Optimizing Reachability Sets in Temporal Graphs by Delaying *

Argyrios Deligkas† Igor Potapov‡

Abstract

A temporal graph is a dynamic graph where every edge is assigned a set of integer time labels that indicate at which discrete time step the edge is available. In this paper, we study how changes of the time labels, corresponding to delays on the availability of the edges, affect the reachability sets from given sources. The questions about reachability sets are motivated by numerous applications of temporal graphs in real life networks: in epidemiology the goal is to minimize the spread of an infection; in scheduling problems arising from supply networks in manufacturing the objective is to maximize coverage and productivity. We introduce control mechanisms for reachability sets that are based on two natural operations of delaying. The first operation, termed merging, is global and batches together consecutive time labels into a single time label in the whole network simultaneously. This corresponds to postponing all events until a particular time. The second, imposes independent delays on the time labels of every edge of the graph. We provide a thorough investigation of the computational complexity of different objectives related to reachability sets when these operations are used. For the merging operation, representing global lockdown effect, we prove NP-hardness results for several minimization and maximization reachability objectives, even for very simple graph structures. For the second operation of independent delays, we prove that the minimization problems are NP-hard when the number of allowed delays is bounded. We complement this with a polynomial-time algorithm for minimising the reachability set in case of unbounded delays.

Keywords: Temporal Graphs; Reachability Sets; Optimization; NP-hard

1 Introduction

A plethora of real life scenarios can be modelled as a dynamic network that changes over time. These scenarios range from train, bus, and distribution schedules, and livestock movements between farms, to virus spreading. Many of these dynamic networks consist of a fixed set of nodes and what changes over time is the connectivity between pairs of them; the locations of the stations, distribution centers, and farms remain the same over time, while the connections between any two of them can change every few minutes, hours, or days. An equivalent way to see these networks is as a sequence of static networks that change according to a predetermined, known in advance, schedule. These types of networks, known as *temporal graphs*, were formalized in the seminal work of [23]. Since then, there is flourish of work on temporal graphs [10, 11, 19, 25, 26, 41]. Formally, in a temporal graph every edge is assigned a set of integer time-labels that

*A preliminary conference version of this work appeared in Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI), New York, New York, USA, 2020.

†Department of Computer Science, Royal Holloway University of London, UK. Email: argyrios.deligkas@rhul.ac.uk. This work was done while the author was a postdoc at Materials Innovation Factory at the University of Liverpool, funded by LRC.

‡Department of Computer Science, University of Liverpool, UK. Email: potapov@liverpool.ac.uk. Research partially funded by the grant EP/R018472/1 "Application driven Topological Data Analysis" and EP/R005613/1 "ALGOUK - A Network for Algorithms and Complexity in the UK"

indicate at which discrete time steps the edge is available. In other words, a temporal graph is a schedule of edge sets $E_1, E_2, \ldots, E_t$ over a fixed set of vertices.

In this paper, we study questions related to *reachability sets* on temporal graphs. The reachability set of vertex v is the set of vertices that there exist *temporal paths* from v to them. Informally speaking, a (strict) temporal path is a path whose edges, additionally to the usual connectivity, use (strictly) increasing time labels [39]. However, in contrast to static graphs, temporal paths do not preserve transitivity. As a result, some well-known concepts from graph theory, like Menger's theorem, do not hold for temporal graphs and have to be redefined [2,20,24].

Reachability sets emerge naturally in a wide range of models and real life applications [3, 30, 32]. One of them, related to temporal graphs, is the minimization of spread of infectious diseases over contact networks. The recent case of coronavirus COVID-19 once again reveals the importance of effective prevention and control mechanisms [1]; "*lockdown*" and "*self-isolation*" were the main prevention mechanisms in this case. Data provide significant evidence that commuter patterns and airline flights in case of infectious diseases in humans [7, 12], and livestock movements between farms for the animal case [27] could spread infectious diseases. In particular, the impact of animal movements has been extensively studied within the epidemics research community. Recent studies have shown that restricting of animal movements is among the most effective methods for controlling the spread of an infection [8, 22, 34–36] and changes in the network of animal movements between farms (nodes in a graph) can significantly decrease the spread of an infection [21, 28].

Contrary to the applications above, there are cases where we wish to maximize the reachability sets under some lockdown restrictions. Consider for example a distribution schedule, where the use of the underlying network for every day comes at a fixed cost. Then, the goal is to optimally utilize the infrastructure of the network by choosing which days to use the network in order to achieve maximum reachability while maintaining low cost. Another example is when a logistics company *has* to give a certain number of days off to its drivers. These days off can be thought as "lockdowns", since no deliveries can be made during this time. On the other hand, it wishes to maximize the number of depots it can reach in order to minimize the negative impact of the lockdowns.

The importance of these problems combined with their inherit temporal nature, made temporal graph theory a tool for analyzing epidemiology in (animal) networks [6, 15–18, 31, 37, 38]. In [18] and [17], the authors studied how reachability sets on temporal graphs change, when the schedule of the edges can change according to specific operations. More specifically, they studied MinMaxReach and MaxMinReach problems where the goal, respectively, was to minimize the maximum, or to maximize the minimum size of reachability sets for a given set of sources. In [18] these objectives were studied under the operation of *reordering* of the edge sets and it was proven that both problems are NP-hard. For MinMaxReach, [17] studied the operations of deletion of whole edge sets, or individual edges within an edge set. It was proven that MinMaxReach under both notions of deletion is NP-hard, but they admit an FPT algorithm when parameterized by the treewidth of the network and the maximum permitted number of reachable vertices. Very recently, the authors in [29] compared delaying operation with deleting operation. In addition, they derived a fixed parameter tractable algorithm for MinReach and they showed that the problem can be optimally solved in polynomial time when the underlying graph is a tree; the objective in MinReach is to minimize the number of vertices reachable by a set of sources.

Although optimization of reachability sets capture important real life problems, some of the proposed solutions are not completely satisfying due to big changes in infrastructure. For

Problem	Graph Class	Sources	Labels/ Edge	Edges/ Step
MinReach	Path	$O(n)$	1	3
MinReach MinMaxReach MinAvgReach	Tree $\Delta = 3$	1	1	1
MaxReach	Path	$O(n)$	1	4
MaxReach MaxMinReach MaxAvgReach	Bipartite $\Delta = 3$	1	1	4
MaxReach MaxMinReach MaxAvgReach	Tree $\Delta = 3$	1	1	8

Table 1: NP-hardness results for the λ-merge operation for any $\lambda \geq 2$. Δ denotes the maximum degree of the graph; Labels/Edge denotes the maximum number of labels an edge can have, i.e., how many times it is available; Edges/Step denotes the maximum number of edges available at any time step.

example, reordering of edge sets can be difficult, costly, or even impossible to perform due to the physical constraints of the network, or due to the number of changes required in the existing infrastructure. For edge-deletions, and delay-based operations studied in this paper, an upper bound on the number of allowed deletions is required. This is crucial, since the deletion of every edge trivially minimizes the reachability sets, but makes the existing network infrastructure useless. In addition, edge deletions can create a "bottleneck" problem in the network even if their number is bounded. The deletion of an edge can create a sink to the network, or completely isolate some of its parts. Instead, we wish to study the following problem.

> *Given a temporal graph and a set of sources, optimize the size of the reachability set of the sources using only* "natural" *and* "infrastructure-friendly" *operations.*

Natural operations should be intuitive and easy to apply; deletion, or postponement, of a temporal edge can naturally happen. On the other hand, bringing forward a temporal edge cannot be always feasible due to physical constraints and infrastructure constraints. Infrastructure-friendly operations should not be too difficult to perform and should not require many changes to the given network and temporal schedule.

Our contribution. Our contribution is twofold. Firstly, we introduce and study two operations, *merging* and *delaying*, that are natural and infrastructure-friendly and were not studied in the past. Merging is reminiscent of lockdown, where all planned temporal actions are postponed globally and delaying is reminiscent of lower tier restrictions, where only specific individual temporal actions might be postponed. Merging operation is parameterized by λ and it batches together λ consecutive edge sets; a λ-merge on $E_1, \dots E_\lambda$ changes the first $\lambda - 1$ edge sets to the empty sets, and the λ-th edge set is the union of all λ edge sets. The delaying operation independently delays a temporal edge; a δ-delay on the label i of edge uv changes it to $i + \delta$. In contrast to deletion of temporal edges that can directly isolate vertices, our operations isolate

some vertices *only temporarily*. Our second contribution is a thorough investigation of the computational complexity of reachability objectives under these operations. In particular, we study MinMaxReach, MinReach, and MinAvgReach where given a set of sources S the objective is as follows.

- In MinMaxReach the goal is to minimize the number of reachable vertices from any vertex in S.
- In MinReach the goal is to minimize the number of reachable vertices from S.
- In MinAvgReach the goal is to minimize the average number of reachable vertices from any vertex in S.

With respect to maximization objectives, we study MaxReach, MaxMinReach, and MaxAvgReach. We proved that these problems are NP-hard under the merging operation even for very restricted classes of graphs; see Table 1. Our reductions allow us to derive hardness results for a variety of settings that emerge in real life. More specifically, we can prove NP-hardness for directed acyclic graphs (DAGs) and unit-disk graphs. Furthermore, for the delaying operation, we studied the minimization problems. They remain NP-hard when we *bound* the number of times we are allowed to use this operation. We complement these results with a polynomial time-algorithm for the case of the unbounded number of delays that works for any δ.

2 Preliminaries

Temporal Graphs. A *temporal graph* $\langle G, \mathcal{T} \rangle$ is a pair of a (directed) graph $G = (V, E)$ and a function $\mathcal{T}$ that maps every edge of G to a list of time steps at which the edge is available. The maximum time step, $t_{\max}$, on the edges of G defines the *lifetime* of the temporal graph. Another interpretation of a temporal graph, which is useful in our case, is to see it as a sequence of subgraphs, or edge-sets, $E_1, E_2, \ldots, E_{t_{\max}}$ of G, known in advance and defined by the function $\mathcal{T}$; at time step t function $\mathcal{T}$ defines a set $E_t \subseteq E$ of edges available for this time step. We will say that an edge has label i, if it is available at time step i. The size of a temporal graph $\langle G, \mathcal{T} \rangle$ is $|V| + \sum_{t \leq t_{\max}} |E_t|$. A *temporal path* in $\langle G, \mathcal{T} \rangle$ from vertex v_1 to vertex v_k is a sequence of edges $v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$ such that each edge v_iv_{i+1} is available at time step t_i and $t_i \leq t_{i+1}$ for every $i \in [k-1]$ and no vertex is visited more than once. A temporal path is *strict* if $t_i < t_{i+1}$. In what follows, unless stated otherwise, we only consider strict temporal paths.

Reachability Sets. A vertex u is *reachable* from vertex v if there exists a temporal path from v to u. We assume that a vertex is reachable from itself. It is possible that u is reachable from v, but v is not reachable from u. The reachability set of v, denoted $\texttt{reach}(v, \langle G, \mathcal{T} \rangle)$, is the set of vertices reachable from v in $\langle G, \mathcal{T} \rangle$ at time $t_{\max}$. Given a temporal graph with lifetime $t_{\max}$ and a time step $t < t_{\max}$, the set $\texttt{reach}_t(v, \langle G, \mathcal{T} \rangle)$ contains all the vertices reachable from v by time t. The set $\texttt{reach}(v, \langle G, \mathcal{T} \rangle)$ can be computed in polynomial time with respect to the size of $\langle G, \mathcal{T} \rangle$; it suffices to check whether there exists a temporal path from v to every vertex in V, which can be done efficiently in polynomial time [40].

Merging. A *merging* operation on $\mathcal{T}$ postpones some of the edge-sets in a particular way. Intuitively, a merging operation "batches together" a number of consecutive edge-sets, making them all available at a later time step; see Figure 1 for an example.

Figure 1: (a) A temporal graph where $E_1 = \{xy\}$ and $E_2 = \{yz\}$; vertex z is reachable from x. (b) The resulting graph after merging E_1 with E_2; vertex z is no longer reachable from x.

Definition 1 (λ-merge) *For every positive integer λ, a λ-merge of edge sets $E_i, E_{i+1}, \ldots, E_{i+\lambda-1}$, replaces $E_{i+\lambda-1} = \bigcup_{i \leq j \leq i+\lambda-1} E_j$ and $E_j = \emptyset$ for every $i \leq j < i + \lambda - 1$.*

Thus, a λ-merge, or equivalently a merge of size λ, at time step i corresponds to the global delay of events from time steps $i, i+1, \ldots, i+\lambda-1$ until time step $i+\lambda-1$. We say that two λ-merges are *independent*, if they merge $E_i, \ldots E_{i+\lambda-1}$ and $E_j, \ldots E_{j+\lambda-1}$ respectively, and $i + \lambda - 1 < j$. [1] When it is clear from the context, instead of writing that we merge E_i with E_{i+1} we will write that we merge i with $i+1$.

Definition 2 ((λ, μ)-merging scheme) *For positive integers λ and μ, a* (λ, μ)-merging scheme *applies μ independent merges on $E_1, E_2, \ldots, E_{t_{\max}}$, where every merge has size at most λ. A merging scheme is* maximal *if there is no other feasible λ-merge available after applying all the other merges in the scheme.*

A (λ, μ)-merging scheme for a temporal graph $\langle G, \mathcal{T} \rangle$ essentially produces a new temporal graph by modifying the schedule $\mathcal{T}$ using μ independent merges of size at most λ. We will use $\mathcal{T}^M_{(\lambda,\mu)}$ to denote the modified schedule according to the merging scheme (λ, μ) and $\langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle$ the corresponding modified temporal graph. A merging scheme is *optimal* for an objective if it optimizes the reachability sets with respect to this objective.

Our goal is to compute merging schemes that optimize some objectives regarding reachability sets from a given set of vertices. The input to the problems we study consists of a temporal graph $\langle G, \mathcal{T} \rangle$, two positive integers λ, μ, and a subset of vertices $S \subseteq V$ which we will term *sources*. The objectives MinReach, MinMaxReach, MinAvgReach, MaxReach, MaxMinReach, MaxAvgReach we study under (λ, μ)-merging schemes are formally defined in the second column of Table 2.

Note that *any* merging operation monotonically decreases the size of the reachability set for *any* set of sources. This does not impose any constraints when we are studying minimization objectives, since every extra merge can improve our objective. However, when we are studying maximization objectives under a (λ, μ)-merging scheme we will require that the merging scheme will perform *at least* μ independent merges of length λ each. This is necessary because the reachability is trivially maximized when we do not perform any merging.

Delaying. While merging operations affect globally the whole graph, *edge delays* affect only one label of an edge. We parameterize the delay operation by δ; the maximum delay we can impose on a label of an edge. Hence, a δ-delay on edge uv at time step i changes the label i to

[1] Independent merges (delays) are commutative in respect to their application. For example, let us consider a temporal graph with 4 nodes $x_1 \xrightarrow{1} x_2 \xrightarrow{2} x_3 \xrightarrow{3} x_4$ and two non-independent merges E_1, E_2 and E_2, E_3. Following the order of E_1, E_2 and then E_2, E_3 we will have a graph $x_1 \xrightarrow{3} x_2 \xrightarrow{3} x_3 \xrightarrow{3} x_4$ and the alternative order of merges E_2, E_3 and then E_1, E_2 will give $x_1 \xrightarrow{2} x_2 \xrightarrow{3} x_3 \xrightarrow{3} x_4$, so it creates some ambiguity for non-independent merges.

Problem	$(\lambda, \mu) -$ Merging	$(\delta, \kappa) -$ Delaying
1. MinReach	$\min \lvert \bigcup_{v \in S} \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\min \lvert \bigcup_{v \in S} \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$
2. MinMaxReach	$\min \max_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\min \max_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$
3. MinAvgReach	$\min \sum_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\min \sum_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$
4. MaxReach	$\max \lvert \bigcup_{v \in S} \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\max \lvert \bigcup_{v \in S} \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$
5. MaxMinReach	$\max \min_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\max \min_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$
6. MaxAvgReach	$\max \sum_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^M_{(\lambda,\mu)} \rangle) \rvert$	$\max \sum_{v \in S} \lvert \texttt{reach}(v, \langle G, \mathcal{T}^D_{(\delta,\kappa)} \rangle) \rvert$

Table 2: Problems 1 - 3 are minimization problems, while Problems 4 - 6 are maximization problems. If $|S| = 1$, then the solution for all minimization problems is the same; similarly for the maximization problems.

$i' \leq i + \delta$. A (δ, κ)-delaying scheme applies at most κ δ-delay operations on a schedule $\mathcal{T}$. We will denote $\mathcal{T}^D_{(\delta,\kappa)}$ the produced schedule. If we allow an unbounded number of delays, we will omit κ and we will simply refer to such a scheme as δ-delaying. The objectives we study are defined at the third column of Table 2.

Max2Sat(3). To produce many of our results we use the problem Max2Sat(3). An instance of Max2Sat(3) is a CNF formula ϕ with n Boolean variables and m clauses, where each clause involves exactly two variables and every variable appears in at most three clauses. The goal is to maximize the number of satisfied clauses. Without loss of generality we will assume that every variable in ϕ appears exactly one time as a positive literal and at most two times as a negative literal. In [5] it was proven that Max2Sat(3) is NP-hard and that it is even hard to approximate better than 1.0005.

3 Merging: Minimization problems

In this section we study minimization problems under merging operations. To begin with, we prove that MinReach under $(2, \mu) -$ Merging is NP-hard even when G is a path with many sources. Then, using this result, we explain how to get NP-hardness for any $\lambda \geq 2$. Next, we show how to extend our construction and get a bipartite planar graph of maximum degree 3 and only one source, and thus we prove NP-hardness for MinReach, MinMaxReach, and MinAvgReach. Our next result is NP-hardness for the same set of problems on trees with one source. For this result, we derive a new construction. Although all of our results are presented for undirected graphs, they can be trivially extended for directed graphs by simply adding both directions to the edges. However, we can extend them for directed acyclic graphs (DAGs) too. For each one of our constructions we show how to get a DAG.

3.1 MinReach on paths

Construction. We will reduce from Max2Sat(3) considering a CNF formula ϕ with n Boolean variables and m clauses. The k-th clause of ϕ will be associated with the time steps $4k, 4k{+}1$ and $4k{+}2$, while the i-th variable will be associated with the time steps $M{+}4i, M{+}4i{+}1$

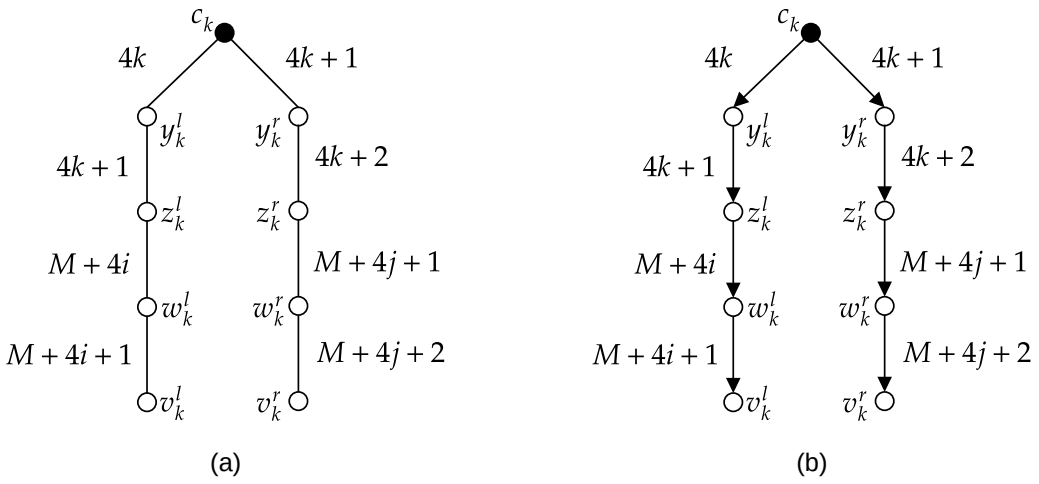

Figure 2: (a) The gadget for the clause $(\bar{x}_i, x_j)$. The labels on the edges denote the time steps they are available; $M = 4m + 4$. The solid vertex, c_k, is a source, i.e., it belongs to S. (b) The gadget for the directed case. The only difference from Subfigure (a) is the addition of the directions that can be used to prove NP-hardness in DAGs.

and $M+4i+2$, where $M = 4m+4$. For every clause of ϕ we construct a path with nine vertices, where the middle vertex is a source. Consider the path for the k-th clause, that involves the variables x_i and x_j. An example of such a path can be found on Figure 2. The middle piece of the path consists of the vertices $z_k^l, y_k^l, c_k, y_k^r, z_k^r$; where c_k belongs to a set of sources, e.g., $c_k \in S$. Edge $c_k y_k^l$ has the label $4k$, edges $y_k^l z_k^l$ and $c_k y_k^r$ have the label $4k+1$, and edge $y_k^r z_k^r$ has the label $4k+2$. The labels on the left and the right pieces of the path depend on the literals of the variables of the clause. If variable x_i appears in the clause with a positive literal, then we pick an arbitrary side of the path, say the left, and add the label $M + 4i + 1$ to the edge $z_k^l w_k^l$ and the label $M + 4i + 2$ to the edge $w_k^l v_k^l$. If x_i appears in the clause with a negative literal, then we add the label $M + 4i$ to the edge $z_k^l w_k^l$ and the label $M + 4i + 1$ to the edge $w_k^l v_k^l$. In order to create a single path, for every $k \in [m-1]$ we connect vertex v_k^r with v_{k+1}^l with a edge with label $4k$. Observe that these edges do not affect the reachability of any merging scheme, since there is no temporal path from any source that uses any of these edges. To see why this is the case, observe that the edges adjacent to edge $v_k^r v_{k+1}^l$ are available at a time step later than $M > k$. Hence, in what follows, we can safely ignore these edges from our analysis.

Observe that for the constructed temporal graph, $\langle G, \mathcal{T} \rangle$, the following hold:

1. $\langle G, \mathcal{T} \rangle$ has $9m$ vertices and lifetime $4m + 4n + 6$.
2. Every vertex is reachable from one of the sources.
3. At every time step there are at most three edges available.
4. Every edge has only one label, i.e., it is available only at one time step.

Clearly, the size of $\langle G, \mathcal{T} \rangle$ is polynomial to the size of ϕ and $|S| = m$. We will ask for a $(2, m + n)$-merging scheme, i.e., we ask for $m + n$ merges of size 2.

Intuition. The correctness of our reduction relies on two ideas. The first idea is that in every subpath, under any $(2, \mu)$-merging scheme, *at most* four vertices are not reachable from S. In order to make four vertices not reachable within a gadget, the following synergy must happen. Vertex c_k should choose which side of the path to "save"; merging $4k$ with $4k+1$ makes the vertices z_k^l, w_k^l, v_k^l unreachable; merging $4k+1$ with $4k+2$ makes the vertices z_k^r, w_k^r, v_k^r unreachable. Observe that only one of the two merges can happen, since the two merges together are not independent. Hence, such a merge makes three vertices of one side unreachable. In order to make the v-vertex of the other side unreachable, one extra merge has to happen. This merge will be translated to a truth assignment for a variable, which is the second idea of our reduction. The merge of $M+4i$ with $M+4i+1$ corresponds to setting x_i to False and the merge of $M+4i+1$ with $M+4i+2$ to True. Note, that $M+4i+1$ can be merged *only* with one of $M+4i$ and $M+4i+2$, since the merge of $M+4i$ with $M+4i+1$ and the merge of $M+4i+1$ with $M+4i+2$ are *not* independent.

Lemma 1 *If there exists an assignment that satisfies ℓ clauses of ϕ, then there exists a $(2, n+m)$-merging scheme such that $3m+\ell$ vertices are not reachable from S.*

Proof. Given an assignment we produce a merging scheme as follows.

- If variable x_i is False, then merge $M+4i$ with $M+4i+1$.
- If variable x_i is True, we merge $M+4i+1$ with $M+4i+2$.

Then, we consider every sub-path that corresponds to a clause that is satisfied and we check the side of this path that is no longer reachable from c_k, i.e., the side where the path was "cut" with respect to reachability from source c_k. For the k-th clause we distinguish the following cases:

- if there is a cut from the side where the edge with label $4k$ lies, then we merge $4k+1$ with $4k+2$;
- if the cut is from the side where the edge $4k+2$ lies, then we merge $4k$ with $4k+1$;
- if there are cuts in both sides, or there are no cuts then we arbitrarily make one of the two merges.

It is not hard to check that under this merging scheme the following holds. In every gadget that corresponds to a satisfied clause we have four vertices not reachable (we have ℓ such clauses, so 4ℓ vertices are not reachable), and in every gadget that corresponds to a non satisfied clause only three vertices are not reachable (we have $m-\ell$ such clauses, so $3m-3\ell$ vertices are reachable in this type of gadgets). Hence, given an assignment that satisfies ℓ clauses, we have made $m+n$ of 2-merges and there are $3m+\ell$ unreachable vertices. ■

Lemma 2 *If there exists an optimal $(2, n+m)$-merging scheme such that $3m+\ell$ vertices are not reachable from S, then there exists an assignment for ϕ that satisfies ℓ clauses.*

Proof. Firstly, observe that for the time steps $4k-1, 4k, 4k+1, 4k+2$, and $4k+3$, for every $k \in [m]$, no more than one 2-merge is necessary. This is because at time steps $4k-1$ and $4k+3$ there are no edges available, hence any merge that includes these time steps does not change the set of reachable vertices. In addition, this merge has to involve time step $4k+1$, otherwise it is meaningless as we argued above. The same holds for the time steps $M+4i-1, M+4i, M+4i+1, M+4i+2$, and $M+4i+3$ for every $i \in [n]$. Hence, in any optimal merging scheme there

is only one 2-merge for every triple $4k, 4k+1, 4k+2$, where $k \in [m]$, and only one 2-merge for every triple $M+4i, M+4i+1, M+4i+2$, where $i \in [n]$. So, under any optimal merging scheme, for every k, in the k-th gadget at least three vertices are not reachable (due to the merge that involves $4k+1$). In addition, at most four vertices are not reachable due to a, potential, extra merge that makes a v-vertex not reachable.

In order to create a truth assignment for the variables of ϕ, we proceed as follows. Consider the k-th gadget that has four vertices that are not reachable under the merging scheme we consider and assume that involves the labels $M+4i$, $M+4i+1$, and $M+4i+2$. If $M+4i+1$ is merged with $M+4i$, then we set x_i to False. If $M+4i+1$ is merged with $M+4i+2$, then we set x_i to True. By the construction of the gadget, observe that x_i satisfies the k-th clause. Any variables with undefined value, we set them to True. So, if there are ℓ gadgets with four unreachable vertices, the constructed assignment satisfies ℓ clauses of ϕ. To complete our proof, we need to argue that the truth assignment for the variables is well-defined, i.e., we did not set x_i both to True and False. For contradiction assume that the value of x_i is not well-defined. This means that $M+4i+1$ is merged with $M+4i$, and that $M+4i+1$ is merged with $M+4i+2$ as well; a contradiction. ■

The combination of Lemmas 1 and 2 already yield NP-hardness for MinReach under $(2,\mu)-$Merging, when μ is part of the input.

Corollary 1 MinReach *is* NP*-hard under* $(2,\mu)-$Merging.

In the next theorem we explain how to get stronger NP-hardness results for MinReach for a number of different parameters.

Theorem 1 MinReach *under* $(\lambda,\mu)-$Merging *is* NP*-hard for any* $\lambda \geq 2$*, even when* G *is a path, and the following constraints hold:*

1. *every edge is available only at one time step;*
2. *at any time step there are at most three edges available;*
3. *the merging scheme is maximal.*

Proof. We start from the temporal graph $\langle G, \mathcal{T}\rangle$ constructed above. Let $E_1, E_2, \ldots, E_{M+4n+2}$ be the edge-sets of $\langle G, \mathcal{T}\rangle$. We will construct a new temporal graph $\langle G, \mathcal{T}'\rangle$ as follows with edge-sets $E'_1, E'_2, \ldots$ where $E'_k = E_j$ if $k = \lambda \cdot j - 2$ for some $j \in \{1, 2, \ldots, M+4n+2\}$, and $E'_k = \emptyset$ otherwise. Observe that in $\langle G, \mathcal{T}'\rangle$, any λ'-merge with $\lambda' < \lambda$ does not affect the set reachable from S since it can merge at most one time step that contains edges from G. Hence, we can replace the 2-merges with λ-merges in Lemmas 1 and 2 and get the proofs exactly in the same way as before. It is not hard to verify that indeed every edge has only one label, i.e., it is available only at one time step, and that at most three edges per time step are available.

In addition, observe that in the proofs of Lemmas 1 and 2 we have not used anywhere the fact that at most μ of 2-merges were allowed. In other words, the hardness does not come from constraining the number of allowed 2-merges, but from the way the 2-merges can be placed. Hence, in Lemma 1 we can arbitrarily extend the produced merging scheme to a maximal merging scheme. In Lemma 2, without loss of generality we can assume that the merging scheme is maximal, since this does not affect the correctness of the lemma. ■

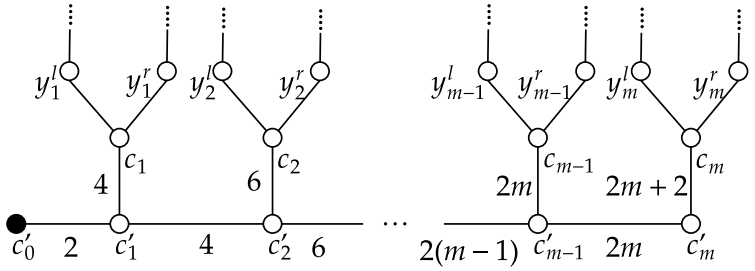

Figure 3: The construction for graphs with one source with $\lambda = 2$.

3.2 Graphs with a single source

Next, we explain how to get NP-hardness for the case with a single source, e.g., where $|S| = 1$. This result immediately implies NP-hardness for MinMaxReach and MinAvgReach under $(\lambda, \mu) -$ Merging, since when $|S| = 1$ these problems coincide with MinReach. It is not hard to get this result, given the previous construction. We can simply add a vertex c_0 in the constructed graph that is the only source and it is connected with every vertex that used to be a source, so c_0 is connected with all the c-vertices from the previous construction, with an edge available at time step 1. Since the next time step that an edge exists in is time step 4, this means that under *any* $(2, \mu)$-merging scheme, at time step 2 every vertex of S has been reached by c_0. Hence, NP-hardness follows from the previous construction. However, as we show next, we can extend our reduction, modify $\langle G, \mathcal{T} \rangle$ as shown in Figure 3, and get a tree of maximum degree 3.

Theorem 2 MinReach, MinMaxReach, *and* MinAvgReach *under* $(\lambda, \mu) -$ Merging *are* NP*-hard for any* $\lambda \geq 2$ *even when:*

1. *there exists only one source;*
2. *G is a tree of maximum degree three and constant pathwidth;*
3. *every edge is available only at one time step;*
4. *at any time step there are at most three edges available;*
5. *the merging scheme is maximal.*

Proof. We will first prove the theorem for MinReach. Then the claims for MinMaxReach and MinAvgReach will immediately follow as all three objectives coincide when we have only one source.

We start from the instance used in the proof of Theorem 1. Let $\langle G, \mathcal{T} \rangle$ be the temporal graph constructed for the proof of Theorem 1 after we delete the edges with label 1, and let $E_1, E_2, \ldots, E_k$ denote its edge sets. Recall, the instance from Theorem 1 consists of m paths where path k has the source c_k. We will create a temporal graph $\langle G', \mathcal{T}' \rangle$ where G is a tree of maximum degree 3. Firstly, we create a path that consists of the vertices $c'_0, c'_1, \ldots, c'_m$ where c'_0 and c'_m are the ends of the path. In addition, we will add the edges $c'_i c_i$ for every $i \in \{1, 2, \ldots, m\}$. Next we define the edge sets $E_1, E_2, \ldots, E_{\lambda \cdot (m+1)+k}$.

- If $t = \lambda$, then $E'_t = \{c'_0 c'_1\}$.

- If $t = \lambda \cdot i$ where $i \in \{2, \ldots, m\}$, then $E'_t = \{c'_{i-1}c'_i, c'_{i-1}c_{i-1}\}$.
- If $t = \lambda \cdot (m+1)$, then $E'_t = \{c'_m c_m\}$.
- If $t = \lambda \cdot (m+1) + j$ where $j \in \{1, \ldots, k\}$, then $E'_t = E_j$.
- In every other case $E_t = \emptyset$.

The vertex c'_0 will be the unique source of the constructed graph. The crucial observation is that under *any* merging scheme that uses λ-merges, at time step $2m + \lambda$ every vertex c_i for $i \in [m]$ will be reached. Hence, at time step $2m + \lambda$, under any (λ, μ)-merging scheme, we get an instance that is equivalent, with respect to reachability from this time step and on, to the instance of the previous section. Then, we can use exactly the same arguments to derive the NP-hardness. ■

4 Merging: Maximization problems

In this section we prove NP-hardness for maximization problems. Before we proceed with the exposition of the results though, we should discuss some issues about maximization reachability problems and merging. Any merge weakly decreases the reachability set from the sources. Hence, while in the minimization problems, in principle, we would like to perform as many merges as possible, for maximization problems we would like to perform the minimum number of merges that are allowed. In addition, the reachability set weakly decreases with λ, i.e., the size of the merge. Hence, for maximization problems it is better to apply the smallest merge possible, i.e., perform only 2-merges. For this reason, if we get NP-hardness for $(2, \mu)$-merging schemes, then we can immediately conclude that finding the optimal (λ, μ)-merging scheme is NP-hard, for any $\lambda \geq 2$. So, for maximization problems, we *require* that *at least* μ λ-merges need to happen. This is motivated by distribution networks; the use of the network comes at a cost, thus we would like to use the network as few times as possible.

Again, we prove our results by reducing from Max2Sat(3). This time though we need to be more careful; in order to make our reduction valid, we should not allow for "dummy" merges, i.e., merges that do not change the reachable vertices from the sources. As before, we first prove NP-hardness for paths with multiple sources. Then, we modify our reduction to get NP-hardness for graphs with one source and we provide a reduction for trees with just one source.

4.1 MaxReach on paths

Construction. We will reduce from Max2Sat(3). Every variable x_i of ϕ will be associated with time steps $3i-2, 3i-1$, and $3i$. For every variable we create a path of length 5 with ends the vertices y_i^l and y_i^r; this path is depicted at Figure 4(b). In this path, the edges at the ends of the paths have label $3i$ and the rest of the edges have label $3i+1$. The paths that are created from variables will be termed *variable-paths*. Both ends of every variable-path are sources. For every clause we create a path of length 4. So, if the k-th clause involves the variables x_i and x_j, we construct a path with ends the vertices v_k^l and v_k^r, and middle the vertex c_k, which will be termed c-vertex. The variable x_i will be related to the sub-path between v_k^l and c_k and the variable x_j will be related to the sub-path between v_k^r and c_k. If x_i appears with a positive literal in the clause, then the labels on the two edges that connect v_k^r and c_k are $3i-2$ and $3i-1$ respectively; else the labels are $3i-1$ and $3i$. Figure 4(a) depicts the path for the clause $(\bar{x}_i, x_j)$. These paths will be termed *clause-paths*. Both ends of every clause-path are sources. To create

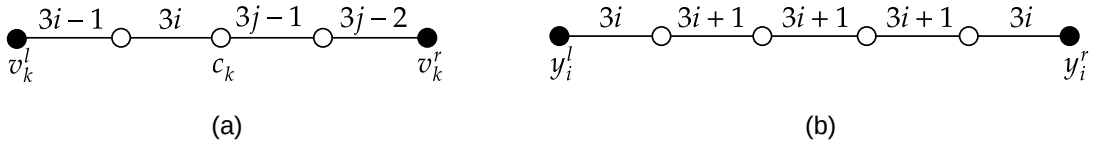

Figure 4: Gadgets for MaxReach on paths. Subfigure (a): clause-path for the clause $(\bar{x}_i, x_j)$. Subfigure (b): variable-path for variable x_i.

one path, we connect the constructed paths through their endpoints. Namely, we set $y_i^r = y_{i+1}^l$ for every $i \in [n-1]$, we set $y_n^r = v_1^l$, and we set $v_k^r = v_{k=1}^l$ for every $k \in [m-1]$.

The temporal graph $\langle G, \mathcal{T} \rangle$ we have constructed has $5n + 4m + 1$ vertices, lifetime $3n + 1$, and at any time step at most four edges available: if $t = 3i - 2$, then there are four edges; if $t = 3i - 1$, then there are three edges; and if $t = 3i$, then there are four edges. These numbers follow from our assumptions on the Max2Sat(3) instance, where every variable appears one time as positive literal and at most two times as negative literal.

Intuition. The labels on the variable-paths guarantee that there exists a $(2, n)$-merging scheme with the following properties:

- it maximizes the number of reachable vertices;
- it does not merge time step $3i$ with $3i + 1$ for any $i \in [n]$.

This guarantees two properties. Firstly, it guarantees that there exists an optimal $(2, n)$-merging scheme where all the vertices, except the c-vertices, are reachable from S. Hence, any optimal merging scheme, i.e., a merging scheme that maximizes the number of reachable vertices from S, *has* to maximize the number of reachable c-vertices. Second, given the subset of c-vertices that are reachable under the produced merging scheme, we can easily deduce a truth assignment that satisfies the clauses that correspond to the reachable c-vertices.

Lemma 3 *If there exists an assignment that satisfies ℓ clauses of ϕ, then there exists a $(2, n)$-merging scheme such that $5n + 3m + 1 + \ell$ vertices are reachable from S.*

Proof. Given an assignment for the variables of ϕ we produce a $(2, n)$-merging scheme as follows. If variable x_i is false, then we merge $3i - 2$ with $3i - 1$; else we merge $3i - 1$ with $3i$. Observe, all the vertices in the variable-paths are reachable under this merging scheme. Consider now the clause-paths. Observe that in these paths all vertices except the c-vertices are reachable. Hence, we have to argue only about the c-vertices of $\langle G, \mathcal{T} \rangle$. Take a clause-path that corresponds to a satisfiable clause; assume that this is the k-th clause that is satisfied by the variable x_i. Then, observe that the c-vertex of this path is reachable from the vertex whose side is related to x_i, say that it is v_k^r. If x_i appears as a positive literal, then the path from v_k^r to c_k uses the labels $3i - 2$ and $3i - 1$, while the merging scheme merges $3i - 1$ with $3i$. If x_i appears as a negative literal, then the path from v_k^r to c_k uses the labels $3i - 1$ and $3i$, while the merging scheme merges $3i - 2$ with $3i - 1$. Hence, we can conclude that $5n + 3m + 1 + \ell$ vertices are reachable under the produced merging scheme. ■

Lemma 4 *If there exists an optimal $(2, n)$-merging scheme such that $5n + 3m + 1 + \ell$ vertices are reachable from S, then there exists an assignment for ϕ that satisfies ℓ clauses.*

Proof. To begin with, observe that it is without loss of generality to assume that the $(2, n)$-merging scheme that maximizes the reachable set from S does not merge $3i$ with $3i + 1$ for any $i \in [n]$. For contradiction, assume that there exists an i such that every $(2, n)$-merging scheme that maximizes the reachable set from S has to merge $3i$ with $3i + 1$. Observe that this merge makes the two vertices in the middle of the i-th variable-path gadget unreachable. We will consider the following two cases for $3i - 1$.

- $3i - 1$ is not merged. Then we can "unmerge" $3i$ and $3i + 1$ and merge instead $3i - 1$ with $3i$. Thus the number of 2-merges remains the same. In addition, the two vertices in the middle of the i-th variable-path are reachable. Finally, in the worst case, the vertices c_k and $c_{k'}$ will become unreachable. These are the two vertices that correspond to the two clauses where variable x_i appears with a negative literal. Hence, we get a different $(2, n)$-merging scheme where the number of reachable vertices is not decreased and does not merge $3i$ with $3i + 1$.

- $3i - 1$ is merged. Then, there should exist a j such that $3j - 1$ that is not merged and at least one of $3j - 2$ and $3j$ is not merged; this is due to the fact that the $(2, n)$-merging scheme that maximizes the set of reachable vertices uses exactly n many 2-merges. So, again we can unmerge $3i$ and $3i + 1$ and instead add a merge that involves $3j - 1$. This way, the number of reachable vertices is not decreased. By the unmerging, the two vertices in the middle of the i-th vertex path are reachable. For the merge that involves $3j - 1$ one of the following should hold.

  - $3j-1$ is merged with $3j-2$. At most one vertex c_k will become unreachable; this is the vertex that corresponds to the clause where variable x_j appears as a positive literal. Thus, the initial $(2, n)$-merging scheme did not maximize the number of reachable vertices from S.
  - $3j - 1$ is merged with $3j$. Then, at most two vertices c_k and $c_{k'}$ become unreachable. These vertices will correspond to the clauses where x_j appears as a negative literal. Hence, we get a different $(2, n)$-merging scheme with the same number of reachable vertices that did not merge $3i$ with $3i + 1$.

From the above we can conclude that we have a $(2, n)$-merging scheme where $3i - 1$ is merged with either $3i - 2$, or $3i$, for every $i \in [n]$. So, given a $(2, n)$-merging scheme that satisfies the constraints above, we construct the truth assignment for the variables of ϕ as follows. If $3i-1$ is merged with $3i-2$, then we set x_i to False; else we set it to True. Observe that for every clause-path where vertex c_k is reachable under the merging scheme, we get that the corresponding clause is satisfied by the produced assignment; this is due to our construction. It remains to show that there exist ℓ clause-paths where their c-vertex is reachable. This is not hard to see. Firstly, all the vertices of the variable-paths are reachable under the merging scheme. Then, observe that under any $(2, n)$-merging scheme, in every clause-path all the vertices except the c-vertex are reachable; these are $3m+1$ in total. Hence, it must be true that under the assumed merging scheme ℓ many c-vertices are reachable from S. ■

Lemmas 3 and 4 imply that MaxReach under $(2, \mu)$ − Merging is NP-hard. As we have already explained, this means that the problem is NP-hard for any $\lambda \geq 2$.

Theorem 3 MaxReach *under* (λ, μ) − Merging *is* NP*-hard for any* $\lambda \geq 2$*, even when the underlying graph is a path, every edge has one label, and at any time step there exist at most four edges available.*

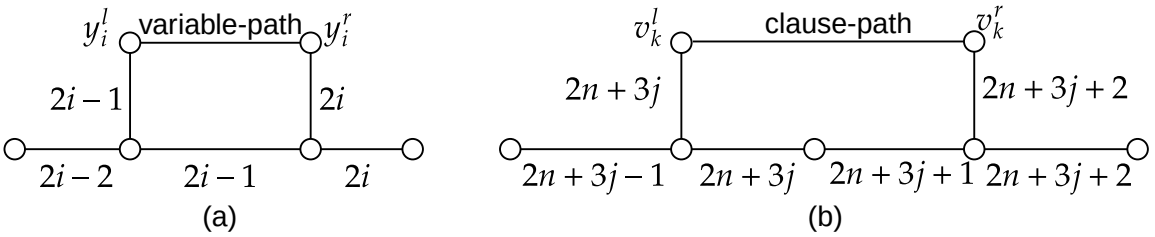

Figure 5: The gadgets used to prove NP-hardness for MaxReach in graphs with only one source. Subfigure (a) depicts how we connect a variable-path, while Subfigure (b) shows how we connect a clause-path to the path we have created.

4.2 Graphs with a single source

In this section we utilize the clause-paths and the variable-paths from the previous section and prove NP-hardness for MaxReach with only one source. Thus, we get as a corollary NP-hardness for MaxMinReach and MaxAvgReach. An easy way to prove NP-hardness for this case, would be to shift all the labels of the edges of the variable-paths and clause-paths by 1, create a vertex v_0 which would be the unique source, and connect it to every endpoint of the clause-paths and variable-paths with edges that appear at time step 1. Instead, next we provide NP-hardness for a more constrained class of graphs.

Construction. We first create a path of length $2n + 3m + 6$ that consists of vertices $w_1, \ldots, w_{2n+3m+7}$, where w_1 is the unique source of the graph. For every $i \in [2, 2n+3m+3]$, the label of edge $w_{i-1}w_i$ is i. The labels of the remaining edges are as follows: $w_{2n+3m+3}w_{2n+3m+4}$ has label $2n+3m+5$; $w_{2n+3m+4}w_{2n+3m+5}$ has label $2n+3m+8$; $w_{2n+3m+5}w_{2n+3m+6}$ has label $2n+3m+11$; $w_{2n+3m+6}w_{2n+3m+7}$ has label $2n+3m+14$. We then use the clause-paths and the variable-paths from the previous section and connect them with the path we have created. More specifically, for the i-th variable-path we connect its left endpoint, y_i^l, with vertex w_{2i-1} with an edge that has label $2i-1$ and its right endpoint, y_i^r, with vertex w_{2i} with an edge that has label $2i$. For the j-th clause-path, we connect endpoint v_k^l with vertex w_{2n+3j} with an edge that has label $2n+3j$ and endpoint v_k^r with vertex $w_{2n+3j+2}$ with an edge that has label $2n+3j+2$. Finally, for every edge that belongs to a clause-path or to a variable-path and used to have label i, in the new construction it will have the label $2n+3m+2+i$. Figure 5 depicts how exactly this is done. Observe that the constructed temporal graph $\langle G, \mathcal{T} \rangle$ has at most four edges available at any time step, G is bipartite since every cycle in the graph has even length, and every edge of G appears only at one time step.

Intuition. The high level intuition behind the following theorem is that *any* $(2, n)$-merging scheme that maximizes the vertices reachable from w_1 *does not* merge any edges with label less than $2n+3m+2$. This is because such a merge makes *at least* the last five w-vertices unreachable while any other merge makes *at most* two vertices unreachable. Hence, at time step $2n+3m+2$ all the endpoints of clause-paths and variable-paths have been reached and thus we have an instance equivalent to the instance from Section 4.1, which we know that it is NP-hard to solve.

Theorem 4 MaxReach, MaxMinReach, MaxAvgReach *are* NP*-hard under the merging operation even on temporal graphs* $\langle G, \mathcal{T} \rangle$ *where*

- *there is one source;*
- *G is bipartite or maximum degree 3;*
- *every edge appears only at one time step;*
- *at any time step there are at most four edges available.*

Proof. Consider an arbitrary $(2, n)$-merging scheme that maximizes the reachability set in the graph $\langle G, \mathcal{T} \rangle$ constructed above. Firstly, observe that this merging scheme does not merge any time step i where $i \leq 2n+3m+2$. To see why this is the case, assume for the sake of contradiction that there exists a $(2, n)$-merging scheme in which i and $i+1$ are merged, where $i \leq 2n+3m+2$. Then, observe that every vertex w_j with $j \geq i+1$ is not reachable. Hence, at least five vertices are not reachable. On the other hand, since the merging scheme has n many 2-merges, it means that there exists a $j \in [n]$ such that $2n+3m+2+3j$ is not merged with $2n+3m+2+3j+1$. If we merge these labels, we make at most two vertices unreachable. This contradicts the fact that the original merging scheme was optimal. Hence, under any optimal $(2, n)$-merging scheme we get that after time step $2n+3m+2$ every endpoint of a clause-path and a variable-path are reachable. In addition, observe that every vertex w_i with $i > 2n+3m+2$ will be reachable independently from the merging scheme chosen, as long as there are no merges before time step $2n+3m+2$. Hence, after time step $2n+3m+2$ we have a sub-instance that it is equivalent to the instance from Section 4.1; they are identical up to a shift of the labels by $2n+3m+2$. We know that MaxReach is NP-hard for this instance, thus the theorem follows. ■

4.3 Trees with one source

In this section we prove that MaxReach is NP-hard even on trees with only one source. Again, our reduction is from Max2Sat(3). First, we explain how to get NP-hardness for forests where every connected component has only one source. Then, using the idea from the previous section, we connect the components of the forest and create a single tree with only one source.

Construction. We will use a similar approach as before and we associate each variable with three consecutive time steps: variable x_i will be associated to time steps $3m+3i-2$, $3m+3i-1$, and $3m+3i$, where m is the number of the clauses. This time though we associate every clause with three consecutive time steps as well. So, the k-th clause with be associated with time steps $3k-2, 3k-1$ and $3k$.

For every clause we create a path with nine vertices where only one of them is a source; an example of this tree is depicted in Figure 6(a). Each such path consists of three pieces: the middle piece, the left piece, and the right piece. For the k-th clause, the middle piece consists of the vertices $z_k^l, y_k^l, c_k, y_k^r, z_k^r$; c_k is a source. Edge $c_k y_k^l$ has the label $3k-2$, edges $y_k^l z_k^l$ and $c_k y_k^r$ have the label $3k-1$, and edge $y_k^r z_k^r$ has the label $3k$. The left part of the path consists of the vertices w_k^l and u_k^l and the right path consists of the vertices w_k^r and u_k^r. The labels on the left and the right piece of the path depend on the literals of the variables of the clause. The left side of the path is associated with the first variable that appears in the clause while the right side of the path is associated with the second variable that appears in the clause. So, assume that x_i is the first variable that appears in the k-th clause.

- If x_i appears in the clause with a positive literal, then on the left part of the path we add the label $3m+3i-2$ to the edge $z_k^l w_k^l$ and the label $3m+3i-1$ to the edge $w_k^l u_k^l$.

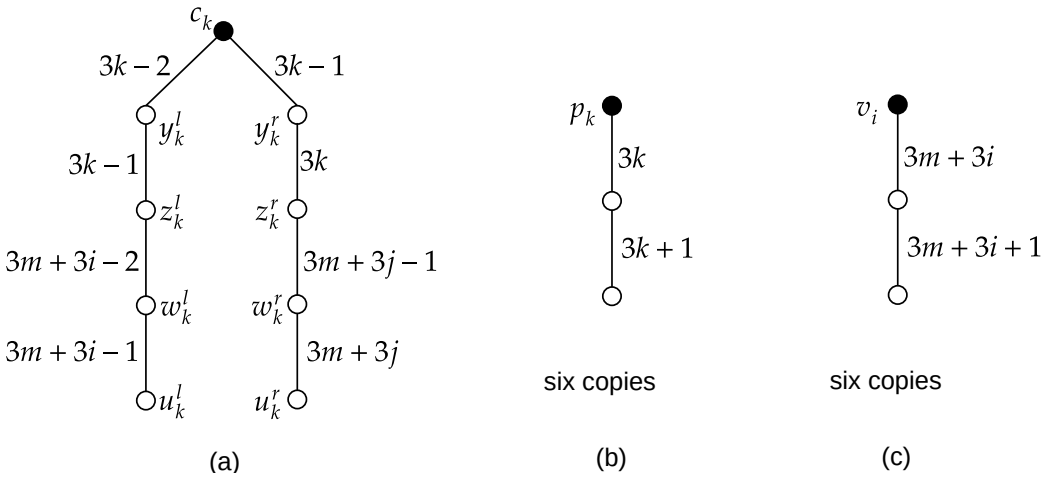

Figure 6: The gadgets used in Section 4.3 to show hardness for MaxReach on trees.

- If x_i appears in the clause with a negative literal, then we add the label $3m+3i-1$ to the edge $z_k^l w_k^l$ and the label $3m+3i$ to the edge $w_k^l u_k^l$.

If x_i is the second variable that appears in the clause, we follow the same rules for the right part of the path. In addition, we create the following. For every $k \in [m]$ we create six paths of length two; see Figure 6(b). One endpoint of every such path is a source. The labels of the edges for every path, from the source to the other end, are $3k$ and $3k+1$. For every $i \in [n]$ we create six paths of length two; the right part of Figure 6(c). One endpoint of every path is a source. The labels of the edges for every one of these six paths, from the source to the other end, are $3m+3i$ and $3m+3i+1$. The constructed temporal graph $\langle G, \mathcal{T} \rangle$ has $27m+18n$ vertices and lifetime $3m+3n+1$. Our construction guarantees that all the vertices of the graph are reachable from one source. We ask for a $(2, m+n)$-merging scheme. Observe that at any time step there exist at most eight edges available.

Intuition. The high level idea behind the construction is that in any optimal $(2, m+n)$-merging scheme all the vertices of the paths of length two are reachable; this is guaranteed by the number of copies of the paths and the choice of the labels in these paths. Hence, an optimal merging scheme maximizes the number of reachable vertices in the gadgets for the clauses. In every such gadget at most six vertices can be reached. In order to reach six vertices the following synergy must happen: the root has to choose a side to reach and then the last part of that side has to indeed not be blocked by a merge. The first step corresponds to a clause choosing which variable will satisfy it; our construction guarantees that it can choose any of the two variables. The second step will happen only if the variable has the "correct" value for the clause.

The construction and the intuition behind this result is similar in spirit with the NP-hardness result of Section 3.1. The main gadget for the clauses shares the same principle as in Section 3.1. There, vertex c_k chooses via a merge which side to "save" and an extra merge at this side must happen in order to achieve this. Here, vertex c_k chooses via a merge which side to reach and an extra merge is needed in order to achieve this.

Lemma 5 *If there exists an assignment for the variables of ϕ that satisfies ℓ clauses, then there exists a $(2, m+n)$-merging scheme for $\langle G, \mathcal{T} \rangle$ such that $22m+18n+2\ell$ vertices are reachable*

from S.

Proof. Given an assignment that satisfies ℓ clauses we create a merging scheme as follows. If x_i is True, then we merge $3m+3i-1$ with $3m+3i$, else we merge $3m+3i-2$ with $3m+3i-1$. In addition, we consider every clause separately. If the k-th clause is satisfied by its first variable, i.e., the variable that corresponds to the left piece of the gadget we created for the clause, then we merge $3k-2$ with $3k-1$; else we merge $3k-1$ with $3k$. If the clause is not satisfied, then we merge $3k-1$ with $3k$. Observe that under this merging scheme all the vertices of the length-two paths are reachable. In addition, in every path that corresponds to a clause at least three vertices between $y_k^l, y_k^r, z_k^l, z_k^r$ are reachable from c_k. Hence, for every unsatisfiable clause there are four reachable vertices. Furthermore, for every satisfiable clause, either the vertices w_k^l and u_k^l are reachable if the clause is satisfied by the first variable, or the vertices w_k^r and u_k^r are reachable if the clause is satisfied by the second variable. It is not hard to see that under this merging scheme $22m+18n+2\ell$ vertices are reachable from S. ■

To prove the other direction, we will first prove an auxiliary lemma.

Lemma 6 *Under any reachability-maximizing $(2, m+n)$-merging scheme for the constructed temporal graph $\langle G, \mathcal{T}\rangle$, the following hold:*

- *$3k-1$ is merged for every $k \in [m]$;*
- *$3m+3i-1$ is merged for every $i \in [n]$.*

Proof. In order to prove the lemma, it suffices to prove that there does not exist a reachability-maximizing $(2, m+n)$-merging scheme that merges

- $3k$ with $3k+1$ for any $k \in [m]$;
- $3m+3i$ with $3m+3i+1$ for any $i \in [n]$.

Observe that any such merge makes six vertices of $\langle G, \mathcal{T}\rangle$ unreachable; these vertices belong to a subset of paths of length two. For the sake of contradiction, assume that we have a reachability-maximizing $(2, m+n)$-merging scheme that merges $3k$ with $3k+1$; for the case where $3m+3i$ is merged with $3m+3i+1$, identical arguments apply. Then, at least one of the following cases is true:

- there exists a $k' \in [m]$, where $k' \neq k$ such that $3k'-1$ is not merged and can be merged with $3k'-2$ or $3k'$;
- there exists an $i \in [n]$, such that $3m+3i-1$ is not merged and can be merged with $3m+3i-2$ or $3m+3i$.

One of the two cases will be true due to the fact that any reachability-maximizing scheme will use exactly $m+n$ many 2-merges and the lifetime of the graph is $3m+3n+1$. If the first case is true, then we can unmerge $3k$ with $3k+1$ and merge $3k'-1$ instead, say with $3k'$. Then, we make all six vertices from the paths of length two reachable, while we make at most five vertices unreachable at the gadget that involves time step $3k'-1$: $c_{k'}$ will reach at least the vertices $y_{k'}^r, y_{k'}^l$ and $z_{k'}^l$. Thus we have increased the number of reachable vertices which contradicts the optimality of the initial merging scheme. If the second case is true, then we unmerge $3k$ with $3k+1$ and merge $3m+3i-1$ instead with one of $3m+3i-2$ or $3m+3i$. Then, the only vertices that were reachable under the previous merging scheme, but are not longer reachable under the

new merging scheme are the u-vertices of clause gadgets that are adjacent to an edge with label $3m+3i-2$ or label $3m+3i$. Observe that there are at most two edges with either of the cases. Hence, again this change increased the number of reachable vertices, a contradiction. ■

Using Lemma 6, we are ready to prove the correctness of our construction.

Lemma 7 *If there exists a reachability-maximizing $(2, m+n)$-merging scheme such that $22m+18n+2\ell$ vertices are reachable from S in $\langle G, \mathcal{T}\rangle$, then there exists a truth assignment that satisfies ℓ clauses of ϕ.*

Proof. From Lemma 6 we know that in any reachability-maximizing merging scheme $3k-1$ will be merged for every $k \in [m]$. Consider the following two cases for $3k-1$.

- If $3k-2$ is merged with $3k-1$, then the vertices z_k^l, w_k^l, and u_k^l become unreachable, while the vertices y_k^l, y_k^r, z_k^r, w_k^r, u_k^r are reachable *regardless* what the remaining merges are.
- If $3k-1$ is merged with $3k$, then the vertices z_k^r, w_k^r, and u_k^r become unreachable, while the vertices y_k^r, y_k^l, z_k^l, w_k^l, u_k^l are reachable *regardless* what the remaining merges are.

Hence, in any reachability-maximizing $(2, m+n)$-merging scheme we get that at least four vertices from every clause-gadget are reachable (including the source vertex c_k). Furthermore, the merging scheme guarantees that all the vertices that belong to paths of length two will be reached. Thus, we have $18m + 4m + 18n = 22m + 18n$ reachable vertices so far.

The remaining 2ℓ reachable vertices will define the assignment for the variables of ϕ. Recall that each clause gadget guarantees that in any reachability-maximizing scheme at most one u-vertex is reachable and in order for this to happen one extra w-vertex is reachable. So, in a reachability-maximizing scheme with $22m + 18n + 2\ell$ reachable vertices, there exist ℓ many u-vertices that are reachable. If a u-vertex is reachable and is adjacent to an edge with label $3m+3i-1$, then we set variable x_i to True; else if it is reachable and is adjacent to an edge with label $3m+3i$, then we set variable x_i to False. If there are any variables for which we have not defined their value, then we arbitrarily set them to True. Observe that this procedure sets correctly the values of the variables; no variable can get more than one value. This is because we cannot have two u-vertices that are adjacent to edges with labels $3m+3i-1$ and $3m+3i$ and both of them are reachable. At least one of them will become unreachable since $3m+3i-1$ is merged. Our construction guarantees that this assignment for x_i will satisfy all the clauses where the corresponding u-vertices are reachable. Since there are ℓ many u-vertices reachable, then we get that ℓ clauses will be satisfied. ■

The combination of the lemmas above already yields NP-hardness for MaxReach on forests. However, we can use a construction similar to one used in the proof of Theorem 4 and get the following theorem.

Theorem 5 MaxReach, MaxMinReach, MaxAvgReach *are* NP*-hard under merging operations even when there exists only one source, G is a tree of constant pathwidth, has maximum degree 3, every edge has only one label, and at any time step there are at most eight edges available.*

Proof. Figure 7 depicts the construction used in this proof. We begin by creating a path of length $7m + 6n + 10$: the path begins with m c'-vertices, then we have $6m$ vertices that are termed p'-vertices, then we have $6n$ v'-vertices, and lastly we have 10 q'-vertices. The labels are defined as follows:

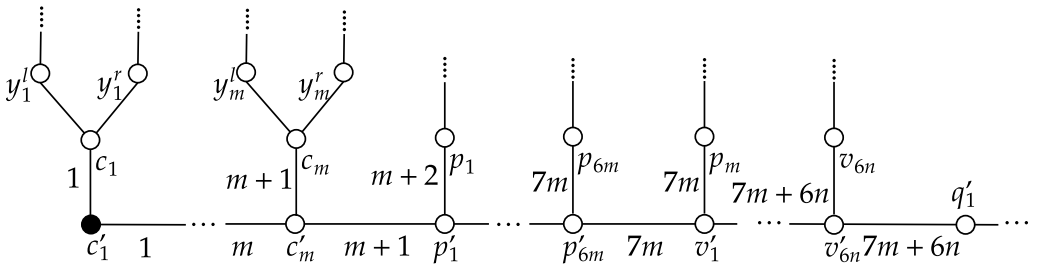

Figure 7: The tree with the one source for the MaxReach NP-hardness.

- edge $c_i'c_{i+1}'$ has label i for every $i \in [m-1]$;
- edge $c_m'p_1'$ has label m;
- edge $p_i'p_{i+1}'$ has label $m+i$ for every $i \in [3m-1]$;
- edge $p_{3m}'v_1'$ has label $7m$;
- edge $v_i'v_{i+1}'$ has label $7m+i$ for every $i \in [n-1]$;
- edge $v_{3n}'q_1'$ has label $7m+6n$;
- edge $q_i'q_{i+1}'$ has label $7m+6n+3i-1$ for every $i \in [9]$.

The unique source will be the vertex c_1'. To complete the construction, we will use the gadgets we have constructed before. For every $k \in [m]$, we add the edge $c_k'c_k$ with label k, where c_k is the vertex of the clause gadget we have created before. The i-th p'-vertex is connected with one p-vertex from the previous gadgets via an edge with label $m+i$, such that every p-vertex is connected to a p'-vertex. Furthermore, the i-th v'-vertex is connected to one v-vertex via an edge with label $7m+i$. Finally, we shift the labels of the gadgets from Figure 6 by $7m+6n$, i.e., we add $7m+6n$ on the label of the edge. For example, if an edge had the label i in the gadgets of Figure 6, now in Figure 7, it will have the label $7m+6n+i$. Observe that the constructed graph is a tree of maximum degree 3, every edge has exactly one label, and at any time step there are at most eight edges available.

The crucial observation is that any reachability-maximizing $(2, m+n)$-merging scheme for the constructed graph will not merge any $i \leq 7m+6n$ since this will make at least the ten q'-vertices unreachable, while any other merge would make at most six vertices unreachable (if we merge $M+3k$ with $M+3k+1$ for $k \in [m]$, or if we merge $M+3m+3i$ with $M+3m+3i+1$ for some $i \in [n]$, where $M = 7m+6n$). We omit this part of the proof since it is almost identical to the proof of Theorem 4. Hence, under any reachability-maximizing $(2, m+n)$-merging scheme, at time step $7m+6n$ every c-vertex, every p-vertex, and every v-vertex will be reached. In addition, all the q'-vertices will be reached under any such merging scheme, hence they do not affect the choices of the merging scheme. The correctness of the theorem follows by the observation that we have a sub-instance that is identical to the NP-hard instance we had before. ■

5 Delaying

In this section we study *edge-independent* delays. Firstly, we show that when the number of allowed delays is bounded, then the minimization problems are NP-hard. Then, we study the

case where the number of δ-delaying operations is unbounded. In contrast to unbounded edge deletions, where the solution to the problems becomes trivial by essentially isolating every source, an unbounded number of independent δ-delays, i.e., when an edge cannot be delayed more than δ time steps, does not trivialize the problems and most importantly does not destroy the underlying network.

Theorem 6 MinReach, MinMaxReach, *and* MinAvgReach *are* NP*-hard under* $\delta-$Delaying, *for any* $\delta \geq 1$, *when the number of operations is bounded by* k. *In addition, they are* $W[1]$*-hard, when parameterized by* k.

Proof. To begin with, we make the following observation that will be helpful later in the proof. For any temporal graph, delaying any edge with label $t_{\max}$ does not decrease the reachability set of any source. This is because if vertex u from edge uv is already reached by a source before $t_{\max}$, then v will be reached from uv independently from any delay on this edge. Hence, if we have a temporal graph with $t_{\max} = 2$, we can assume without loss of generality that in any optimal solution we delay edges with label 1 *only*.

Firstly, we study MinReach $(\delta, k)-$Delaying. We will prove that the problem is hard even when $|S| = 1$, i.e., there is only one source. As before, this result implies that MinMaxReach and MinAvgReach are NP-hard under delaying as well.

To prove our result, we follow a similar technique as the authors of [17] and we reduce from Clique. An instance of Clique consists of a graph $G' = (V', E')$ and an integer k. We want to decide if G' contains a clique of size k, i.e., a complete subgraph with k vertices. We construct a temporal graph $\langle G, \mathcal{T}\rangle$, where $G = (V, E)$, as follows. For every vertex and every edge of G' we create a vertex in G; we term the former ones v-vertices and the latter e-vertices. In addition, we create the vertex v^* which will be the the only source. Hence, $|V| = |V'| + |E'| + 1$. The edge-set E consists of:

- v^*v, where v is a v-vertex;
- vu, where v is a v-vertex, u is an e-vertex and the corresponding edge from E' contains the vertex v.

Every edge in G that contains v^* has label 1 and the rest of the labels have label 2. Finally, we set $\delta \geq 1$. We claim that G' contains a clique of size k if and only if there is a (δ, k)-delaying scheme such that $|V| - \frac{k(k-1)}{2}$ vertices are reachable from v^* in G.

Assume now that G' contains a clique k and let $X = \{v_1, \ldots, v_k\}$ denote the set of its vertices and let Y denote the set of e-vertices corresponding to edges of X. Then, we delay by δ all the edges v^*v, where $v \in X$. We claim that no vertex in Y is reachable from v^*. To see why this is the case, let $y \in Y$ correspond to the edge v_iv_j where $v_i \in X$ and $v_j \in X$. Observe that by the construction of $\langle G, \mathcal{T}\rangle$ there are only two temporal paths from v^* to y: $v^* \xrightarrow{1} v_i \xrightarrow{2} y$ and $v^* \xrightarrow{1} v_j \xrightarrow{2} y$. Hence, after delaying the edges v^*v_i and v^*v_j, we see that none of these paths is valid any more. Since $|Y| = \frac{k(k-1)}{2}$, we get that v^* reaches $|V| - \frac{k(k-1)}{2}$ vertices.

To prove the other direction, recall that since $\langle G, \mathcal{T}\rangle$ has lifetime 2, it suffices to delay only edges with label 1. So assume that we can delay k edges with label 1 by δ such that $|\texttt{reach}(v^*, \langle G, \mathcal{T}^D_{(\delta,k)}\rangle)| = |V| - \frac{k(k-1)}{2}$. Let $v^*v_1, \ldots, v^*v_k$ be the edges we delayed. Observe that all the v-vertices are reachable from v^*. Hence, after the delays, $\frac{k(k-1)}{2}$ e-vertices are not reachable from v^*. So, by the construction of $\langle G, \mathcal{T}\rangle$, in G' there should be $\frac{k(k-1)}{2}$ edges between the vertices $v_1, \ldots, v_k$. Hence, G' contains a clique of size k. This establishes the NP-hardness of

$\mathcal{T}_\delta^D \leftarrow \mathcal{T}$;
for $1 \leq t < t_{\max} + \delta$ **do**
  Compute $RE_t(\langle G, \mathcal{T}_\delta^D \rangle, S)$
  foreach *edge* $e \in RE_t(\langle G, \mathcal{T}_\delta^D \rangle, S)$ **do**
    Change the label i of e to $t+1$, if this is δ-possible;
    Update $\mathcal{T}_\delta^D$;
  end
end

Algorithm 1: Algorithm for $\delta -$ Delaying.

MinReach $(\delta, k) -$ Delaying. Since $|S| = 1$, the hardness for the other two problems follows. Finally, since Clique is $W[1]$-complete [14] and we have a parameterized m-reduction, we get the $W[1]$-hardness for the problems when they are parameterized by the number of allowed delays k. ■

5.1 A polynomial-time algorithm

Next, we provide a polynomial-time algorithm for the minimization problems under $\delta -$ Delaying, for any $\delta \geq 1$, when we are allowed to perform an unbounded number of δ-delays.

Intuition. The algorithm at every time step maintains the set of currently-reachable vertices and the current labeling of the edges. For every time step t we compute the vertices that became reachable in this time step and the corresponding edges that have been used to reach them. In other words, the algorithm computes the edges with label t that were used to reach some vertices in this time step. Then, it checks if it can delay these edges for another one time step, i.e., it checks if it can change their label t to $t+1$. Recall that an edge might not be allowed for further delays if it is already delayed for δ time steps. This procedure guarantees that for every time step t the number of reachable vertices is minimized, and hence it guarantees the correctness of the algorithm.

Let us define the *reachability network* which will be used by our algorithm. Given a temporal graph $\langle G, \mathcal{T} \rangle$ with lifetime $t_{\max}$ and a set of sources S, for every $t \leq t_{\max}$, we define $RV_t(\langle G, \mathcal{T} \rangle, S)$ to be the set of vertices that are reached at time t *for the first time* from a vertex in S. Put formally, $v \in RV_t(\langle G, \mathcal{T} \rangle, S)$, if for every $t' < t$ there is no path from any $s \in S$ to v that arrives at time t'. Additionally, we define $RE_t(\langle G, \mathcal{T} \rangle, S)$ to be the set of temporal edges with label t that are incident to vertices in $RV_t(\langle G, \mathcal{T} \rangle, S)$. An example of these notions is depicted in Figure 8. Observe that we can decide if $v \in RV_t(\langle G, \mathcal{T} \rangle, S)$ via computing the earliest arrival paths between every $s \in S$ and v, which can be done in time polynomial with respect to the size of $\langle G, \mathcal{T} \rangle$ [40]. Similarly, we can efficiently compute $RE_t(\langle G, \mathcal{T} \rangle, S)$. Finally, assume that the edge uv *before any delaying* had the label x and after some delays it has the label $t \leq x + \delta$. We say that it is δ-possible to change a label of uv from t to $t+1$ if the $t + 1 - x \leq \delta$; i.e., the edge is not delayed more than δ time steps.

Lemma 8 *Algorithm 1 is optimal for* MinReach, MinMaxReach, MinAvgReach *under* $\delta -$ Delaying.

Proof. To prove the lemma, we will prove by induction that at any time step $1 \leq t < t_{\max}$ Algorithm 1 minimizes $\sum_{i=1}^{t} RV_i(G, \mathcal{T}_\delta^D, S)$, i.e., it minimizes the number of reachable vertices

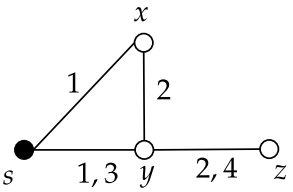

Figure 8: In the temporal graph $\langle G, \mathcal{T} \rangle$ above with $S = s$, we have $RV_1(\langle G, \mathcal{T} \rangle, s) = \{x, y\}$, $RV_2(\langle G, \mathcal{T} \rangle, s) = \{z\}$, $RE_1(\langle G, \mathcal{T} \rangle, s) = \{sx, sy\}$, and $RE_2(\langle G, \mathcal{T} \rangle, s) = \{yz\}$. Any other set is empty.

until time t. For $t = 1$ the claim clearly holds. Every vertex reachable at time step 1 will be reached unless all of the edges with label 1 adjacent to a source get delayed. Algorithm 1 indeed delays all these edges, hence the claim holds. For the induction hypothesis, assume that the lemma holds up to time step $t-1$. This means that $\sum_{i=1}^{t-1} RV_i(G, \mathcal{T}_\delta^D, S) \leq \sum_{i=1}^{t-1} RV_i(G, \hat{\mathcal{T}}_\delta, S)$ where $\hat{\mathcal{T}}_\delta$ is any other δ-delaying scheme. In order to prove the lemma it suffices to prove that $RV_t(G, \mathcal{T}_\delta^D, S) \leq RV_t(G, \hat{\mathcal{T}}_\delta, S)$; the correctness will follow from the induction hypothesis. To see why this is the case, observe that the algorithm will delay all the edges it can with label t until time step $t+1$. Hence, the set of reachable vertices will increase only by the minimum possible number; this number is dictated by the number of edges that it is not δ-possible to delay. ■

With Lemma 8 in hand we can prove the following theorem.

Theorem 7 MinReach, MinMaxReach, MinAvgReach *can be solved in polynomial time under* δ − Delaying.

Proof. Lemma 8 shows that Algorithm 1 is indeed optimal for MinReach, MinMaxReach, MinAvgReach under δ − Delaying. It remains to show that the algorithm indeed requires polynomial time with respect to the input temporal graph. This is easy to see. At every iteration the algorithm checks the labels of the edges and increases the label of a subset of them. This clearly can be done in time linear in the size of the graph. In addition, we have $t_{\max}+\delta$ iterations, hence the overall running time of the algorithm is polynomial in the size of the input temporal graph. ■

6 Discussion and Extensions

In this section we discuss some further results that can be easily derived from our NP-hardness reductions. These include inapproximability results and NP-hardness for the graph classes of directed acyclic graphs and unit-disk graphs. In addition, we highlight several challenging problems our work creates, or leaves open.

Inapproximability results. Our hardness results immediately imply, or can be easily extended to prove, several other interesting results. Firstly, we observe that all of our reductions under the merging operations are approximation preserving. Thus, since we use Max2Sat(3) in our reductions, we get that there are no approximation schemes for these problems, unless P = NP. More formally, we get that there exists an absolute constant c such that the objectives we study are NP-hard even to approximate better than c under merging operations. A natural question is to ask whether we can find a polynomial-time algorithm with constant approximation.

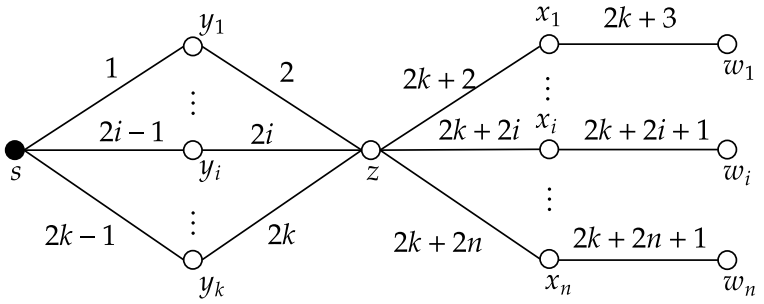

Figure 9: An example where the $(2, k)$-merging scheme that minimizes the reachability set of s, merges label $2i-1$ with label $2i$ for every $i \in [k]$. This results to a reachability set of size $k+1$; s can reach only the vertices $y_1, y_2, \ldots, y_k$. Any strict subset of these merges does not make any vertex unreachable, while any other 2-merge can make at most one vertex unreachable. This example shows that the greedy algorithm cannot work. If we choose to merge an edge that maximizes the set of unreachable vertices given the current merges we will end up with a bad approximation; consider for example the case where $k \ll n$.

> *Is there a polynomial-time algorithm for reachability objectives that achieves constant approximation under the merging operation? If not, can we get any non-trivial approximation in polynomial time?*

We believe this is a challenging question due to the temporal nature of the model and thus novel techniques are required in order to tackle it. For example, there are instances where the optimal merging scheme uses k merges and makes almost the whole graph not reachable, but any subset of these merges does *not* decrease the reachability set at all, while any other merge makes only a constant number of vertices unreachable. Figure 9 demonstrates such an example. This indicates that a good approximation algorithm cannot consider the merges independently.

Directed graphs. Another remark is that all of our constructions can be converted to be directed acyclic graphs (DAGs) without breaking the correctness of the reductions. Figure 2 shows how to add directions to the gadget used to prove NP-hardness for MinReach in paths under delaying. In the majority of our constructions there is a natural "flow", so the directions can be added trivially. For example, the directions of the edges to all tree-constructions should be towards the leaves of the tree and the unique vertex with no incoming edges is the vertex that belongs to S. The only non-trivial case is the construction depicted in Figure 4. For this construction we show in Figure 10 how to add directions to the edges without introducing any directed cycles. The correctness of the reduction is identical to the undirected case and for this reason it is omitted. So, all of hardness results hold in very constrained classes of directed graphs. However, real-life transportation networks are rarely acyclic and they may contain temporal cyclic paths. Hence, the following question can capture the scenario described above.

> *What is the complexity of* MaxReach *when there are at least k temporal paths between any two vertices?*

Large number of sources. Another aspect of our hardness reductions is that the number of sources is a constant fraction of the vertices of the graph, if not just one. A natural question

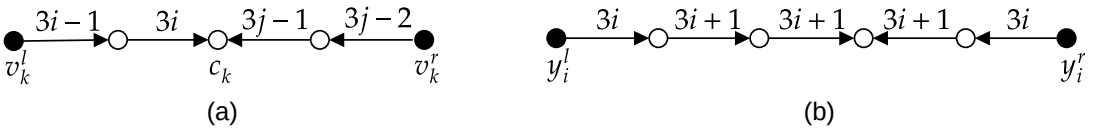

Figure 10: The directed gadgets for MaxReach on paths. Subfigure (a): clause-path for the clause $(\bar{x}_i, x_j)$. Subfigure (b): variable-path for variable x_i.

to ask then is what happens when almost all vertices are sources, i.e., we have only a constant number of *non-sources*. Our reductions cannot handle these cases. On the other hand, under this scenario, all reachability problems are far from trivial. Intuitively, MinReach and MaxReach should become easier, since we care only for a constant number of vertices. However, the multiplicity of labels on the edges can actually make the problem difficult. For the remaining reachability problems, the large number of sources actually makes the problem more complicated. An interesting question is the parameterized complexity of these problems with respect to non-sources. We conjecture that in this case there exist FPT algorithms for MinReach and MaxReach. On the other hand, the complexity of the remaining problems seems more challenging, since there is no obvious reduction between any pair of problems. In fact, we do not know the complexity of the following problems.

> *What is the complexity of* MinMaxReach *and* MinAvgReach *when all vertices are sources?*

Unit-disk graphs. A less obvious observation is that all of our NP-hardness results for merging hold even for temporal unit-disk graphs[2]. This observation is obvious for paths, but it is not completely obvious for the rest of the graph classes we have constructed, since there is no direct drawing of these *specific* graphs via unit disk graphs. However, we can get NP-hardness with some minor modifications. Figure 11 shows how to modify the construction from Figure 3 in order to get a unit-disk graph. More specifically, we slightly modify the graph by adding the q'-vertices and updating the labels as it is shown in Figure 11(b). Following exactly the same arguments as in Section 3.2, we can prove that MinReach remains NP-hard in the created instance. Then, Figure 11(c) shows how to draw this graph as a unit-disk graphs. Using similar tricks we can extend the rest of our NP-hardness results for unit-disk graphs for MaxReach; these constructions are very easy to derive and thus they are omitted.

SIS and SIR epidemic models. The susceptible-infectious-susceptible (SIS) and the susceptible-infectious-resistant (SIR) models are two well-studied models in epidemiology. There, every "infected" vertex spreads the infection to its neighbors only for a limited time after the time it was infected. In SIS a vertex can be infected more than once, while in SIR a vertex can be infected at most once. An elegant way to capture SIS and SIR models in temporal networks is via Δ-restless walks and Δ-restless paths, respectively. Recall that a temporal path is a sequence of edges $v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$ such that each edge v_iv_{i+1} is available at time step t_i, $t_i \leq t_{i+1}$ for every $i \in [k-1]$, and no vertex is visited more than once. A temporal walk allows to visit vertices more than once. A temporal path (walk) is Δ-restless if $t_{i+1} \in [t_i, t_i + \Delta]$. Δ-restless paths

[2]Recall, a graph is unit disk, if it can be drawn via a set of cycles on the plane, where every cycle corresponds to a vertex and every intersection between two cycles corresponds to an edge between the corresponding vertices.

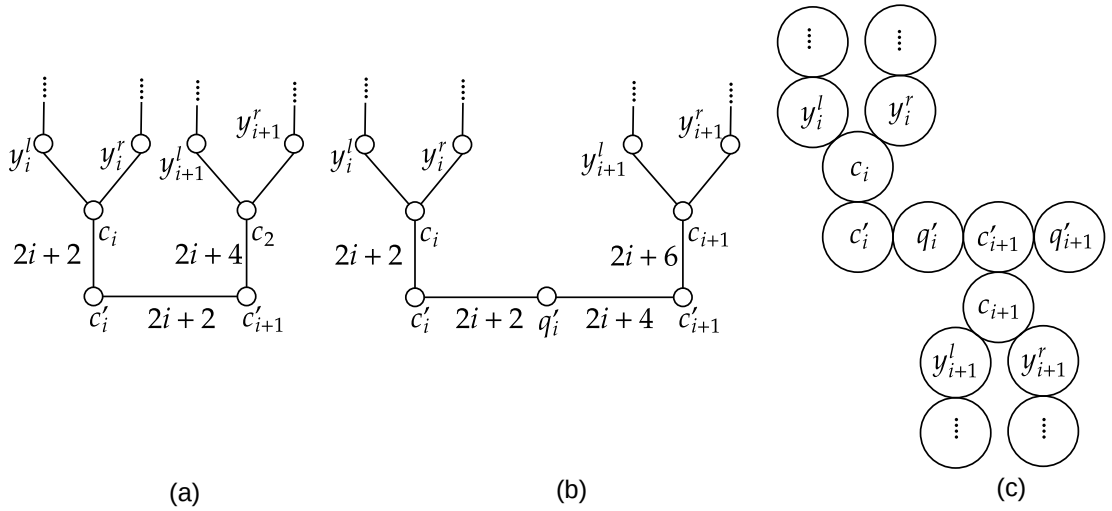

Figure 11: Subfigure (a): The gadget from MinReach on trees. Subfigure (b): The addition of the q'-vertices and the modification of the labels. Subfigure (c): Drawing of the gadget as a unit-disk graph.

were proposed and studied in [9], while Δ-restless walks were studied in [4]. There is a stark difference on the complexity of computing Δ-restless walks and Δ-restless paths: the former can be computed in polynomial time [4], while the latter is NP-hard [9] even when $\Delta = 1$. This result implies that MinReach is trivially NP-hard under Δ-restless paths, and thus MinMaxReach and MinAvgReach are NP-hard as well in this setting. However, the complexity of the problem is wide open under restless walks. To this end, we highlight the following intriguing question.

What is the complexity of MinReach *under Δ-restless walks for constant Δ?*

Non-strict temporal paths. As we have highlighted in the Preliminaries, we have studied only strict temporal paths. For the merging operation, strict temporal paths are crucial. If we consider non-strict temporal paths, then any merging operation does *not* change the reachability set of any vertex. On the other hand, our results for delaying operations hold even if we consider strict temporal paths.

Delaying and maximization objectives. For delaying we studied only the minimization problems. In a different direction, we think that maximization problems under delaying operations is another important problem to study. In this setting there is a plethora of problems to study and all of them having immediate applications to real life problems. These questions can consider only the reachability objectives we studied in this paper, or take the time needed to reach a vertex into account as well. When time is taken into account, delaying operations can help us to reach *faster* some vertices [13]; see Figure 12. However, sometimes this comes at a cost of making some other vertices unreachable. We choose to state another fundamental question related to maximum reachability which does not have an obvious answer.

Given a temporally disconnected temporal graph, can we make it connected using a bounded number of delaying operations?

Figure 12: Subfigure (a): The initial graph. x reaches z at time step 5. Subfigure (b): The delaying of label 1 to 2 for edge yz allowed x to reach z at time step 2. Observe though, this delay made x unreachable from z.

Network re-optimization. As we have seen our work creates many interesting questions for future research both related to reachability questions and to temporal graphs in general. Reachability objectives on temporal graphs can be studied under other notions of temporal paths; restless temporal paths is such an example [9, 33]. We view the our paper as a first step towards a new conceptual research direction in temporal graphs; *network re-optimization*.

> *Given a temporal network with an* existing *solution for a problem, can we utilize the current infrastructure in a better way and improve the solution without significantly changing the network?*

We believe that the answer is positive and that it deserves to be further studied.

Acknowledgements

The authors would like to thank Valentin Bura and Vladimir Gusev for several interesting discussions on reachability sets of temporal graphs at the beginning of the project. In addition, the authors would like to thank the anonymous reviewers whose insightful comments significantly helped us to improve the write-up of the paper.

References

[1] Infection prevention and control and preparedness for COVID-19 in healthcare settings. Technical report, European Centre for Disease Prevention and Control, 2020.

[2] E. C. Akrida, J. Czyzowicz, L. Gasieniec, L. Kuszner, and P. G. Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46 – 60, 2019.

[3] R. E. Allen, A. A. Clark, J. A. Starek, and M. Pavone. A machine learning approach for real-time reachability analysis. In *Proc. of IROS*, pages 2202–2208, 2014.

[4] M. Bentert, A.-S. Himmel, A. Nichterlein, and R. Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):1–26, 2020.

[5] P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proc. of ICALP*, pages 200–209, 1999.

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

[7] D. Brockmann and D. Helbing. The hidden geometry of complex, network-driven contagion phenomena. *Science*, 342(6164):1337–1342, 2013.

[8] M. G. Buhnerkempe, M. Tildesley, et al. The impact of movements and animal density on continental scale cattle disease outbreaks in the United States. *PLoS One*, 9(3):e91724, 2014.

[9] A. Casteigts, A.-S. Himmel, H. Molter, and P. Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, pages 1–49, 2021.

[10] A. Casteigts, R. Klasing, Y. M. Neggaz, and J. G. Peters. Computing parameters of sequence-based dynamic graphs. *Theory of Computing Systems*, 63(3):394–417, Apr 2019.

[11] J. Chen, H. Molter, M. Sorge, and O. Suchý. Cluster Editing in Multi-Layer and Temporal Graphs. In *Proc. of ISAAC*, volume 123, pages 24:1–24:13, 2018.

[12] V. Colizza, A. Barrat, M. Barthélemy, and A. Vespignani. The role of the airline transportation network in the prediction and predictability of global epidemics. *Proceedings of the National Academy of Sciences*, 103(7):2015–2020, 2006.

[13] A. Deligkas, E. Eiben, and G. Skretas. Minimizing reachability times on temporal graphs via shifting labels. *arXiv preprint arXiv:2112.08797*, 2021.

[14] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

[15] J. Enright and R. R. Kao. Epidemics on dynamic networks. *Epidemics*, 24:88 – 97, 2018.

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

[17] J. A. Enright, K. Meeks, G. B. Mertzios, and V. Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *J. Comput. Syst. Sci.*, 119:60–77, 2021.

[18] J. A. Enright, K. Meeks, and F. Skerman. Assigning times to minimise reachability in temporal graphs. *J. Comput. Syst. Sci.*, 115:169–186, 2021.

[19] T. Erlebach, M. Hoffmann, and F. Kammer. On temporal graph exploration. *J. Comput. Syst. Sci.*, 119:1–18, 2021.

[20] T. Erlebach and J. T. Spooner. Faster Exploration of Degree-Bounded Temporal Graphs. In *Proc. of MFCS*, volume 117, pages 36:1–36:13, 2018.

[21] M. C. Gates and M. E. Woolhouse. Controlling infectious disease through the targeted manipulation of contact network structure. *Epidemics*, 12:11–19, 2015.

[22] A. E. Jones, J. Turner, C. Caminade, A. E. Heath, M. Wardeh, G. Kluiters, P. J. Diggle, A. P. Morse, and M. Baylis. Bluetongue risk under future climates. *Nature Climate Change*, 9(2):153, 2019.

[23] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[24] G. B. Mertzios, O. Michail, and P. G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.

[25] O. Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[26] O. Michail and P. G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1 – 23, 2016.

[27] A. Mitchell, D. Bourn, J. Mawdsley, W. Wint, R. Clifton-Hadley, and M. Gilbert. Characteristics of cattle movements in Britain–an analysis of records from the cattle tracing system. *Animal Science*, 80(3):265–273, 2005.

[28] S. Mohr, M. Deason, M. Churakov, T. Doherty, and R. Kao. Manipulation of contact network structure and the impact on foot-and-mouth disease transmission. *Preventive veterinary medicine*, 157:8–18, 2018.

[29] H. Molter, M. Renken, and P. Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *In Proc. MFCS*, volume 202 of *LIPIcs*, pages 76:1–76:15, 2021.

[30] R. Niskanen, I. Potapov, and J. Reichert. On decidability and complexity of low-dimensional robot games. *Journal of Computer and System Sciences*, 107:124 – 141, 2020.

[31] M. Nöremark and S. Widgren. Epicontacttrace: an r-package for contact tracing during livestock disease outbreaks and for risk-based surveillance. *BMC veterinary research*, 10(1):71, 2014.

[32] I. Potapov. From post systems to the reachability problems for matrix semigroups and multicounter automata. In *Proc. of DLT*, pages 345–356, 2004.

[33] S. Thejaswi and A. Gionis. Restless reachability in temporal graphs. *arXiv preprint arXiv:2010.08423*, 2020.

[34] H.-H. Thulke, D. Eisinger, and M. Beer. The role of movement restrictions and pre-emptive destruction in the emergency control strategy against CSF outbreaks in domestic pigs. *Preventive veterinary medicine*, 99(1):28–37, 2011.

[35] J. Turner, R. G. Bowers, and M. Baylis. Modelling bluetongue virus transmission between farms using animal and vector movements. *Scientific reports*, 2:319, 2012.

[36] J. Turner, A. Jones, A. Heath, M. Wardeh, C. Caminade, G. Kluiters, R. Bowers, A. Morse, and M. Baylis. The effect of temperature, farm density and foot-and-mouth disease restrictions on the 2007 uk bluetongue outbreak. *Scientific reports*, 9(1):112, 2019.

[37] E. Valdano, L. Ferreri, C. Poletto, and V. Colizza. Analytical computation of the epidemic threshold on temporal networks. *Physical Review X*, 5(2):021005, 2015.

[38] E. Valdano, C. Poletto, A. Giovannini, D. Palma, L. Savini, and V. Colizza. Predicting epidemic risk from past temporal contact data. *PLoS computational biology*, 11(3):e1004152, 2015.

[39] J. Whitbeck, M. Dias de Amorim, V. Conan, and J.-L. Guillaume. Temporal reachability graphs. In *Mobicom*, pages 377–388, 2012.

[40] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. *Proc. of VLDB Endowment*, 7(9):721–732, 2014.

[41] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comput. Syst. Sci.*, 107:72–92, 2020.