, \lambda)$, where $\text{tail}(e) = u$ and $\text{head}(e) = v$ denote the tail and head vertex, respectively, $t(e) = t$ denotes the *time label*, and $\lambda(e) = \lambda$ denotes the *traversal time* of e. The time label is a point in time, after which e is unavailable. The traversal time is the time span it takes to travel along the temporal arc e. If a temporal arc e is *delayed* by δ, its time label increases by δ. A temporal graph G is (D, δ)-*delayed*, for $D \subseteq E$, denoted as $G_{(D,\delta)}$, if the temporal arcs in D are delayed by δ. Hence, $G_{(D,\delta)}$ is the (D, δ)-delayed temporal graph version of the temporal graph G, where we replace each temporal arc $e \in D$ with $e' = (\text{tail}(e), \text{head}(e), t(e) + \delta, \lambda(e))$. In the literature this type of delay is usually denoted as *starting delay* [Fü22a]. For an arc e in a temporal graph G we denote by $\text{arr}_G(e) = t(e) + \lambda(e)$ the *arrival time* of e (at vertex $\text{head}(e)$) in the temporal graph G, i. e., $t(e)$ and $\lambda(e)$ are w.r.t. the temporal graph G. We omit G and just write $\text{arr}(e)$ if G is clear from context. A *temporal walk* $W = (v_1, \ldots, v_k)$ in G is a walk where we require in addition that for each temporal arc $e_i = (v_i, v_{i+1}, \cdot, \cdot)$, $1 \leq i < k$, in W we have $\text{arr}(e_i) \leq t(e_{i+1})$. Similar to the static version, we define a *temporal path* to be a temporal walk with pairwise distinct vertices.

Throughout this paper, we assume $t(e) > 0$ and $\lambda(e) > 0$ for all temporal arcs e and write t and λ if e is clear from context. Furthermore, we drop the prefix "temporal" if there is no risk of confusion. Unless otherwise stated, we denote with n the number of vertices and with m the number of arcs of the graph G, i. e., $n = |V|$ and $m = |E|$.

2.1 The Robust Connection Game

We now define the *robust connection game*. This is a round-based game between a *traveler* and an *adversary*. Our goal is to model an online planning scenario, where delays are decided on by the adversary only on arrival of the traveler at a vertex and the traveler may reconsider his next steps. An instance of robust connection game is given by a tuple (G, s, z, x, δ) where $G = (V, E)$ is a graph, $s, z \in V$, and x and δ are positive integers. The meaning is as follows:

The traveler starts at the vertex s at timestamp 1 and wants to reach the vertex z in finitely many rounds. Initially, we have $D = \emptyset$ and the adversary has a budget of x. In each round, the traveler is located at a vertex $u \in G_{(D,\delta)}$ at some timestamp t and the adversary first announces the delay of a subset D' of the arcs. The delayed arcs are restricted to those arcs that have u as its tail and a time label t' with $t \leq t'$ (arcs with $t > t'$ could be allowed, however the adversary does not gain an advantage from delaying such arcs). Furthermore, the number of arcs that can be announced as delayed is limited to the remaining budget $x - |D|$ of the adversary. The budget of the adversary is decreased after each round by the number of arcs that are announced as delayed, i. e., by $|D'|$. Once the delays have been announced, a delay of δ time steps is applied to the (newly) delayed arcs, that is, we compute $G_{(D \cup D', \delta)}$. Afterwards, the traveler has to move to a next vertex $v \neq u$, either through a delayed or not delayed arc e, setting the current time t to $\text{arr}_{G_{(D \cup D', \delta)}}(e)$. Once the traveler is at v, the next round begins, i. e., the adversary can again announce the delay of a set D' of arcs.

Each arc can be delayed at most once, i. e., once an arc has been announced as delayed and the delay was applied, it can never be re-delayed again. The game ends if the traveler either reaches z, or if the traveler is stuck at a vertex $u \neq z$, that is, they reach u at a time t where there is no further arc e, s. t., $t \leq t(e)$. In the former case, the traveler wins the game, in the later case the adversary wins. The traveler has a *winning strategy*, if the vertex z can always be reached independent of the announced delays. We define the corresponding decision problem as follows.

Robust Connection Game (RCG)
Input: A temporal graph $G = (V, E)$, two vertices $s, z \in V$, and two positive integers x, δ, with $x \leq |E|$.
Question: Does the traveler have a winning strategy in the robust connection game (G, s, z, x, δ)?

3 Solving Robust Connection Game in Exponential-Time and Polynomial-Space

Let $I = (G, s, z, x, \delta)$ be an instance of Robust Connection Game (RCG). We provide in this section an exponential-time dynamic programming (DP) algorithm to check whether the traveler has a winning strategy in I. Füchsle et al. [Fü22b] presented a polynomial-time DP-algorithm for the related Delayed-Routing Game. While our result follows a similar structure, the Robust Connection Game does not possess polynomially many game states, which will also reflect in the running time.

For an instance I of Robust Connection Game as above we describe a state of the game with the tuple (v, t, D), where $v \in V$ is the vertex the traveler is currently at, $t \in \mathbb{N}$ is the current timestamp, and $D \subseteq E$ denotes the set of delayed arcs. The remaining budget of the adversary equals to $y := x - |D|$. Observe that we do not have to consider all possible points in time but can restrict our attention to the set $T := \{1, \mathrm{arr}(e), \mathrm{arr}(e) + \delta \mid e \in E)\}$ of all (delayed) arrival times at vertices. In the following, we make use of the concept of (delayed) arcs available at a vertex $u \in V$ at the timestamp $t \in T$. For the set of arcs E in a (D, δ)-delayed graph $G_{(D,\delta)}$, they are defined as $E^t_{G_{(D,\delta)}}(u) := \{e \in E \mid \mathrm{tail}(e) = u, t \leq t(e)\}$. Since δ is a fixed value, we simply write E^t_D.

To solve instance I, for each of the possible game states (v, t, D), we denote in a table F whether the traveler has a winning strategy (`true`), or not (`false`). Lemma 3.1 describes how the states of our game depend on each other.

Lemma 3.1. *Assuming that the empty disjunction evaluates to* `false`*, we have the following equivalences for all $v \in V \setminus \{z\}$, $t \in T$, and $X \in \{D \mid D \subseteq E, |D| \leq x\}$.*

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

$$F(v, t, D) = \bigwedge_{\substack{D' \subseteq E^t_D(v) \setminus D \\ s.t.\ |D'| + |D| \leq x}} \quad \bigvee_{e \in E^t_{D \cup D'}(v)} F\left(head(e), arr_{G_{(D \cup D', \delta)}}(e), D \cup D'\right) \tag{2}$$

Proof. We first observe in Eq. (1) that the traveler is at the destination vertex z, i. e., the game is over and the traveler wins the game. Hence, Eq. (1) is trivially correct. We proceed with showing the correctness of Eq. (2). To do that, we show both directions explicitly and build our arguments on the definition of the game given in Sect. 2.1.

Assume that the traveler is at timestamp t at vertex $v \neq z$ and the arcs $D \subseteq E$ have already been delayed by the adversary. Therefore, we are in the game state (v, t, D). Furthermore, assume that the traveler has a winning strategy in this state, i. e., $F(v, t, D)$ is `true`. This means that no matter which additional arcs the adversary decides to delay, the traveler can reach z. So assume that the adversary announces the delay of the arcs in D', which by our definition must have its tail at v. A winning strategy at (v, t, D) consists, by the definition of Robust Connection Game, of using one outgoing arc e of v, available at t, after being aware of the additionally delayed arcs. However, observe that if the winning strategy at (v, t, D) consists of moving in the presence of the additional delays D' to the vertex $u = \mathrm{head}(e)$, for some arc e available at v at t, then the traveler must have a winning strategy in the state $\left(\mathrm{head}(e), \mathrm{arr}_{G_{(D \cup D', \delta)}}(e), D \cup D'\right)$. That is, $F\left(u, \mathrm{arr}_{G_{(D \cup D', \delta)}}(e), D \cup D'\right)$ must have been `true` as well. Since the set of announced delayed arcs D' was chosen arbitrarily w.r.t. the constraints enforced by the game, the right side of Eq. (2) therefore correctly evaluates to true. The other direction is deferred to a full version. □

The game starts in the state $(s, 1, \emptyset)$ and, as a consequence of Lemma 3.1, the traveler has a winning strategy iff $F(s, 1, \emptyset) = \texttt{true}$. In the following, we derive the time required to compute $F(s, 1, \emptyset)$ as well as the space consumption of our approach.

Lemma 3.2 (★). *Our approach solves RCG in* $O(n \cdot m^2 \cdot (m+1)^{2x})$ *time.*

Lemma 3.3. *Our approach for RCG can be implemented to use* $O(x \cdot m)$ *space.*

Proof. As we show in the proof of Lemma 3.2, there are $O(n \cdot m \cdot (m+1)^x)$ possible game states. Naïvely enumerating all possible game states would thus require an exponential amount of space. To circumvent this, we first observe that, by Eq. (2), evaluating whether the traveler has a winning strategy in the initial game state $(s, 1, \emptyset)$ results in a search tree $\mathcal{T}$, in which we enumerate in every odd level of $\mathcal{T}$ all possible subsets D' of delayed arcs, and in every even level of $\mathcal{T}$ the possible arcs the traveler can use. If we now use depth first search (DFS) on $\mathcal{T}$ to compute $F(s, 1, \emptyset)$, we only have to store the states of a single path from the root of $\mathcal{T}$ to a leave of $\mathcal{T}$ at a time. If we fix the order in which we enumerate at each internal node of $\mathcal{T}$ its children, the space requirement for DFS is linear in the depth of $\mathcal{T}$. We conclude the proof with observing that the depth of $\mathcal{T}$ is $O(|T|) = O(m)$, since we increase at every other level in $\mathcal{T}$ the timestamp. Each game state requires $O(x)$ space, since we need to store, in the worst case, that many delayed arcs. □

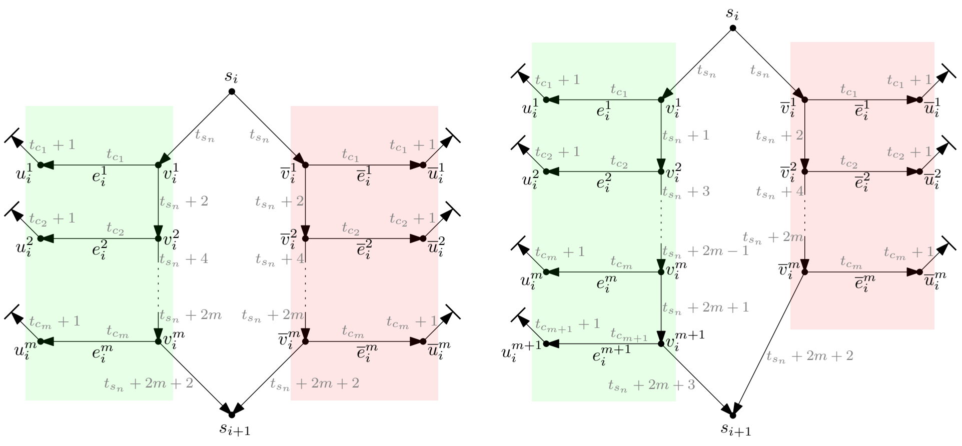
Using Lemmas 3.2 and 3.3 we summarize the main result of this section in Theorem 3.4.

Theorem 3.4. *Let* $I = (G = (V, E), s, z, x, \delta)$ *be an instance of* Robust Connection Game *with* $n = |V|$ *and* $m = |E|$*. We can solve* I *in* $O(n \cdot m^2 \cdot (m+1)^{2x})$ *time using* $O(x \cdot m)$ *space. Therefore,* Robust Connection Game *is in* PSPACE.

4 Robust Connection Game is PSPACE-hard

A quantified boolean formula (QBF) is a formula $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi$ with $Q_i \in \{\exists, \forall\}$. Deciding satisfiability of QBFs is well known to be PSPACE-complete. We show PSPACE-hardness by providing a reduction from deciding satisfiability of QBFs. Similar to our problem, deciding satisfiability can be formulated as finding a winning strategy for a 2-player game. In the QBF Game, we have an $\exists$-player and a $\forall$-player. For each Q_i, if $Q_i = \exists$ the $\exists$-player selects a truth assignment for x_i. Otherwise, if $Q_i = \forall$ the $\forall$-player selects a truth assignment for x_i. The $\exists$-player wins if after the n-th round φ has a satisfying assignment. A QBF ϕ is satisfiable if and only if the $\exists$-player has a winning strategy [Sh19].

We provide a reduction from QBF Game to Robust Connection Game. Let $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi$ be a QBF. We can safely assume that φ is in conjunctive normal form, i. e., $\varphi = C_1 \wedge \cdots \wedge C_m$. We construct an instance $I = (G, s, z, x, \delta)$ of the Robust Connection Game from ϕ. In particular we will create G based on the order and type of quantifiers in ϕ by placing a gadget (a specific subgraph) for every such quantifier. The travel time of every arc e in G is uniformly set to $\lambda(e) = 1$. Additionally, we specifiy a start vertex s and an end vertex z in G. Finally we set $\delta = 1$ and the budget $x = nm + |\forall| + 1$, where $|\forall|$ is the number of universally quantified variables. The significance of the value of x will become obvious with the construction of G.



(a) Existential variable gadget (b) Universal variable gadget

Fig. 1: Depiction of an existential and universal variable gadget. Escape arcs to z are depicted as upwards pointing arcs.

In this section we will first describe the structure of the created instance (in particular the construction of G). We also state some observations about the forced behavior of the traveler and the adversary when playing the Robust Connection Game on I. Finally, we will describe the equivalences between the choices of the traveler/the adversary and the $\exists$-player/the $\forall$-player in the QBF Game.

4.1 Construction of the graph G

First let Q_i be the i-th quantifier, quantifying the variable x_i. We place a gadget for Q_i, which is a specific subgraph of G. If Q_i is an existential quantifier we place the existential gadget $\mathcal{G}_i^{\exists}$, otherwise we place the universal gadget $\mathcal{G}_i^{\forall}$, both defined below. Every such gadget has an entry vertex s_i and an exit vertex s_{i+1}, i. e., the exit vertex of a gadget $\mathcal{G}_i$ is the entry vertex of the next gadget $\mathcal{G}_{i+1}$, with the exception of $\mathcal{G}_n$, whose exit vertex s_{n+1} is not an entry vertex, since no following gadget exists. The starting time of all outgoing arcs of s_i will be identical and denoted as t_{s_i}. We define $t_{s_i} = (i-1) \cdot 2(m+2)$. Thus, the traveler needs to be able to traverse the variable gadget in $2(m+2)$ units of time or less to have a winning strategy, otherwise they arrive at the next gadget and can not continue on from s_{i+1}. We will now first describe the construction of the two gadget types in detail, before carrying on with the further construction of G.

Existential variable gadget. We begin with the existential variable gadget $\mathcal{G}_i^{\exists}$ (see Fig. 1a). For each clause C_j, the gadget contains two vertex pairs u_i^j, v_i^j and $\overline{u}_i^j, \overline{v}_i^j$ connected via arcs $e_i^j = (u_i^j, v_i^j, t_{c_j}, 1)$ and $\overline{e}_i^j = (\overline{u}_i^j, \overline{v}_i^j, t_{c_j}, 1)$ respectively. We will call e_i^j a *positive* and $\overline{e}_i^j$ a *negative* arc. It is important to mention that $t_{c_j} > t_{s_{n+1}}$, i. e., any positive or negative arc of any gadget has a leaving time later than the outgoing arcs of s_{n+1}.

Furthermore, there are two distinct paths from s_i to s_{i+1}, namely $P_i = \langle s_i, u_i^1, \dots, u_i^m, s_{i+1}\rangle$ and $\overline{P}_i = \langle s_i, \overline{u}_i^1, \dots, \overline{u}_i^m, s_{i+1}\rangle$. The traveler chooses one of these two paths by choosing which of the two outgoing arcs at s_i they traverse. The starting times of two consecutive arcs of both P_i and $\overline{P}_i$ are always two time units apart, thus the traveler requires at most $2(m+1)$ units of time to travel from s_i to s_{i+1}. Lastly, at each vertex $u_i^j, \overline{u}_i^j$ there will be an arc to z, starting at timestamp $t_{c_j} + 1$. We will refer to such arcs as *escape arcs*.

Observation 4.1. *If the traveler arrives at s_i of $\mathcal{G}_i^\exists$ at a timestamp $t \le t_{s_i}$, the adversary cannot prevent the traveler from arriving at s_{i+1} at a timestamp $t' \le t_{s_{i+1}}$ by delaying any or all arcs of P_i and $\overline{P}_i$.*

Proof. The traveler can clearly take either of the two arcs (s_i, v_i^1), $(s_i, \overline{v}_i^1)$, if they are delayed or not. Since the arrival time of any arcs on the paths is always two units before the leaving time of the next and $\delta = 1$, the observation follows. □

Observation 4.2. *If the traveler arrives at a vertex v_i^j of $\mathcal{G}_i^\exists$, they can reach u_i^j at a timestamp $t \le t_{c_j} + 1$ and subsequently reach z if and only if, the adversary does not delay e_i^j.*

Proof. We can assume that the escape arc a at u_i^j is not delayed since delaying an escape arc can never prevent the traveler from reaching z. The observation immediately follows from the fact that $t(e_i^j) + \lambda(e_i^j) = t(a)$ if e_i^j is not delayed and $t(e_i^j) + \lambda(e_i^j) > t(a)$ if it is. □

The following Lemma 4.3 summarizes the properties of the gadget, taking into account Observation 4.1 and Observation 4.2.

Lemma 4.3 (★). *If the traveler arrives at s_i of $\mathcal{G}_i^\exists$ at a timestamp $t \le t_{s_i}$, either they can reach z or they arrive at s_{i+1} at a timestamp $t' \le t_{s_{i+1}}$ via P_i (or $\overline{P}_i$) and the adversary has delayed all m arcs e_i^j (or all m arcs $\overline{e}_i^j$).*

Universal variable gadget. The universal variable gadget $\mathcal{G}_i^\forall$ has a very similar structure to the existential variable gadget with only two minor modifications (see Fig. 1b). First, we add two additional vertices u_i^{m+1} and v_i^{m+1} which are connected by an arc $e_i^{m+1} = (u_i^{m+1}, v_i^{m+1}, t_{c_{j+1}}, 1)$. Again, $t_{c_{j+1}} > t_{s_{n+1}}$ holds. There will be an escape arc from v_i^{m+1} leaving at timestamp $t_{c_{j+1}} + 1$. Furthermore, the node u_i^{m+1} is inserted into the path P_i after node u_i^m. The starting times of the arcs along path P_i are adapted accordingly to guarantee the two time unit difference between two consecutive arcs along the path. Secondly, the starting times for all arcs along path P_i, except for the first arc, are decreased by one unit.

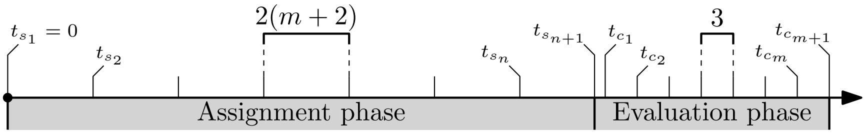


Fig. 2: Schedule of the reduction

Thus, the traveler requires at most $2(m + 1) + 1$ units of time to travel from s_i to s_{i+1} along path P_i. Everything else remains the same.

Note that while Observation 4.2 directly translates to $\mathcal{G}_i^{\forall}$, Observation 4.1 does not. Instead we make the following observation.

Observation 4.4. *If the traveler arrives at s_i of $\mathcal{G}_i^{\forall}$ at a timestamp $t \leq t_{s_i}$, the adversary can prevent the traveler from traversing P_i by delaying the arc (s_i, v_i^1).*

Note that the traversal of $\overline{P}_i$ can still not be impeded by the adversary.

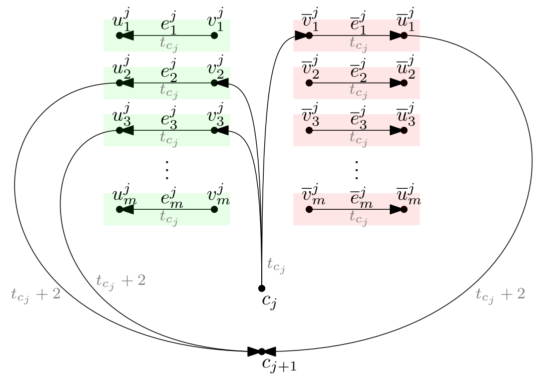
Similarly to the existential variable gadget, the traveler will either reach z or s_{i+1} after traversing the gadget. However, in the latter case, three different outcomes may arise with respect to the number of delayed edges. Note that one of the possibilities (Situation b in Lemma 4.5) will lead to a forced loss of the traveler later on. We will argue this after presenting all gadgets.

Lemma 4.5 (★). *If the traveler arrives at s_i of $\mathcal{G}_i^{\forall}$ at a timestamp $t \leq t_{s_i}$, either they can reach z or the traveler reaches s_{i+1} at timestamp $t' \leq t_{s_{i+1}}$ and one of the following situations occurs:*

(a) All $m + 1$ positive arcs e_i^j are delayed.
(b) All m negative arcs $\overline{e}_i^j$ are delayed.
(c) All m negative arcs $\overline{e}_i^j$ as well as the first arc in P_i are delayed.

The construction of G and the observed traversal behavior of the two players maps naturally to an assignment of a variable x_i to either true or false if the set of all positive or all negative arcs in $\mathcal{G}_i$ is delayed, respectively. We call the traversal of the traveler from s until s_{n+1}, the *assignment phase*. Now we continue with the remaining construction of G, which will model the evaluation of this variable assignment. The remaining traversal of the traveler starting at s_{n+1} will accordingly be called the *evaluation phase* (see Fig. 2 and 4).

In the evaluation phase, we check whether the variable assignments implied by the choices of the traveler and the adversary result in each clause C_j in φ to evaluate to true. This is done by placing a gadget $\mathcal{G}_j^C$ for every clause C_j. The gadget $\mathcal{G}_j^C$ contains an entry node c_j and an exit node c_{j+1} (which is again the entry node for $\mathcal{G}_{j+1}^C$). The gadgets have to be traversed in sequential order. The starting time of all outgoing arcs of c_j is identical and denoted as t_{c_j}. We define $t_{c_j} = t_{s_{n+1}} + 1 + 3(j - 1)$ (three time units per clause).



u_1^j e_1^j v_1^j
u_2^j e_2^j v_2^j
u_3^j e_3^j v_3^j
u_m^j e_m^j v_m^j
$\overline{v}_1^j$ $\overline{e}_1^j$ $\overline{u}_1^j$
$\overline{v}_2^j$ $\overline{e}_2^j$ $\overline{u}_2^j$
$\overline{v}_3^j$ $\overline{e}_3^j$ $\overline{u}_3^j$
$\overline{v}_m^j$ $\overline{e}_m^j$ $\overline{u}_m^j$
t_{c_j}
$t_{c_j}+2$
c_j
c_{j+1}

Fig. 3: Clause gadget

The last vertex of the assignment phase, s_{n+1}, is connected to c_1 by an arc with starting time $t_{s_{n+1}}$. Furthermore, there will be an additional arc leaving s_{n+1} to new vertex v, also leaving at timestamp $t_{s_{n+1}}$. Vertex v is connected by an escape arc to vertex z with starting time $t_{s_{n+1}} + 1$. Lastly, the final exit node c_{m+1} of the last clause gadget is connected to target vertex z with starting time $t_{c_{m+1}}$.

Lemma 4.6 (★). *When the traveler reaches c_1, if the adversary has delayed less than $nm + |\forall| + 1$ arcs, where $|\forall|$ is the number of universally bound variables, the traveler was able to reach z in the assignment phase.*

Further if the game progresses to a state, in which the traveler reaches c_1 and was so far unable to reach z, the budget of the adversary must be 0 and due to Lemma 4.6 we will assume that this is the case for the remainder of this reduction.

Clause gadget. The clause gadget for some clause C_j has an arc from c_j to v_i^j (or $\overline{v}_i^j$) for each literal x_i (or $\neg x_i$) in C_j with starting time t_{c_j}. Equivalently, there is an arc from u_i^j (or $\overline{u}_i^j$) to c_{j+1} for each literal x_i (or $\neg x_i$) with starting time $t_{c_j} + 2$. This forms a set of parallel paths, each containing some arc e_i^j (or $\overline{e}_i^j$). We will call the the set of arcs e_i^j (or $\overline{e}_i^j$) the *literal arcs* of $\mathcal{G}_j^C$. Note that the set of literal arcs can include both positive and negative arcs of the variable gadgets. The gadget is depicted in Fig. 3. Also note that if an arc $a = (u_i^j, c_{j+1})$ (or $(\overline{u}_i^j, c_{j+1})$) is delayed, the traveler will be stuck at c_{j+1}. A direct consequence of Lemma 4.6 is that at the adversary will always have at least 1 delay remaining in the assignment phase. This prevents the traveler from traversing a literal arc during the assignment phase and using a to "jump ahead" to the evaluation phase, since the adversary could delay a. Conversely, in the evaluation phase, the adversary has no budget left and cannot delay a.

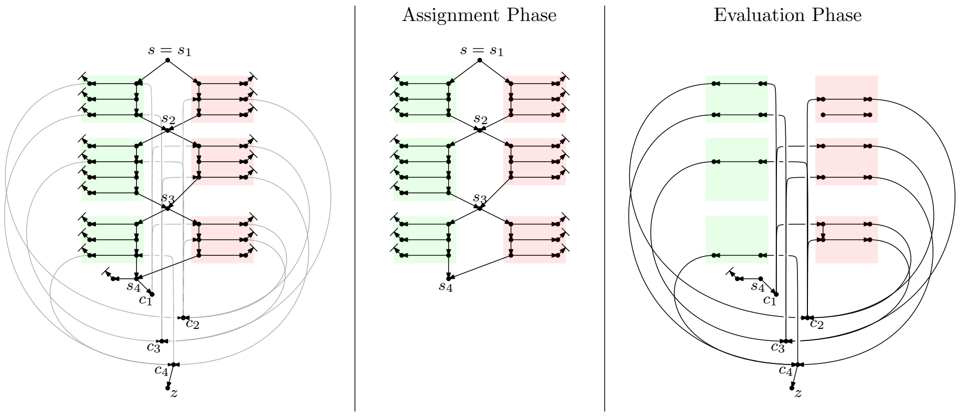


Fig. 4: Resulting graph G from reducing formula $\phi = \exists x_1 \forall x_2 \exists x_3((x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3))$. On the right you see the two phases extracted from G.

Lemma 4.7 (★). *If the traveler arrives at the entry node c_j of $\mathcal{G}_j^C$ at a timestamp $t \leq t_{c_j}$ they can travel to the exit node c_{j+1} of $\mathcal{G}_j^C$ if and only if at least one literal arc of $\mathcal{G}_j^C$ has been delayed.*

With the construction of G and therefore the whole instance I concluded, we can move on to the main statement of the reduction.

4.2 Proving PSPACE-completeness

Here we establish the connection and in fact the equivalence of a the graph traversal in the Robust Connection Game and the variable assignments in the QBF Game.

Lemma 4.8 (★). *There is a winning strategy for QBF Game for ϕ if and only if there is a winning strategy for the traveler in the reduction instance of Robust Connection Game.*

Since deciding if there is a winning strategy in the QBF Game is PSPACE-hard, we conclude the following theorem from Lemma 4.8 and Theorem 3.4.

Theorem 4.9. *Robust Connection Game is* PSPACE*-complete.*

5 Conclusion

We close with directions for future research: First, the running time for Robust Connection Game has the form $n \cdot m^{O(x)}$, where n, m, and x are the number of vertices, temporal arcs, and delayed arcs, respectively. It is natural to ask whether we can remove the dependence of the exponent on x to obtain a running time of the form $f(x) \cdot (n \cdot m)^{O(1)}$. Furthermore, what if we ask for routes with only a small number c of changeovers? Is there a running time of $(n \cdot m)^{f(c)}$ possible? Finally, it would be interesting to see how we can incorporate delays that vary between different routes or change over time into our model and whether this impacts our results.

Acknowledgements. The authors would like to thank Manuel Sorge for his input on the hardness proof, the fruitful discussion, and the feedback on the first version of this paper.

References

[22] Statistisches Bundesamt Deutschland - GENESIS-Online: Ergebnis 46181-0005, Accessed: 25.05.2023, Statistisches Bundesamt Deutschland, 2022, URL: `https://www-genesis.destatis.de/genesis/online?sequenz=tabelleErgebnis&selectionname=46181-0005`.

[23] Erläuterung Pünktlichkeitswerte April 2023, Accessed: 25.05.2023, Deutsche Bahn, 2023, URL: `https://www.deutschebahn.com/de/konzern/konzernprofil/zahlen_fakten/puenktlichkeitswerte-6878476`.

[Ba16] Bast, H.; Delling, D.; Goldberg, A. V.; Müller-Hannemann, M.; Pajor, T.; Sanders, P.; Wagner, D.; Werneck, R. F.: Route Planning in Transportation Networks. In: Algorithm Engineering - Selected Results and Surveys. Vol. 9220, LNCS, pp. 19–80, 2016.

[Be96] Berman, K. A.: Vulnerability of scheduled networks and a generalization of Menger's Theorem. Networks 28/3, pp. 125–134, 1996.

[De23] Depian, T.; Kern, C.; Röder, S.; Terziadis, S.; Wallinger, M.: Network Navigation with Online Delays is PSPACE-complete. CoRR abs/2308.12265/, 2023, URL: `https://arxiv.org/abs/2308.12265`.

[Fü22a] Füchsle, E.; Molter, H.; Niedermeier, R.; Renken, M.: Delay-Robust Routes in Temporal Graphs. In: Proc. 39th International Symposium on Theoretical Aspects of Computer Science (STACS '22). Vol. 219. LIPIcs, 30:1–30:15, 2022.

[Fü22b] Füchsle, E.; Molter, H.; Niedermeier, R.; Renken, M.: Temporal Connectivity: Coping with Foreseen and Unforeseen Delays. In: Proc. 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND '22). Vol. 221. LIPIcs, 17:1–17:17, 2022.

[Sh19] Shukla, A.; Biere, A.; Pulina, L.; Seidl, M.: A Survey on Applications of Quantified Boolean Formulas. In: Proc. 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI '19). Pp. 78–84, 2019.