Minimizing Reachability Times on Temporal Graphs via Shifting Labels

Argyrios Deligkas[1] , **Eduard Eiben**[1] , **George Skretas**[2]
[1]Royal Holloway, University of London
[2]Hasso Plattner Institute, University of Potsdam
{argyrios.deligkas, eduard.eiben}@rhul.ac.uk, georgios.skretas@hpi.de

Abstract

We study how we can accelerate the spreading of information in temporal graphs via shifting operations; a problem that captures real-world applications varying from information flows to distribution schedules. In a temporal graph there is a set of fixed vertices and the available connections between them change over time in a predefined manner. We observe that, in some cases, shifting some connections, i.e., advancing or delaying them, can decrease the time required to reach from some vertex (source) to another vertex. We study how we can minimize the maximum time a set of sources needs to reach every vertex, when we are allowed to shift some of the connections. If we restrict the allowed number of changes, we prove that, already for a single source, the problem is NP-hard, and W[2]-hard when parameterized by the number of changes. Then we focus on unconstrained number of changes. We derive a polynomial-time algorithm when there is one source. When there are two sources, we show that the problem becomes NP-hard; on the other hand, we design an FPT algorithm parameterized by the treewidth of the graph plus the lifetime of the optimal solution, that works for any number of sources. Finally, we provide polynomial-time algorithms for several graph classes.

1 Introduction

Every day million pieces of information need to be disseminated and most often it is desirable to minimize their delivery time. In many cases, the diffusion of information depends on schedules of physical and online meetings between entities that form a dynamic network that changes over time, depending on their availability. Usually, these schedules are constrained due to physical limitations, laws, available infrastructure, and costs of extra resources and meetings. These constraints are usually unavoidable since it is difficult, if not impossible, to bypass them. On the other hand, careful changes on the existing scheduling timetable can significantly reduce the time a piece of information needs to reach every recipient.

For an example motivated by real life, consider the scenario where there are three employees A, B, and C in a university, where each one of them has some information that needs to reach every other person. The central timetabling team has already arranged some meetings for them. Between employees A and B there is one meeting at 9am and one at 11am. Between B and C there exists a meeting arranged at 8am and another one at 4pm. Then observe that under the current timetable, the information A has will reach C at 4pm – B will receive the information from A at 9am and they will transmit it to C at 4pm – while the information C has can reach A at 9am. However, if we *delay* the 8am meeting between B and C to 10am, then the information of A can reach C at 10am, and the information of C can still reach A at 11am. Figure 1 depicts this example.

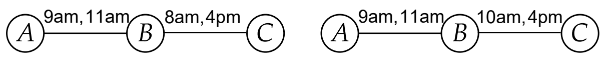

Figure 1: Left: The original timetable. Right: The modified timetable with the delayed meeting time.

The inherent temporal nature of timetabling, combined with the existence of the underlying temporal network, allows us to concisely formulate the scheduling scenario described above as an optimization problem over a *temporal graph* [Kempe *et al.*, 2002]. At a high level, a temporal graph consists of a fixed set of vertices and a timetable that defines available connections, or *temporal edges*, at any point in time. A temporal path is a sequence of temporal edges which, additionally to the usual connectivity, respects the time constraints, i.e., temporal edges that are used later in the sequence, are available later in time than the ones that come before [Whitbeck *et al.*, 2012]. As we have seen in the example above, there exist cases where delaying some temporal edges makes some temporal paths "faster". In other words, both advancing and delaying some edges can make the existing infrastructure more efficient in terms of reachability times.

Our contribution. The problem studied in this paper, which we term REACHFAST, can be summarized as follows.

> *Given a temporal graph and a set of sources,* shift *some temporal edges, i.e.,* advance *or* delay*, such that the maximum time any source needs to reach all the vertices is* minimized.

First, we study REACHFAST with a single source in the graph. We distinguish between two cases: either some constraint on the advances/delays is imposed, or no such constraints exist. We study two natural constrained versions that can appear in real life scenarios:

- REACHFAST(k), where k bounds the number of edges we can shift;
- REACHFASTTOTAL(k), where k bounds the sum of absolute shifts over all edges.

In Theorem 2 we prove that both constrained versions are W[2]-hard when parameterized by k and NP-hard otherwise. We complement these negative results with an algorithm that solves (unconstrained) REACHFAST in polynomial time.

Then, we depart from the single source case and study the problem when there are more sources. In Theorem 5, we prove that REACHFAST becomes NP-hard even if there are just two sources and the temporal graph is rather restricted: the graph has lifetime 1 and (static) diameter 6. This indicates that tractability requires some constraints in the input temporal graph. We derive efficient algorithms for REACHFAST for several graph classes of the underlying network. For trees, we show that REACHFAST can be solved in polynomial time for any number of sources. Then, we consider several parallel paths with common endpoints. For this class, we design a polynomial-time algorithm for REACHFAST when there are only two sources and they are the endpoints of the paths. Our last result considers graphs of bounded treewidth; this class of graphs has received a lot of attention in the past. We show that in this case REACHFAST can be encoded in Monadic Second Order logic, where the size of the formula depends only on the deadline by which we want to reach all the vertices from every source. Then, using Courcelle's Theorem, we get a fixed parameter tractable algorithm for the problem, parameterized by the treewidth of the graph and the lifetime of the optimal solution that works for any number of sources.

Related work. The modification of temporal graphs such that an objective is optimized, has received significant attention recently, mainly motivated by virus-spread minimization [Braunstein and Ingrosso, 2016; Enright and Kao, 2018]. A line of work studies minimization of reachability sets for a set of sources over a temporal graph using a variety of operations: delaying operations were first studied in [Deligkas and Potapov, 2020], alongside merging operations; edge-deletion and temporal edge-deletions operations were studied in [Enright *et al.*, 2021a; Enright and Meeks, 2018]; reordering of the temporal edges was studied in [Enright *et al.*, 2021b]. Finally, [Molter *et al.*, 2021] studies the relationship between the delaying and the edge-deletion operations under the reachability-minimization objective. All of the above mentioned reachability problems heavily depend on the computation of temporal paths, a problem that has received a lot of attention [Wu *et al.*, 2014; Wang *et al.*, 2015; Wu *et al.*, 2016; Wu *et al.*, 2012] and different definitions of temporal paths have been proposed [Casteigts *et al.*, 2021; Thejaswi and Gionis, 2020]. In [Li *et al.*, 2018] a similar idea to ours was explored under a slightly different objective. There, every temporal edge had a traversal-time that depends on the time it is used and the goal was to find temporal paths that minimize the overall traversal-time. The authors observed that, for their objective, it might be beneficial to wait until they use an edge, since this might decrease the total traversal-time. Another recent work of interest [Klobas *et al.*, 2022] studies the problem of finding the minimum number of labels that have to be added so that a temporal graph G is temporally connected. This problem seems really close to ours, albeit they have some crucial differences. Namely, temporal connectivity requires "both-ways" connections, while we focus on "one-way" connections.

2 Preliminaries

For $n \in \mathbb{N}$, we denote $[n] := \{1, 2, \dots, n\}$. A *temporal graph* $\mathcal{G} := \langle G, \mathcal{E} \rangle$ is defined by an *underlying graph* $G = (V, E)$ and a sequence of edge-sets $\mathcal{E} = (E_1, E_2, \dots, E_{t_{\max}})$. It holds that $E = E_1 \cup E_2 \cup \dots \cup E_{t_{max}}$. $E_{t_{max}} \neq \emptyset$ and the *lifetime* of $\mathcal{G}$ is $t_{\max}$. An edge $e \in E$ has label i, if it is available at time step i, i.e., $e \in E_i$. The traversal time of an edge $e \in E_i$ at time step i is $\mathrm{tr}(i, e)$; i.e., staring from one endpoint of the edge at time step i, we reach the the other endpoint at the time step $i + \mathrm{tr}(i, e)$.

A *temporal path* in $\langle G, \mathcal{E} \rangle$ from vertex v_1 to vertex v_m is a sequence of edges $P = (v_i v_{i+1}, t_i)_{i=1}^{m-1}$ such that for every $i \in [m]$ it holds that:

- $v_i v_{i+1} \in E_{t_i}$, i.e. $v_i v_{i+1}$ is available at time step t_i;
- $t_i + \mathrm{tr}(t_i, v_i v_{i+1}) \leq t_{i+1}$ for every $i \in [m-1]$.

The *arrival* time of P is $t_m = t_{m-1} + \mathrm{tr}(t_{m-1}, v_{m-1} v_m)$. We call P a foremost temporal (v_1, v_m)-path if for every $j \in [m-1]$ there is no temporal path P'_j from v_1 to v_{j+1} such that the arrival time of P'_j is strictly smaller that the arrival time of $P_j = (v_i v_{i+1}, t_i)_{i=1}^{j}$.

A vertex v is *reachable* from vertex u by time t in a temporal graph $\mathcal{G}$ if there exists a temporal path from u to v with arrival time at most t. It is possible that u is reachable from v, but v is not reachable from u. The reachability set of v, denoted $\mathtt{reach}(v, \mathcal{G})$, contains all the vertices reachable from v. *The reaching-time* of v, denoted $\mathtt{reachtime}(v, \mathcal{G})$, is t if there exists a temporal path from v to every vertex $u \in V$ with arrival time at most t. If there is a vertex $u \notin \mathtt{reach}(v, \mathcal{G})$, then we will say that v has infinite reaching-time.

The *shifting* of edge $uv \in E_i$ by $\delta \in \mathbb{Z} \setminus \{0\}$, denoted $\mathtt{delay}(uv, i) = \delta$, refers to replacing the label i of this edge with the label $i + \delta$. When δ has a positive value, we say that the label is *delayed*, and when δ has a negative value, we say that the label is *advanced*. Note that shifting an edge is allowed only when $i + \delta \geq 1$, but we allow $i + \delta > t_{max}$, i.e. we are allowed to increase the lifetime of the graph by delaying edges. Given a temporal graph $\mathcal{G}$ and a set of edges $X \subseteq E \times [t_{max}]$ we denote $\tilde{\mathcal{G}}(X)$ a class of temporal graphs where every edge in X is shifted. A temporal graph $\mathcal{G}' \in \tilde{\mathcal{G}}(X)$ has k edges shifted, if $|X| = k$; the total shift of $\mathcal{G}'$ is k if $\sum_{i \in [t_{\max}]} \sum_{uv \in E_i} |\mathtt{delay}(uv, i)| = k$. Observe that shifting an edge can increase the reachability set of a vertex v, or it can decrease the reaching-time between two vertices. See Figure 2 for an example.

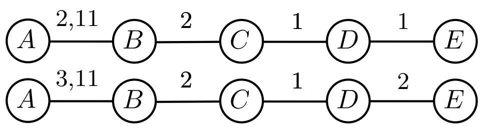

Figure 2: Top: The original temporal graph of a network where we assume that every label has traversal time equal to 1. Note that vertex C reaches vertex A at time step 12 and vertex E is unreachable. Bottom: The modified temporal graph by delaying edges AB and DE. Note that now, vertex C reaches vertex A at time step 4 and vertex E can now be reached at time step 3.

The ReachFast problem. An instance of ReachFast consists of a temporal graph $\mathcal{G} := \langle G, \mathcal{E} \rangle$ and a set of sources $S \subseteq V$. The goal is to minimize the reaching-time any source requires to reach all the vertices of V under shifting operations. We say that a temporal graph $\mathcal{G}'$ is a *solution* to an instance $\langle \mathcal{G} = \langle G, \mathcal{E} \rangle, S \rangle$ of ReachFast if $\mathcal{G}' \in \tilde{\mathcal{G}}(X)$ for some $X \subseteq E \times [t_{max}]$ and for every $v \in S$ we have $\texttt{reach}(v, \mathcal{G}') = V(G)$. The *value* of a solution $\mathcal{G}'$ is $\max_{v \in S} \texttt{reachtime}(v, \mathcal{G}')$. We say that a solution $\mathcal{G}'$ is *optimal* if the value of $\mathcal{G}'$ is minimized among all solutions. In other words, our objective is

$$\min_{X \subseteq V, \mathcal{G}' \in \tilde{\mathcal{G}}(X)} \max_{v \in S} \texttt{reachtime}(v, \mathcal{G}').$$

In addition, we study two *constrained* versions of the problem. In ReachFast(k) we require that $|X| \leq k$, where k is a part of the input; in ReachFastTotal(k) we require that $\sum_{i \in [t_{\max}]} \sum_{uv \in E_i} |\texttt{delay}(uv, i)| \leq k$.

Observation 1. *Given a temporal graph $\mathcal{G} := \langle G, \mathcal{E} \rangle$, where the underlying graph G is connected and a set of sources $S \subseteq V$. If every edge of $\mathcal{G}$ has at least $|S|$ labels, then* ReachFast *on $\mathcal{G}$ has a solution.*

The next observation shows that when we can shift any number of edges, ReachFast under shifting reduces to ReachFast under delaying operations *only*.

Observation 2. *Let $\mathcal{G} := \langle G, \mathcal{E} \rangle$ be a temporal graph, $S \subseteq V(G)$ a set of sources and let $\mathcal{G}' := \langle G, \mathcal{E}' \rangle$ be the temporal graph constructed from $\mathcal{G}$ by advancing the labels of each edge e to the first k_e time steps, where k_e is the number of labels of e. Then, any solution for* ReachFast *under S in graph $\mathcal{G}'$ that only delays edges, is also a solution for* ReachFast *under S in $\mathcal{G}$.*

Parameterized complexity. We refer to the standard books for a basic overview of parameterized complexity theory [Cygan *et al.*, 2015; Downey and Fellows, 2013], and assume that readers are aware of the graph parameter *treewidth* and the complexity classes FPT, XP, W[1], and W[2].

Monadic Second Order Logic. We consider *Monadic Second Order* (MSO) logic on labeled graphs in terms of their incidence structure whose universe contains vertices and edges; the incidence between vertices and edges is represented by a binary relation. Let $\Phi(X_1, X_2, \ldots, X_k)$ be an MSO formula with a free set variables $X_1, \ldots, X_k$. For a labeled graph $G = (V, E)$ and sets $S_1, \ldots, S_k \subseteq E$ we write $G \models \Phi(S_1, \ldots, S_k)$ if the formula Φ holds true on G whenever X_i is instantiated with S_i for all $i \in [k]$.

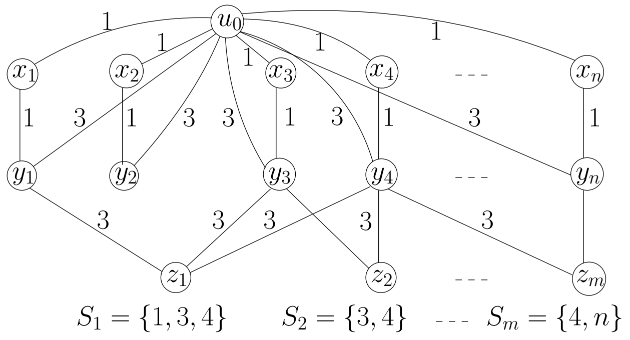

Figure 3: The construction used at Theorem 2. The subsets of the HITTING SET problem are $S_1 = \{1,3,4\}, S_2, = \{3,4\}, \ldots, S_m = \{4,n\}$.

Fact 1 ([Arnborg *et al.*, 1991]). *Let $\Phi(X_1, \ldots, X_k)$ be an MSO formula with free sets variables $X_1, \ldots, X_k$ and w a constant. Then there is a linear-time algorithm that, given a labeled graph $G = (V, E)$ of treewidth at most w, decides whether there exists k sets $S_1, \ldots, S_k, \subseteq E(G)$ such that $G \models \Phi(S_1, \ldots, S_k)$. Moreover, if such solution exists, the algorithm constructs one such solution.*

We note that [Arnborg *et al.*, 1991] does not explicitly construct the solution, however, for each formula, they construct a tree automaton that works along the decomposition and stores $f(w, |\Phi(X_1, \ldots, X_k)|)$-many records (for some computable function f) for the optimal solutions. To make it constructive, for each record we only need to remember a viable representative.

3 One Source

In this section we study the most basic case of ReachFast problem, where there is only one source. To begin with, we show that both constrained versions of the problem are W[2]-hard when parameterized by k.

Theorem 2. *Problems* ReachFastTotal(k) *and* ReachFast(k) *are* W[2]*-hard when parameterized by k.*

Proof sketch. We will prove the theorem via a reduction from HittingSet. An instance of HittingSet consists of a collection of subsets $S_1, S_2, \ldots, S_m$ over $[n]$ and a positive integer k, and we need to decide if there is a set $T \subset [n]$ of size k, such that $S_j \cap T \neq \emptyset$ for every $j \in [m]$. HittingSet is known to be W[2]-hard when parameterized by k (see, e.g., [Cygan *et al.*, 2015]).

Given an instance of HittingSet with m sets over $[n]$, known to be W[2]-hard when parameterized by the solution size k, we construct a temporal graph $\mathcal{G}$ with $2n + m + 1$ vertices as follows (see Figure 3).

First we let u_0 be the source. Then, we construct the vertices $x_1, x_2, \ldots, x_n$, and the vertices $y_1, y_2, \ldots, y_n$. For every $i \in [n]$ we have three edges: $u_0 x_i$ with label 1; $x_i y_i$ with label 1; $u_0 y_i$ with label 3. Finally, we construct the vertices $z_1, z_2, \ldots, z_m$, where vertex z_j corresponds to subset S_j. If $i \in S_j$, then we create the edge $y_i z_j$ with label 3.

Algorithm 1: One Source Reach Fast Algorithm

```
Require: A temporal graph $\mathcal{G} := \langle G, \mathcal{E} \rangle$ and a source $v$.
Ensure: An optimal solution $\mathcal{G}'$.
1: $\mathcal{S} \leftarrow v$; $\mathcal{S}_t \leftarrow \emptyset$; $r = 0$;
2: while $|\mathcal{S}| \neq |V|$ do
3:     for every vertex $u \in \mathcal{S}$ do
4:         for every neighbor $w \notin \mathcal{S}$ of $u$ do
5:             for $i = 1, 2, \dots, r$ do
6:                 if $uw \in E_i$ then
7:                     $min = i + \mathrm{tr}(i, uw)$
8:                     for $j == i, j++, min < j$ do
9:                         if $min < \mathrm{tr}(j, wu)$ then
10:                            $min = \mathrm{tr}(j, wu)$
11:                    $\mathtt{delay}(uw, i) = j - i$
12:                    $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \{w\}$
13:    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_t$; $\mathcal{S}_t \leftarrow \emptyset$; $r++$;
```

We claim that there exists a solution to REACHFAST(k) and REACHFASTTOTAL(k), such that u_0 reaches all the vertices at time step 3 that in the temporal graph we constructed above, if and only if there is a solution to the original HITTINGSET instance with $|T| = k$. Note that the above reduction still holds when the traversal time of each edge is larger than 1. ☐

The proof of Theorem 2 immediately implies the following corollary. Therefore, if we want to prove any positive result, then we need to either restrict the underlying graph structure, or study the unconstrained version of REACHFAST.

Corollary 3. REACHFAST(k) *and* REACHFASTTOTAL(k) *are* NP*-hard when k is part of the input.*

On the positive side, we can design a polynomial-time algorithm, termed Algorithm 1, for REACHFAST, when there is a single source. We will utilize Observation 2 and wlog we will assume that all the labels appear at the first time step; thus we have to consider only delaying operations.

Algorithm 1 uses a greedy approach. In every round r, it checks every vertex u that has been reached by the source by time t_r, and delays every incident unused temporal edge uw to the time step with the minimum arrival time to w, if vertex w has not yet been reached by source vertex v.

Theorem 4. *Algorithm 1 solves the unconstrained version of the* REACHFAST *with one source in polynomial time.*

4 Two Sources

In this section we study the case where we have more than one sources. Since we know from Theorem 2 that the constrained versions of REACHFAST are NP-hard even when there is a unique source, we will only consider the unconstrained case. As we will show below, REACHFAST becomes hard even if there are just two sources. In fact, the underlying graph that results from our reduction always admits an optimal solution of a constant value. Note that this also means that the problem is even APX-hard; i.e., unless P = NP, there is no polynomial-time c-approximation for some $c > 1$. We highlight that the constructed temporal graph has lifetime 1 and every label has traversal time 1 as well. This means that, essentially, there are no temporal constraints in the edges.

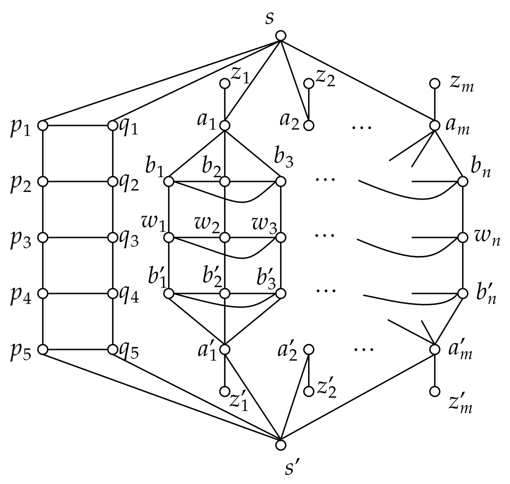

Figure 4: The construction used in Theorem 5. The first clause of the MONLINNAE3SAT instance is $c_1 = (x_1 \vee x_2 \vee x_3)$.

Theorem 5. *It is* NP*-complete to decide whether* REACHFAST *with two sources admits a solution of value 6, even on temporal graphs of lifetime 1.*

We will prove the theorem via a reduction from a special version of NAE3SAT, termed MONLINNAE3SAT [Darmann and Döcker, 2020]. We are given a CNF formula ϕ that involves n variables and m clauses that satisfy the following constraints: every clause is made up of exactly three distinct literals; each variable appears exactly four times; there are no negations in the formula; each pair of clauses shares at most one variable. The goal is to decide whether there exists a truth assignment, such that for each clause at least one literal is set to true and at least one literal is set to false.

Construction. Given an instance ϕ of MONLINNAE3SAT, we will create a temporal graph $\mathcal{G}$ with $3n + 4m + 12$ variables. Since the graph has lifetime 1, we do not have to specify the labels of the edges. The two sources are the vertices s and s'. For every clause c_j we create two clause-vertices a_j, a'_j, two auxiliary vertices z_j, z'_j, and the edges $sa_j, s'a'_j$ and $a_jz_j, a'_jz'_j$. For every variable x_i, we create the vertices b_i, b'_i and w_i. If variable x_i appears in clause c_j, then we add the edges a_ib_j and $a'_ib'_j$. In addition if variables x_i and x_j belong to the same clause we add the edges $b_ib_j, b'_ib'_j$ and w_iw_j. Finally, we create a "ladder" between s and s' using the vertices p_i, q_i where $i \in [5]$. The "ladder" is created by the edges $sp_1, sq_1, s'p_5, s'q_5$ and the edges p_iq_i for every $i \in [5]$. Figure 4 depicts ours construction. Theorem 5 follows immediately from the following two lemmas.

Lemma 6. *If the instance of* MONLINNAE3SAT *is satisfiable, then for the constructed temporal graph $\mathcal{G}$ we have that* $\min_{X \subseteq V, \mathcal{G}' \in \tilde{\mathcal{G}}(X)} \max_{v \in S} \mathtt{reachtime}(v, \mathcal{G}') = 6$.

Proof sketch. We create the labelling for $\mathcal{G}$ whose pattern is depicted in Figure 5. The proof is completed by observing

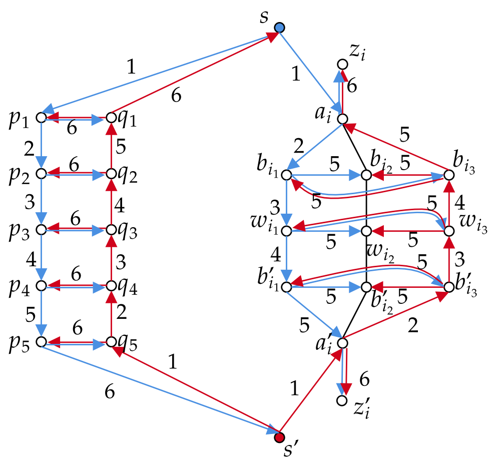

Figure 5: The labelling of $\tilde{\mathcal{G}}(X)$ for a given satisfying assignment of MONLINNAE3SAT, where in clause $c_j = (x_{i_1} \vee x_{i_2} \vee x_{i_3})$ we have $x_{i_1} = \texttt{True}$ and $x_{i_3} = \texttt{False}$; the assignment of x_{i_2} does not affect the reachability. The coloured directed paths, are the temporal paths from the corresponding source. The edges with two colors and two arrows are used at the same time step by both colours.

that under this labelling scheme s and s' reach every vertex by time step 6. □

Lemma 7. *If the constructed temporal graph $\mathcal{G}$ admits a solution of value at most* 6*, then there is a satisfying assignment for the instance of* MONLINNAE3SAT.

5 Tractable Cases

In this section, we give algorithms for REACHFAST for three special classes of underlying graphs: trees, parallel paths, and graphs of bounded treewidth. Due to Observation 2, we focus only on delaying operations, and assume that all the labels of an edge form an interval starting at time step one.

5.1 Trees

First, we focus on temporal graphs with many sources where the underlying graph is a tree. To devise a polynomial-time algorithm, we want to limit the number of solutions we have to check in order to find an optimal one. To achieve this, we show that a lot of prospective delays are *unnecessary*, i.e., they do not contribute to minimizing the maximum `reachtime` objective. We first formally define unnecessary labels and then show that they can be removed from a solution L without impacting the solution value negatively.

Definition 8. *Consider a tree $\mathcal{G}$, a set of sources S and a solution $\mathcal{G}'$ to the* REACHFAST *problem, where we apply a delay d on an edge xy that leads to the label ℓ on xy in $\mathcal{G}'$. Assume that in $\mathcal{G}'$ exists*

- *a source $r \in S$ and a leaf vertex u, where the foremost temporal path from r to u, $P_{r,u} = (r, u_1, \dots, x, y, \dots, u_k, u)$, does not use label ℓ, but uses another label $\ell' > \ell$, and*
- *a source $r' \in S$ and a leaf u', where the foremost temporal path from r' to u', $P_{r',u'} = (r, u'_1, \dots, y, x, \dots, u'_k, u')$, does not use label ℓ, but uses another label $\ell'' > \ell$.*

We call such a delay d unnecessary*. We call all other delays* necessary.

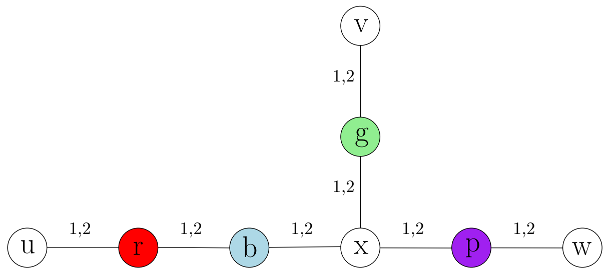

Figure 6: A tree graph with four sources: r,b,g,p

Lemma 9. *Given a tree graph $\mathcal{G}$, a set of sources S, a solution $\mathcal{G}'$ to the* REACHFAST *problem, removing an unnecessary delay d from $\mathcal{G}'$, creates a solution $\mathcal{G}''$ such that* $\max_{v\in S} \texttt{reachtime}(v, \mathcal{G}'') \leq \max_{v\in S} \texttt{reachtime}(v, \mathcal{G}')$.

Proof. Let us first give an intuitive explanation. Consider Figure 6, where we have 4 sources. Assume that we have a solution $\mathcal{G}'$, where the first label ℓ on edge ru is delayed to time step 2, the second one ℓ' to time step 4, and the foremost temporal path from p to u uses ℓ'. Even if r uses label ℓ to reach vertex u, it can also use ℓ' since p uses that label to reach vertex u. This is what makes the delay on label ℓ unnecessary and we can remove it without impacting the solution value. Let us move to the formal proof.

Based on Definition 8, let us assume that there exist two sources $r, r' \in S$, where the foremost temporal paths from r, r' to leaf vertices u, u', respectively, $P_{r,u} = (r, u_1, \dots, x, y, \dots, u_k)$ and $P'_{r,u} = (r', u'_1, \dots, y, x, \dots, u'_k)$ do not use label ℓ, but uses another label $\ell' > \ell, \ell'' > \ell$, respectively. We will show that after removing the delay d, for every $b \in S$ and every vertex v, the arrival time of a foremost path from b to v does not exceed $\max_{r\in S} \texttt{reachtime}(r, \mathcal{G}')$.

Consider any source b whose foremost temporal path $P_{b,v}$ from b to v uses the label ℓ in $\mathcal{G}'$ to traverse from x to y. Removing the delay d, makes this temporal path invalid. However, recall that the foremost path $P_{r,u}$ traverses from x to y using label $\ell' > \ell$. Since, $P_{r,u}$ is the foremost path, it is not possible to reach the vertex y from r earlier than by arrival time achieved by using label ℓ'. In particular, the foremost path $P_{r,v}$ also uses the label ℓ' on the edge xy (note that since G is a tree $P_{r,v}$ has to traverse the edge xy from x to y). Therefore, to reach the vertex v from b after we remove the delay d we can take the prefix of $P_{b,v}$ to reach the vertex x and the suffix of $P_{r,v}$ to reach from x to v at the same time as the foremost path from r arrives to v. We can follow an analogous argument in the case when the path from b to v uses the label ℓ to traverse from y to x but using the existence of $P'_{r,u}$ instead of $P_{r,u}$. Therefore, $\max_{v\in S} \texttt{reachtime}(v, \mathcal{G}'') \leq \max_{v\in S} \texttt{reachtime}(v, \mathcal{G}')$. □

Lemma 9 shows us that every solution with unnecessary delays can be mapped to an equal or better solution by removing all unnecessary delays. This leads to the following observation and lemmas that limit the search space for an optimal solution.

Observation 3. *Given a tree graph $\mathcal{G}$, and a set of sources S, there exists an optimal solution to the* REACHFAST *problem that contains no unnecessary delays.*

Lemma 10. *In every solution $\mathcal{G}'$ obtained by only necessary delays, there exists either a vertex u or an edge vw where (i) the position of u (vw) in the graph is between at least two sources and (ii) there is a time step t such that every source $r \in S \setminus \{u\}$ reaches u for the first time at t or, respectively, a path with arrival time at most $\max_{r \in S}$ $\texttt{reachtime}(r, \mathcal{G}')$ from every source traverses the edge uv with label t. We call such a vertex u,* bottleneck vertex, and such an edge vw, *bottleneck edge.*

Lemma 11. *Consider a solution $\mathcal{G}'$ for temporal graph $\mathcal{G}$ with only necessary delays that has bottleneck vertex u/edge vw. Then there exists a solution $\mathcal{G}''$ with only necessary delays such that $\max_{r \in S}$ $\texttt{reachtime}(r, \mathcal{G}'') \leq \max_{r \in S}$ $\texttt{reachtime}(r, \mathcal{G}')$ and the maximum time any source requires to reach u/vw is the minimum such time among all the solutions.*

Lemmas 10,11 give us a nice property of an optimal solution that we can algorithmically exploit. There exists an optimal solution that uses only necessary delays and minimizes the time to reach some bottleneck/edge. Note that the number of possibilities for the bottleneck is bounded by the size of the graph and we can go over all the possibilities and compute a solution as if the given possibility is a bottleneck. Given a bottleneck b we are left with two problems: (1) finding minimum time t_b needed to reach the bottleneck and (2) reaching every vertex in G when starting at b at time t_b after using some of the labels. Problem (2) can be straightforwardly reduced to REACHFAST with one source and we can use Algorithm 1 to solve it.The following lemma resolves problem (1).

Lemma 12. *There is a polynomial-time algorithm, that given a tree and a vertex u, delays only necessary labels such that the maximum time any source requires to reach u is minimized.*

Proof. In order to only use necessary delays, for every source vertex r that has vertex $w \in P_{r,u}$, every such source r must reach w before we delay label ℓ on edge wx (where x is the vertex on path $w \in P_{r,u}$ that has not been reached by source r. The algorithm works as follows. We keep a counter t that tracks the current time step and C that tracks the vertices that we still need to check. Initially, $t = 1$ and $C = V$. In each iteration of the algorithm, we check each vertex $w \in C$ to see whether there exists a source r that has not reached w. If there exists at least one such source, we move to the next vertex in C and increase t by 1. If every source r has reached w, we delay the first label on edge wx to time step t. Then, we move to the next vertex in C and increase t by 1. This generates our solution $\mathcal{G}'$. □

Theorem 13. REACHFAST *on trees can be solved in polynomial time.*

Proof sketch. The algorithm iteratively considers every edge and every vertex of the tree as if this was a bottleneck edge/vertex in an optimal solution and it outputs the solution that achieves the minimum time. For every bottleneck edge/vertex, it works in two phases. In phase 1, it uses Lemma 12 to find the minimum time needed in order all sources to reach this vertex (or the two vertices of the bottleneck edge we consider). Then, in phase 2 it uses Algorithm 1 to reach all vertices treating the bottleneck vertex as a source; if we have a bottleneck edge, then it considers them a single source. □

5.2 Parallel Paths

In this section we study parallel paths $P_1, P_2, \dots$ with common endpoints and two sources. where the endpoints are the sources, denoted A and B. Our algorithm for this case relies on guessing the time that sources reach each other in an optimal solution. In what follows, for a clearer and more intuitive presentation, we will consider every source to be represented by a distinct color. Thus, the goal is to design algorithms that minimize the time every vertex is reached by all colors. For each combination of guesses for the two sources, the algorithm also guesses which paths between A and B achieve the time of the optimal solution. Again, for each guessed path, the algorithm guesses how each source/color optimally reaches every vertex in the rest of the paths.

Theorem 14. *There is a polynomial time algorithm for the* REACHFAST *problem on graphs that consist of several parallel paths with two common endpoints, where the two endpoints are the sources.*

Proof sketch. The algorithm goes through every possible pair of reaching-times (r_1, r_2) that sources A and B can have to reach one another. This process requires $O(t_{max} + n - 1)^2$ repetitions. Note that this process runs through every possible solution of the REACHFAST problem since by time step t_{max} every label on the graph is available to be delayed and the static distance between the sources cannot be more than $n-1$. Since every label is available, if every edge has two labels and there are two sources on the graph, there is no reason for a color to delay using an edge, since there is always a second label available for the other color. If there is only one label on an edge of a path P_x, there are only two valid choices: either (i) both colors traverse the one label edge together (possibly in the opposite direction), or (ii) only one color uses path P_x. Thus the algorithm runs through every possible scenario that might yield a solution to the REACHFAST problem.

In Step 1, the algorithm checks every possible combination of paths that can achieve the reaching-time imposed by each for-loop. This can be achieved in polynomial time, since every loop checks a combination of two paths, and any graph can have at most n parallel paths between the two sources.

In Step 2, for every path combination of P_x, P_y that can achieve the desired reaching-time, the algorithm uses the multiple source tree algorithm as a subroutine to transmit the color to the rest of the edges on the paths. Specifically, for each path P_w we need to distinguish between several cases depending on whether the sources A and B reach all the vertices of P_w directly on the path P_w or, for example, the source

A uses P_x to reach B before reaching some vertices of the path P_w. Unfortunately in the latter case we cannot use Theorem 13 directly, as the vertex B becomes available for the "color" of source A only at time step r_1, but a minor modification makes the algorithm to work. $\square$

5.3 Bounded-Treewidth Graphs

In this subsection, we show that REACHFAST admits an FPT algorithm parameterized by treewidth plus the value of the sought solution D; i.e., when looking for a solution $\mathcal{G}'$ such that for all $v \in S$ it holds that $\texttt{reachtime}(v, \mathcal{G}') \leq D$. Here D can be viewed as a *deadline* by which we have to reach all vertices of the graph from all sources. As in the previous sections, we again assume that we are only allowed to delay the edges, which we can assume without loss of generality due to Observation 2.

Theorem 15. *Given a temporal graph $\mathcal{G} := \langle G, \mathcal{E}\rangle$ such that G has treewidth bounded by a constant, a set of sources $S \subseteq V(G)$, and a constant D, there is a linear-time algorithm that either computes a solution $\mathcal{G}'$ of value at most D or decides that the value of an optimal solution is at least $D + 1$.*

Proof Sketch. To prove the theorem we construct a labeled graph H such that treewidth of H is at most treewidth of G and an MSO formulation $\Phi(\mathbf{X})$ such that the size of formula $\Phi(\mathbf{X})$ depends only on D and $\mathbf{X} = (X_1^1, X_1^2, \ldots X_1^D, X_2^1, X_2^2, \ldots X_D^D)$. Observe that if for some edge $e \in E(G)$ and some time step i, we have $\mathrm{tr}(i, e) > D$, then we will never delay the edge e to the time step i and we can assume that whenever $\mathrm{tr}(i, e) > D$, then $\mathrm{tr}(i, e) = D + 1$. For sets $S_1, \ldots, S_D$ of edges, let $\mathcal{G}[S_1, \ldots, S_D]$ denote the temporal graph obtained from $\mathcal{G}$ by delaying, for each $i \in [D]$, every edge in S_i to E_i. Moreover, for every $i \in [D]$ let $\{S_i^1, \ldots, S_i^D\}$ be the partition of S_i such that edge $e \in S_i$ belongs to S_i^j if and only if $\mathrm{tr}(i, e) = j$. We will show that $H \models \Phi(S_1^1, \ldots, S_D^D)$ if and only if in the temporal graph $\mathcal{G}[S_1, \ldots, S_D]$ it holds for all $v \in S$ that $\texttt{reachtime}(v, \mathcal{G}[S_1, \ldots, S_D]) \leq D$. The result then follows by Fact 1. To construct H we let

1. $V(H) = V(G)$,
2. $E(H) = \bigcup_{i\in[D]} E_i$ (note that $E(H) \subseteq E(G)$),
3. the edge $e \in E(H)$ has label (I, T), where $I \subset [D]$ and $T = (t_1, \ldots, t_D) \in [D + 1]^D$ such that (1) for all $i \in [D]$ we have $i \in I$ if and only if $e \in E_i$, and (2) for all $j \in [D]$, we have $\mathrm{tr}(j, e) = t_j$,
4. every vertex in S has label S.

It is easy to see that H is a subgraph of G and hence the treewidth of H is bounded by the treewidth of G. The MSO formulation $\Phi(\mathbf{X})$ is a conjunction of four building blocks (subformulas) that can each be easily expressed as an MSO formulation of the size depending only on D. The first block check that each of the sets X_i^j is an edge set. The second block checks that every edge in X_i^j has a label (I, T) with $t_i = j$. The third block checks that if some edge is precisely at the time steps in $I \subseteq [D]$ after the delaying, that is the edge appears in some X_i^j for every $i \in I$ and no other X_i^j, then we can delay this edge to these time steps. This is simply a big conjuction of small subformulas stating that if an edge is precisely in some given X_i^j's then the edge has one of the labels that allow for these X_i^j's. Finally, the fourth block verifies that for every source vertex s and every other vertex t in the graph, there is an s-t path using only edges in some X_i^j's. Note that each such path has to have length at most D and each edge on the path should belong to one of X_i^j. Moreover, such temporal path is valid exactly when for two consecutive edges $e_1 \in X_{i_1}^{j_1}$, $e_2 \in X_{i_2}^{j_2}$, we have $i_1 + j_1 \leq i_2$. Hence the fourth block is a disjunction over all less than D^3 many valid possibilities of an s-t path that reaches the target by time D and checking that there exists an s-t path with edges in the prescribed sets X_i^j. $\square$

6 Conclusion

We view our paper as part of a greater agenda whose goal is to make existing infrastructures more efficient, by making minimal changes to the currently-adopted solutions. We introduced and studied the complexity of REACHFAST problem, where the goal was to minimize the maximum reaching-time of a set of sources towards every vertex of the graph using shifting operations.

Since we study optimization problems that are NP-hard, instead of focusing on restricted classes of instances, one can ask to relax the optimality condition and ask for some approximate solutions. Here we would like to point out that Theorem 5 implies the problem is APX-hard even for two sources, however existence of a constant approximation for some constant $c > 7/6$ is an interesting open question. In both, REACHFAST (k) and REACHFASTTOTAL (k) there are two criteria one might wish to optimize: the maximum reaching-time, i.e., a deadline D, and k. Our reduction from HITTING SET is approximation-preserving for k and hence a $(1 - o(1)) \log(|V(G)|)$-approximation algorithm for value k is unlikely, unless P = NP. Similarly, for fixed k, the reduced instance we obtain either admits a solution with $D = 3$ or does not admit a solution, and polynomial time algorithm that approximates D for fixed k is again highly unlikely.

Our work also creates several interesting directions for future research that can be studied under the shifting operations. Instead of minimizing the maximum reaching-time of any source, we could try to minimize the average time a source needs to reach all the vertices of the graph. Another objective is to minimize the maximum, or average, time a vertex is reached by *any* of the sources; this would be desirable, for example, in the case where a company has several warehouses over a country. The dual version of the problem is intriguing as well. Here, instead of sources, we have a set of *sinks*, i.e., every vertex wants to reach them as fast as possible. This model can capture for example the scenario where the sinks correspond to hospitals. While some of our results should work for some cases of this setting, we expect that novel algorithmic techniques are required in order to tackle them.

References

[Arnborg *et al.*, 1991] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[Braunstein and Ingrosso, 2016] Alfredo Braunstein and Alessandro Ingrosso. Inference of causality in epidemics on temporal contact networks. *Scientific reports*, 6:27538, 2016.

[Casteigts *et al.*, 2021] Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, pages 1–49, 2021.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Darmann and Döcker, 2020] Andreas Darmann and Janosch Döcker. On a simple hard variant of not-all-equal 3-sat. *Theoretical Computer Science*, 815:147–152, 2020.

[Deligkas and Potapov, 2020] Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. In *Proc. of AAAI*, pages 9810–9817, 2020.

[Downey and Fellows, 2013] Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer, 2013.

[Enright and Kao, 2018] Jessica Enright and Rowland Raymond Kao. Epidemics on dynamic networks. *Epidemics*, 24:88 – 97, 2018.

[Enright and Meeks, 2018] Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: a new application for treewidth. *Algorithmica*, 80(6):1857–1889, 2018.

[Enright *et al.*, 2021a] Jessica A. Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *J. Comput. Syst. Sci.*, 119:60–77, 2021.

[Enright *et al.*, 2021b] Jessica A. Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *J. Comput. Syst. Sci.*, 115:169–186, 2021.

[Kempe *et al.*, 2002] 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.

[Klobas *et al.*, 2022] Nina Klobas, George B Mertzios, Hendrik Molter, and Paul G Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241, page 62, 2022.

[Li *et al.*, 2018] Lei Li, Kai Zheng, Sibo Wang, Wen Hua, and Xiaofang Zhou. Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory. *The VLDB Journal*, 27(3):321–345, 2018.

[Molter *et al.*, 2021] Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 76:1–76:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Thejaswi and Gionis, 2020] Suhas Thejaswi and Aristides Gionis. Restless reachability in temporal graphs. *arXiv preprint arXiv:2010.08423*, 2020.

[Wang *et al.*, 2015] Sibo Wang, Wenqing Lin, Yi Yang, Xiaokui Xiao, and Shuigeng Zhou. Efficient route planning on public transportation networks: A labelling approach. In *Proc. of SIGMOD*, pages 967–982. ACM, 2015.

[Whitbeck *et al.*, 2012] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *Mobicom*, pages 377–388, 2012.

[Wu *et al.*, 2012] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proc. VLDB Endow.*, 5(5):406–417, 2012.

[Wu *et al.*, 2014] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. of VLDB Endowment*, 7(9):721–732, 2014.

[Wu *et al.*, 2016] Huanhuan Wu, Yuzhen Huang, James Cheng, Jinfeng Li, and Yiping Ke. Reachability and time-based path queries in temporal graphs. In *Proc. of ICDE*, pages 145–156. IEEE, 2016.