The Complexity of Transitively Orienting Temporal Graphs

George B. Mertzios
Department of Computer Science, Durham University, UK

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

Malte Renken
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Computer Engineering & Informatics Department, University of Patras, Greece

Philipp Zschoche
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany

Abstract

In a *temporal network* with discrete time-labels on its edges, entities and information can only "flow" along sequences of edges whose time-labels are non-decreasing (resp. increasing), i.e. along temporal (resp. strict temporal) paths. Nevertheless, in the model for temporal networks of [Kempe, Kleinberg, Kumar, JCSS, 2002], the individual time-labeled edges remain undirected: an edge $e = \{u, v\}$ with time-label t specifies that "u communicates with v at time t". This is a symmetric relation between u and v, and it can be interpreted that the information can flow in either direction. In this paper we make a first attempt to understand how the direction of information flow on one edge can impact the direction of information flow on other edges. More specifically, naturally extending the classical notion of a transitive orientation in static graphs, we introduce the fundamental notion of a *temporal transitive orientation* and we systematically investigate its algorithmic behavior in various situations. An orientation of a temporal graph is called *temporally transitive* if, whenever u has a directed edge towards v with time-label t_1 and v has a directed edge towards w with time-label $t_2 \geq t_1$, then u also has a directed edge towards w with some time-label $t_3 \geq t_2$. If we just demand that this implication holds whenever $t_2 > t_1$, the orientation is called *strictly* temporally transitive, as it is based on the fact that there is a strict directed temporal path from u to w. Our main result is a conceptually simple, yet technically quite involved, polynomial-time algorithm for recognizing whether a given temporal graph $\mathcal{G}$ is transitively orientable. In wide contrast we prove that, surprisingly, it is NP-hard to recognize whether $\mathcal{G}$ is strictly transitively orientable. Additionally we introduce and investigate further related problems to temporal transitivity, notably among them the *temporal transitive completion* problem, for which we prove both algorithmic and hardness results.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Discrete mathematics

Keywords and phrases Temporal graph, transitive orientation, transitive closure, polynomial-time algorithm, NP-hardness, satisfiability.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.50

Related Version *A full version of the paper is available at [36]*: `https://arxiv.org/abs/2102.06783`

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.
Hendrik Molter: Supported by the German Research Foundation (DFG), project MATE (NI 369/17), and by the Israeli Science Foundation (ISF), grant No. 1070/20.
Malte Renken: Supported by the German Research Foundation (DFG), project MATE (NI 369/17).

Paul G. Spirakis: Supported by the NeST initiative of the School of EEE and CS at the University of Liverpool and by the EPSRC grant EP/P02002X/1.

1 Introduction

A *temporal* (or *dynamic*) network is, roughly speaking, a network whose underlying topology changes over time. This notion concerns a great variety of both modern and traditional networks; information and communication networks, social networks, and several physical systems are only few examples of networks which change over time [26, 38, 41]. Due to its vast applicability in many areas, the notion of temporal graphs has been studied from different perspectives under several different names such as *time-varying*, *evolving*, *dynamic*, and *graphs over time* (see [13–15] and the references therein). In this paper we adopt a simple and natural model for temporal networks which is given with discrete time-labels on the edges of a graph, while the vertex set remains unchanged. This formalism originates in the foundational work of Kempe et al. [27].

▶ **Definition 1** (Temporal Graph [27]). *A* temporal graph *is a pair* $\mathcal{G} = (G, \lambda)$*, where* $G = (V, E)$ *is an underlying (static) graph and* $\lambda : E \to \mathbb{N}$ *is a* time-labeling *function which assigns to every edge of* G *a discrete-time label.*

Mainly motivated by the fact that, due to causality, entities and information in temporal graphs can only "flow" along sequences of edges whose time-labels are non-decreasing (resp. increasing), Kempe et al. introduced the notion of a *(strict) temporal path*, or *(strict) time-respecting path*, in a temporal graph (G, λ) as a path in G with edges $e_1, e_2, \ldots, e_k$ such that $\lambda(e_1) \leq \ldots \leq \lambda(e_k)$ (resp. $\lambda(e_1) < \ldots < \lambda(e_k)$). This notion of a temporal path naturally resembles the notion of a *directed* path in the classical static graphs, where the direction is from smaller to larger time-labels along the path. Nevertheless, in temporal paths the individual time-labeled edges remain undirected: an edge $e = \{u, v\}$ with time-label $\lambda(e) = t$ can be abstractly interpreted as "u communicates with v at time t". Here the relation "communicates" is symmetric between u and v, i.e. it can be interpreted that the information can flow in either direction.

In this paper we make a first attempt to understand how the direction of information flow on one edge can impact the direction of information flow on other edges. More specifically, naturally extending the classical notion of a transitive orientation in static graphs [23], we introduce the fundamental notion of a *temporal transitive orientation* and we thoroughly investigate its algorithmic behavior in various situations. Imagine that v receives information from u at time t_1, while w receives information from v at time $t_2 \geq t_1$. Then w *indirectly* receives information from u through the intermediate vertex v. Now, if the temporal graph correctly records the transitive closure of information passing, the directed edge from u to w must exist and must have a time label $t_3 \geq t_2$. In such a *transitively oriented* temporal graph, whenever an edge is oriented from a vertex u to a vertex w with time-label t, we have that *every* temporal path from u to w arrives no later than t, and that there is no temporal path from w to u. Different notions of temporal transitivity have also been used for automated temporal data mining [40] in medical applications [39], text processing [45]. Furthermore, in behavioral ecology, researchers have used a notion of orderly (transitive) triads A-B-C to quantify dominance among species. In particular, animal groups usually form dominance hierarchies in which dominance relations are transitive and can also change with time [32].

One natural motivation for our temporal transitivity notion may come from applications where confirmation and verification of information is vital, where vertices may represent entities such as investigative journalists or police detectives who gather sensitive information. Suppose that v queried some important information from u (the information source) at time t_1, and afterwards, at time $t_2 \geq t_1$, w queried the important information from v (the intermediary). Then, in order to ensure the validity of the information received, w might want to verify it by *subsequently* querying the information directly from u at some time $t_3 \geq t_2$. Note that w might first receive the important information from u through various other intermediaries, and using several channels of different lengths. Then, to maximize confidence about the information, w should query u for verification only after receiving the information from the latest of these indirect channels.

It is worth noting here that the model of temporal graphs given in Definition 1 has been also used in its extended form, in which the temporal graph may contain multiple time-labels per edge [34]. This extended temporal graph model has been used to investigate temporal paths [3, 9, 11, 16, 34, 47] and other temporal path-related notions such as temporal analogues of distance and diameter [1], reachability [2] and exploration [1, 3, 20, 21], separation [22, 27, 48], and path-based centrality measures [12, 28], as well as recently non-path problems too such as temporal variations of coloring [37], vertex cover [4], matching [35], cluster editing [18], and maximal cliques [8, 25, 46]. However, in order to better investigate and illustrate the inherent combinatorial structure of temporal transitivity orientations, in this paper we mostly follow the original definition of temporal graphs given by Kempe et al. [27] with one time-label per edge [7, 17, 19]. Throughout the paper, whenever we assume multiple time-labels per edge we will state it explicitly; in all other cases we consider a single label per edge.

In static graphs, the transitive orientation problem has received extensive attention which resulted in numerous efficient algorithms. A graph is called *transitively orientable* (or a *comparability* graph) if it is possible to orient its edges such that, whenever we orient u towards v and v towards w, then the edge between u and w exists and is oriented towards w. The first polynomial-time algorithms for recognizing whether a given (static) graph G on n vertices and m edges is comparability (i.e. transitively orientable) were based on the notion of *forcing* an orientation and had running time $O(n^3)$ (see Golumbic [23] and the references therein). Faster algorithms for computing a transitive orientation of a given comparability graph have been later developed, having running times $O(n^2)$ [43] and $O(n + m \log n)$ [29], while the currently fastest algorithms run in linear $O(n+m)$ time and are based on efficiently computing a modular decomposition of G [30, 31]; see also Spinrad [44]. It is fascinating that, although all the latter algorithms compute a valid transitive orientation if G is a comparability graph, they fail to recognize whether the input graph is a comparability graph; instead they produce an orientation which is non-transitive if G is not a comparability graph. The fastest known algorithm for determining whether a given orientation is transitive requires matrix multiplication, currently achieved in $O(n^{2.37286})$ time [5].

Our contribution. In this paper we introduce the notion of *temporal transitive orientation* and we thoroughly investigate its algorithmic behavior in various situations. An orientation of a temporal graph $\mathcal{G} = (G, \lambda)$ is called *temporally transitive* if, whenever u has a directed edge towards v with time-label t_1 and v has a directed edge towards w with time-label $t_2 \geq t_1$, then u also has a directed edge towards w with some time-label $t_3 \geq t_2$. If we just demand that this implication holds whenever $t_2 > t_1$, the orientation is called *strictly* temporally transitive, as it is based on the fact that there is a strict directed temporal path from u to w. Similarly, if we demand that the transitive directed edge from u to w has time-label $t_3 > t_2$, the orientation is called *strongly* (resp. *strongly strictly)* temporally transitive.

Although these four natural variations of a temporally transitive orientation seem superficially similar to each other, it turns out that their computational complexity (and their underlying combinatorial structure) varies massively. Indeed we obtain a surprising result in Section 3: deciding whether a temporal graph $\mathcal{G}$ admits a *temporally transitive* orientation is solvable in polynomial time (Section 3.2), while it is NP-hard to decide whether it admits a *strictly temporally transitive* orientation (Section 3.1). On the other hand, it turns out that, deciding whether $\mathcal{G}$ admits a *strongly* or a *strongly strictly* temporal transitive orientation is (easily) solvable in polynomial time as they can both be reduced to 2SAT satisfiability.

Our main result is that, given a temporal graph $\mathcal{G} = (G, \lambda)$, we can decide in polynomial time whether $\mathcal{G}$ is transitively orientable, and at the same time we can output a temporal transitive orientation if it exists. Although the analysis and correctness proof of our algorithm is technically quite involved, our algorithm is simple and easy to implement, as it is based on the notion of *forcing* an orientation.[1] Our algorithm extends and generalizes the classical polynomial-time algorithm for computing a transitive orientation in static graphs described by Golumbic [23]. The main technical difficulty in extending the algorithm from the static to the temporal setting is that, in temporal graphs we cannot simply use orientation forcings to eliminate the condition that a *triangle* is not allowed to be cyclically oriented. To resolve this issue, we first express the recognition problem of temporally transitively orientable graphs as a Boolean satisfiability problem of a *mixed* Boolean formula $\phi_{\text{3NAE}} \wedge \phi_{\text{2SAT}}$. Here ϕ_{3NAE} is a 3NAE (i.e. 3-Not-All-Equal) formula and ϕ_{2SAT} is a 2SAT formula. Note that every clause $\texttt{NAE}(\ell_1, \ell_2, \ell_3)$ of ϕ_{3NAE} corresponds to the condition that a specific triangle in the temporal graph cannot be cyclically oriented. However, although deciding whether ϕ_{2SAT} is satisfiable can be done in linear time with respect to the size of the formula [6], the problem Not-All-Equal-3-SAT is NP-complete [42].

Our algorithm iteratively produces at iteration j a formula $\phi^{(j)}_{\text{3NAE}} \wedge \phi^{(j)}_{\text{2SAT}}$, which is computed from the previous formula $\phi^{(j-1)}_{\text{3NAE}} \wedge \phi^{(j-1)}_{\text{2SAT}}$ by (almost) simulating the classical greedy algorithm that solves 2SAT [6]. The 2SAT-algorithm proceeds greedily as follows. For every variable x_i, if setting $x_i = 1$ (resp. $x_i = 0$) leads to an immediate contradiction, the algorithm is forced to set $x_i = 0$ (resp. $x_i = 1$). Otherwise, if each of the truth assignments $x_i = 1$ and $x_i = 0$ does not lead to an immediate contradiction, the algorithm arbitrarily chooses to set $x_i = 1$ or $x_i = 0$, and thus some clauses are removed from the formula as they were satisfied. The argument for the correctness of the 2SAT-algorithm is that new clauses are *never added* to the formula at any step. The main technical difference between the 2SAT-algorithm and our algorithm is that, in our case, the formula $\phi^{(j)}_{\text{3NAE}} \wedge \phi^{(j)}_{\text{2SAT}}$ is *not* necessarily a sub-formula of $\phi^{(j-1)}_{\text{3NAE}} \wedge \phi^{(j-1)}_{\text{2SAT}}$, as in some cases we need to also add clauses. Our main technical result is that, nevertheless, at every iteration j the formula $\phi^{(j)}_{\text{3NAE}} \wedge \phi^{(j)}_{\text{2SAT}}$ is satisfiable if and only if $\phi^{(j-1)}_{\text{3NAE}} \wedge \phi^{(j-1)}_{\text{2SAT}}$ is satisfiable. The proof of this result (see Theorem 9) relies on a sequence of structural properties of temporal transitive orientations which we establish. This phenomenon of deducing a polynomial-time algorithm for an algorithmic graph problem by deciding satisfiability of a mixed Boolean formula (i.e. with both clauses of two and three literals) occurs rarely; this approach has been successfully used for the efficient recognition of simple-triangle (known also as "PI") graphs [33].

In the second part of our paper (Section 4) we consider a natural extension of the temporal orientability problem, namely the *temporal transitive completion* problem. In this problem we are given a (partially oriented) temporal graph $\mathcal{G}$ and a natural number k, and the question is whether it is possible to add at most k new edges (with the corresponding time-labels) to

[1] That is, orienting an edge from u to v *forces* us to orient another edge from a to b.

$\mathcal{G}$ such that the resulting temporal graph is (strongly/strictly/strongly strictly) transitively orientable. We prove that all four versions of temporal transitive completion are NP-complete, even when the input temporal graph is completely unoriented. In contrast we show that, if the input temporal graph $\mathcal{G}$ is *directed* (i.e. if every time-labeled edge has a fixed orientation) then all versions of temporal transitive completion are solvable in polynomial time. As a corollary of our results it follows that all four versions of temporal transitive completion are fixed-parameter-tractable (FPT) with respect to the number q of unoriented time-labeled edges in $\mathcal{G}$.

In the third and last part of our paper (Section 5) we consider the *multilayer transitive orientation* problem. In this problem we are given an undirected temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$, and we ask whether there exists an orientation F of its edges (i.e. with exactly one orientation for each edge of G) such that, for every 'time-layer" $t \geq 1$, the (static) oriented graph induced by the edges having time-label t is transitively oriented in F. Problem definitions of this type are commonly referred to as multilayer problems [10]. Observe that this problem trivially reduces to the static case if we assume that each edge has a single time-label, as then each layer can be treated independently of all others. However, if we allow $\mathcal{G}$ to have multiple time-labels on every edge of G, then we show that the problem becomes NP-complete, even when every edge has at most two labels.

Due to space constraints, some of our results are deferred to a full version [36].

2 Preliminaries and Notation

Given a (static) undirected graph $G = (V, E)$, an edge between two vertices $u, v \in V$ is denoted by the unordered pair $\{u, v\} \in E$, and in this case the vertices u, v are said to be *adjacent*. If the graph is directed, we will use the ordered pair (u, v) (resp. (v, u)) to denote the oriented edge from u to v (resp. from v to u). For simplicity of the notation, we will usually drop the parentheses and the comma when denoting an oriented edge, i.e. we will denote (u, v) just by uv. Furthermore, $\widehat{uv} = \{uv, vu\}$ is used to denote the set of both oriented edges uv and vu between the vertices u and v.

Let $S \subseteq E$ be a subset of the edges of an undirected (static) graph $G = (V, E)$, and let $\widehat{S} = \{uv, vu : \{u, v\} \in S\}$ be the set of both possible orientations uv and vu of every edge $\{u, v\} \in S$. Let $F \subseteq \widehat{S}$. If F contains *at least one* of the two possible orientations uv and vu of each edge $\{u, v\} \in S$, then F is called an *orientation* of the edges of S. F is called a *proper orientation* if it contains *exactly one* of the orientations uv and vu of every edge $\{u, v\} \in S$. Note here that, in order to simplify some technical proofs, the above definition of an orientation allows F to be not proper, i.e. to contain *both* uv and vu for a specific edge $\{u, v\}$. However, whenever F is not proper, this means that F can be discarded as it cannot be used as a part of a (temporal) transitive orientation. For every orientation F denote by $F^{-1} = \{vu : uv \in F\}$ the *reversal* of F. Note that $F \cap F^{-1} = \emptyset$ if and only if F is proper.

In a temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$, whenever $\lambda(\{v, w\}) = t$ (or simply $\lambda(v, w) = t$), we refer to the tuple $(\{v, w\}, t)$ as a *time-edge* of $\mathcal{G}$. A triangle of (G, λ) on the vertices u, v, w is a *synchronous triangle* if $\lambda(u, v) = \lambda(v, w) = \lambda(w, u)$. Let $G = (V, E)$ and let F be a proper orientation of the whole edge set E. Then $(\mathcal{G}, F)$, or (G, λ, F), is a *proper orientation* of the temporal graph $\mathcal{G}$. A *partial proper orientation* F of $\mathcal{G} = (G, \lambda)$ is an orientation of a subset of E. To indicate that the edge $\{u, v\}$ of a time-edge $(\{u, v\}, t)$ is oriented from u to v (that is, $uv \in F$ in a (partial) proper orientation F), we use the term $((u, v), t)$, or simply (uv, t). For simplicity we may refer to a (partial) proper orientation just as a (partial) orientation, whenever the term "proper" is clear from the context.

A static graph $G = (V, E)$ is a *comparability graph* if there exists a proper orientation F of E which is *transitive*, that is, if $F \cap F^{-1} = \emptyset$ and $F^2 \subseteq F$, where $F^2 = \{uw : uv, vw \in F$ for some vertex $v\}$ [23]. Analogously, in a temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$, we define a proper orientation F of E to be *temporally transitive*, if:

whenever (uv, t_1) and (vw, t_2) are oriented time-edges in $(\mathcal{G}, F)$ such that $t_2 \geq t_1$, there exists an oriented time-edge (wu, t_3) in $(\mathcal{G}, F)$, for some $t_3 \geq t_2$.

In the above definition of a temporally transitive orientation, if we replace the condition "$t_3 \geq t_2$" with "$t_3 > t_2$", then F is called *strongly temporally transitive*. If we instead replace the condition "$t_2 \geq t_1$" with "$t_2 > t_1$", then F is called *strictly temporally transitive*. If we do both of these replacements, then F is called *strongly strictly temporally transitive*. Note that strong (strict) temporal transitivity implies (strict) temporal transitivity, while (strong) temporal transitivity implies (strong) strict temporal transitivity. Furthermore, similarly to the established terminology for static graphs, we define a temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$, to be a *(strongly/strictly) temporal comparability graph* if there exists a proper orientation F of E which is *(strongly/strictly) temporally transitive*.

We are now ready to formally introduce the following decision problem of recognizing whether a given temporal graph is temporally transitively orientable or not.

Temporal Transitive Orientation (TTO)

Input: A temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$.
Question: Does $\mathcal{G}$ admit a temporally transitive orientation F of E?

In the above problem definition of TTO, if we ask for the existence of a strictly (resp. strongly, or strongly strictly) temporally transitive orientation F, we obtain the decision problem Strict (resp. Strong, or Strong Strict) Temporal Transitive Orientation (TTO).

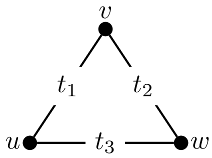
Let $\mathcal{G} = (G, \lambda)$ be a temporal graph, where $G = (V, E)$. Let $G' = (V, E')$ be a graph such that $E \subseteq E'$, and let $\lambda' \colon E' \to \mathbb{N}$ be a time-labeling function such that $\lambda'(u, v) = \lambda(u, v)$ for every $\{u, v\} \in E$. Then the temporal graph $\mathcal{G}' = (G', \lambda')$ is called a *temporal supergraph of* $\mathcal{G}$. We can now define our next problem definition regarding computing temporally orientable supergraphs of $\mathcal{G}$.

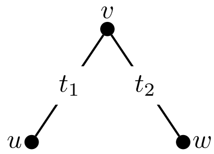
Temporal Transitive Completion (TTC)

Input: A temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$, a (partial) orientation F of $\mathcal{G}$, and an integer k.
Question: Does there exist a temporal supergraph $\mathcal{G}' = (G', \lambda')$ of (G, λ), where $G' = (V, E')$, and a transitive orientation $F' \supseteq F$ of $\mathcal{G}'$ such that $|E' \setminus E| \leq k$?

Similarly to TTO, if we ask in the problem definition of TTC for the existence of a strictly (resp. strongly, or strongly strictly) temporally transitive orientation F', we obtain the decision problem Strict (resp. Strong, or Strong Strict) Temporal Transitive Completion (TTC).

Now we define our final problem which asks for an orientation F of a temporal graph $\mathcal{G} = (G, \lambda)$ (i.e. with exactly one orientation for each edge of G) such that, for every "time-layer" $t \geq 1$, the (static) oriented graph defined by the edges having time-label t is transitively oriented in F. This problem does not make much sense if every edge has exactly one time-label in $\mathcal{G}$, as in this case it can be easily solved by just repeatedly applying any known static transitive orientation algorithm. Therefore, in the next problem definition, we assume that in the input temporal graph $\mathcal{G} = (G, \lambda)$ every edge of G potentially has multiple time-labels, i.e. the time-labeling function is $\lambda : E \to 2^{\mathbb{N}}$.





	$t_1 = t_2 = t_3$	$t_1 < t_2 = t_3$	$t_1 \leq t_2 < t_3$	$t_1 = t_2$	$t_1 < t_2$
TTO	non-cyclic	$wu = wv$	$vw \implies uw$ $vu \implies wu$	$uv = wv$	$uv \implies wv$
Strong TTO	$\perp$	$wu \wedge wv$	$vw \implies uw$ $vu \implies wu$	$uv = wv$	$uv \implies wv$
Strict TTO	$\top$	non-cyclic	$vw \implies uw$ $vu \implies wu$	$\top$	$uv \implies wv$
Str. Str. TTO	$\top$	$vu \implies wu$ $uv \implies wv$	$vw \implies uw$ $vu \implies wu$	$\top$	$uv \implies wv$

Table 1 Orientation conditions imposed by a triangle (left) and an induced path of length two (right) in the underlying graph G for the decision problems (Strict/Strong/Strong Strict) TTO. Here, $\top$ means that no restriction is imposed, $\perp$ means that the graph is not orientable, and in the case of triangles, "non-cyclic" means that all orientations except the ones that orient the triangle cyclicly are allowed.

Multilayer Transitive Orientation (MTO)

Input: A temporal graph $\mathcal{G} = (G, \lambda)$, where $G = (V, E)$ and $\lambda : E \to 2^{\mathbb{N}}$.
Question: Is there an orientation F of the edges of G such that, for every $t \geq 1$, the (static) oriented graph induced by the edges having time-label t is transitively oriented?

3 The recognition of temporally transitively orientable graphs

In this section we investigate the computational complexity of all variants of TTO. We show that TTO as well as the two variants Strong TTO and Strong Strict TTO, are solvable in polynomial time, whereas Strict TTO turns out to be NP-complete.

The main idea of our approach to solve TTO and its variants is to create Boolean variables for each edge of the underlying graph G and interpret setting a variable to 1 or 0 with the two possible ways of directing the corresponding edge.

More formally, for every edge $\{u, v\}$ we introduce a variable x_{uv} and setting this variable to 1 corresponds to the orientation uv while setting this variable to 0 corresponds to the orientation vu. Now consider the example of Figure 1(a), i.e. an induced path of length two in the underlying graph G on three vertices u, v, w, and let $\lambda(u, v) = 1$ and $\lambda(v, w) = 2$. Then the orientation uv "forces" the orientation wv. Indeed, if we otherwise orient $\{v, w\}$ as vw, then the edge $\{u, w\}$ must exist and be oriented as uw in any temporal transitive orientation, which is a contradiction as there is no edge between u and w. We can express this "forcing" with the implication $x_{uv} \implies x_{wv}$. In this way we can deduce the constraints that all triangles or induced paths on three vertices impose on any (strong/strict/strong strict) temporal transitive orientation. We collect all these constraints in Table 1.

When looking at the conditions imposed on temporal transitive orientations collected in Table 1, we can observe that all conditions except "non-cyclic" are expressible in 2SAT. Since 2SAT is solvable in linear time [6], it immediately follows that the strong variants of temporal transitivity are solvable in polynomial time, as the next theorem states.

▶ **Theorem 2.** Strong TTO *and* Strong Strict TTO *are solvable in polynomial time.*

In the variants TTO and Strict TTO, however, we can have triangles which impose a "non-cyclic" orientation of three edges (Table 1). This can be naturally modeled by a not-all-equal (NAE) clause.[2] However, if we now naïvely model the conditions with a Boolean formula, we obtain a formula with 2SAT clauses and 3NAE clauses. Deciding whether such a formula is satisfiable is NP-complete in general [42]. Hence, we have to investigate these two variants more thoroughly.

The only difference between the triangles that impose these "non-cyclic" orientations in these two problem variants is that, in TTO, the triangle is *synchronous* (i.e. all its three edges have the same time-label), while in Strict TTO two of the edges are synchronous and the third one has a smaller time-label than the other two. As it turns out, this difference of the two problem variants has important implications on their computational complexity. In fact, we obtain a surprising result: TTO is solvable in polynomial time while Strict TTO is NP-complete.

3.1 Strict TTO is NP-Complete

In this section we show that in contrast to the other variants, Strict TTO is NP-complete.

▶ **Theorem 3.** Strict TTO *is NP-complete even if the temporal input graph has only four different time labels.*

3.2 A polynomial-time algorithm for TTO

Let $G = (V, E)$ be a static undirected graph. There are various polynomial-time algorithms for deciding whether G admits a transitive orientation F. However our results in this section are inspired by the transitive orientation algorithm described by Golumbic [23], which is based on the crucial notion of *forcing* an orientation. The notion of forcing in static graphs is illustrated in Figure 1 (a): if we orient the edge $\{u, v\}$ as uv (i.e., from u to v) then we are forced to orient the edge $\{v, w\}$ as wv (i.e., from w to v) in any transitive orientation F of G. Indeed, if we otherwise orient $\{v, w\}$ as vw (i.e. from v to w), then the edge $\{u, w\}$ must exist and it must be oriented as uw in any transitive orientation F of G, which is a contradiction as $\{u, w\}$ is not an edge of G. Similarly, if we orient the edge $\{u, v\}$ as vu then we are forced to orient the edge $\{v, w\}$ as vw. That is, in any transitive orientation F of G we have that $uv \in F \Leftrightarrow wv \in F$. This forcing operation can be captured by the binary forcing relation Γ which is defined on the edges of a static graph G as follows [23].

$$uv\ \Gamma\ u'v' \quad \text{if and only if} \quad \begin{cases} \text{either } u = u' \text{ and } \{v, v'\} \notin E \\ \text{or } v = v' \text{ and } \{u, u'\} \notin E \end{cases}. \tag{1}$$

We now extend the definition of Γ in a natural way to the binary relation Λ on the edges of a temporal graph (G, λ), see Equation (2). For this, observe from Table 1 that the only cases, where we have $uv \in F \Leftrightarrow wv \in F$ in any temporal transitive orientation of (G, λ), are

[2] A not all equal clause is a set of literals and it evaluates to `true` if and only if at least two literals in the set evaluate to different truth values.

Figure 1 The orientation uv forces the orientation wu and vice-versa in the examples of (a) a static graph G where $\{u,v\},\{v,w\} \in E(G)$ and $\{u,w\} \notin E(G)$, and of (b) a temporal graph (G,λ) where $\lambda(u,w) = 3 < 5 = \lambda(u,v) = \lambda(v,w)$.

when (i) the vertices u,v,w induce a path of length 2 (see Figure 1 (a)) and $\lambda(u,v) = \lambda(v,w)$, as well as when (ii) u,v,w induce a triangle and $\lambda(u,w) < \lambda(u,v) = \lambda(v,w)$. The latter situation is illustrated in the example of Figure 1 (b). The binary forcing relation Λ is only defined on pairs of edges $\{u,v\}$ and $\{u',v'\}$ where $\lambda(u,v) = \lambda(u',v')$, as follows.

$$uv \ \Lambda \ u'v' \ \text{ if and only if } \ \lambda(u,v) = \lambda(u',v') = t \text{ and } \begin{cases} u = u' \text{ and } \{v,v'\} \notin E, \text{or} \\ v = v' \text{ and } \{u,u'\} \notin E, \text{or} \\ u = u' \text{ and } \lambda(v,v') < t, \text{or} \\ v = v' \text{ and } \lambda(u,u') < t. \end{cases} \tag{2}$$

Note that, for every edge $\{u,v\} \in E$ we have that $uv \ \Lambda \ uv$. The forcing relation Λ for temporal graphs shares some properties with the forcing relation Γ for static graphs. In particular, the reflexive transitive closure Λ^* of Λ is an equivalence relation, which partitions the edges of each set $E_t = \{\{u,v\} \in E : \lambda(u,v) = t\}$ into its Λ-*implication classes* (or simply, into its *implication classes*). Two edges $\{a,b\}$ and $\{c,d\}$ are in the same Λ-implication class if and only $ab \ \Lambda^* \ cd$, i.e. there exists a sequence $ab = a_0b_0 \ \Lambda \ a_1b_1 \ \Lambda \ \ldots \ \Lambda \ a_kb_k = cd$, with $k \geq 0$. Note that, for this to happen, we must have $\lambda(a_0,b_0) = \lambda(a_1,b_1) = \ldots = \lambda(a_k,b_k) = t$ for some $t \geq 1$. Such a sequence is called a Λ-chain from ab to cd, and we say that ab (eventually) Λ-forces cd. Furthermore note that $ab \ \Lambda^* \ cd$ if and only if $ba \ \Lambda^* \ dc$. For the next lemma, we use the notation $\widehat{A} = \{uv, vu : uv \in A\}$.

▶ **Lemma 4.** *Let A be a Λ-implication class of a temporal graph (G,λ). Then either $A = A^{-1} = \widehat{A}$ or $A \cap A^{-1} = \emptyset$.*

▶ **Definition 5.** *Let F be a proper orientation and A be a Λ-implication class of a temporal graph (G,λ). If $A \subseteq F$, we say that F* respects *A.*

▶ **Lemma 6.** *Let F be a proper orientation and A be a Λ-implication class of a temporal graph (G,λ). Then F respects either A or A^{-1} (i.e. either $A \subseteq F$ or $A^{-1} \subseteq F$), and in either case $A \cap A^{-1} = \emptyset$.*

The next lemma, which is crucial for proving the correctness of our algorithm, extends an important known property of the forcing relation Γ for static graphs [23, Lemma 5.3] to the temporal case.

▶ **Lemma 7** (Temporal Triangle Lemma)**.** *Let (G,λ) be a temporal graph and with a synchronous triangle on the vertices a,b,c, where $\lambda(a,b) = \lambda(b,c) = \lambda(c,a) = t$. Let A,B,C be three Λ-implication classes of (G,λ), where $ab \in C$, $bc \in A$, and $ca \in B$, where $A \neq B^{-1}$ and $A \neq C^{-1}$.*

1. *If some $b'c' \in A$, then $ab' \in C$ and $c'a \in B$.*
2. *If some $b'c' \in A$ and $a'b' \in C$, then $c'a' \in B$.*
3. *No edge of A touches vertex a.*

Deciding temporal transitivity using Boolean satisfiability. Starting with any undirected edge $\{u, v\}$ of the underlying graph G, we can clearly enumerate in polynomial time the whole Λ-implication class A to which the oriented edge uv belongs (cf. Equation (2)). If the reversely directed edge $vu \in A$ then Lemma 4 implies that $A = A^{-1} = \widehat{A}$. Otherwise, if $vu \notin A$ then $vu \in A^{-1}$ and Lemma 4 implies that $A \cap A^{-1} = \emptyset$. Thus, we can also decide in polynomial time whether $A \cap A^{-1} = \emptyset$. If we encounter a Λ-implication class A such that $A \cap A^{-1} \neq \emptyset$, then it follows by Lemma 6 that (G, λ) is not temporally transitively orientable.

In the remainder of the section we will assume that $A \cap A^{-1} = \emptyset$ for every Λ-implication class A of (G, λ), which is a *necessary* condition for (G, λ) to be temporally transitive orientable. Moreover it follows by Lemma 6 that, if (G, λ) admits a temporally transitively orientation F, then either $A \subseteq F$ or $A^{-1} \subseteq F$. This allows us to define a Boolean variable x_A for every Λ-implication class A, where $x_A = \overline{x_{A^{-1}}}$. Here $x_A = 1$ (resp. $x_{A^{-1}} = 1$) means that $A \subseteq F$ (resp. $A^{-1} \subseteq F$), where F is the temporally transitive orientation which we are looking for. Let $\{A_1, A_2, \ldots, A_s\}$ be a set of Λ-implication classes such that $\{\widehat{A_1}, \widehat{A_2}, \ldots, \widehat{A_s}\}$ is a partition of the edges of the underlying graph G.[3] Then any truth assignment τ of the variables $x_1, x_2, \ldots, x_s$ (where $x_i = x_{A_i}$ for every $i = 1, 2, \ldots, s$) corresponds bijectively to one possible orientation of the temporal graph (G, λ), in which every Λ-implication class is oriented consistently.

Now we define two Boolean formulas ϕ_{3NAE} and ϕ_{2SAT} such that (G, λ) admits a temporal transitive orientation if and only if there is a truth assignment τ of the variables $x_1, x_2, \ldots, x_s$ such that both ϕ_{3NAE} and ϕ_{2SAT} are simultaneously satisfied. Intuitively, ϕ_{3NAE} captures the "non-cyclic" condition from Table 1 while ϕ_{2SAT} captures the remaining conditions. Here ϕ_{3NAE} is a 3NAE formula, i.e., the disjunction of clauses with three literals each, where every clause $\texttt{NAE}(\ell_1, \ell_2, \ell_3)$ is satisfied if and only if at least one of the literals $\{\ell_1, \ell_2, \ell_3\}$ is equal to 1 and at least one of them is equal to 0. Furthermore ϕ_{2SAT} is a 2SAT formula, i.e., the disjunction of 2CNF clauses with two literals each, where every clause $(\ell_1 \vee \ell_2)$ is satisfied if and only if at least one of the literals $\{\ell_1, \ell_2\}$ is equal to 1.

For simplicity of the presentation we also define a variable x_{uv} for every directed edge uv. More specifically, if $uv \in A_i$ (resp. $uv \in A_i^{-1}$) then we set $x_{uv} = x_i$ (resp. $x_{uv} = \overline{x_i}$). That is, $x_{uv} = \overline{x_{vu}}$ for every undirected edge $\{u, v\} \in E$. Note that, although $\{x_{uv}, x_{vu} : \{u, v\} \in E\}$ are defined as variables, they can equivalently be seen as *literals* in a Boolean formula over the variables $x_1, x_2, \ldots, x_s$. The process of building all Λ-implication classes and all variables $\{x_{uv}, x_{vu} : \{u, v\} \in E\}$ is given by Algorithm 1.

Description of the 3NAE formula ϕ_{3NAE}. The formula ϕ_{3NAE} captures the "non-cyclic" condition of the problem variant TTO (presented in Table 1). The formal description of ϕ_{3NAE} is as follows. Consider a synchronous triangle of (G, λ) on the vertices u, v, w. Assume that $x_{uv} = x_{wv}$, i.e., x_{uv} is the same variable as x_{wv}. Then the pair $\{uv, wv\}$ of oriented edges belongs to the same Λ-implication class A_i. This implies that the triangle on the vertices u, v, w is never cyclically oriented in any proper orientation F that respects A_i or A_i^{-1}. Note that, by symmetry, the same happens if $x_{vw} = x_{uw}$ or if $x_{wu} = x_{vu}$. Assume, on the contrary, that $x_{uv} \neq x_{wv}$, $x_{vw} \neq x_{uw}$, and $x_{wu} \neq x_{vu}$. In this case we add to ϕ_{3NAE} the clause $\texttt{NAE}(x_{uv}, x_{vw}, x_{wu})$. Note that the triangle on u, v, w is transitively oriented if and only if $\texttt{NAE}(x_{uv}, x_{vw}, x_{wu})$ is satisfied, i.e., at least one of the variables $\{x_{uv}, x_{vw}, x_{wu}\}$ receives the value 1 and at least one of them receives the value 0.

[3] Here we slightly abuse the notation by identifying the undirected edge $\{u, v\}$ with the set of both its orientations $\{uv, vu\}$.

Algorithm 1 Building the Λ-implication classes and the edge-variables.

Input: A temporal graph (G, λ), where $G = (V, E)$.
Output: The variables $\{x_{uv}, x_{vu} : \{u, v\} \in E\}$, or the announcement that (G, λ) is temporally not transitively orientable.

$s \leftarrow 0$; $E_0 \leftarrow E$
while $E_0 \neq \emptyset$ **do**
  $s \leftarrow s + 1$; Let $\{p, q\} \in E_0$ be arbitrary
  Build the Λ-implication class A_s of the oriented edge pq (by Equation (2))
  if $qp \in A_s$ **then** $\{A_s \cap A_s^{-1} \neq \emptyset\}$
    return "NO"
  else
    x_s is the variable corresponding to the directed edges of A_s
    for every $uv \in A_s$ **do**
      $x_{uv} \leftarrow x_s$; $x_{vu} \leftarrow \overline{x_s}$ $\{x_{uv}$ and x_{vu} become aliases of x_s and $\overline{x_s}\}$
    $E_0 \leftarrow E_0 \setminus \widehat{A_s}$
return Λ-implication classes $\{A_1, A_2, \ldots, A_s\}$ and variables $\{x_{uv}, x_{vu} : \{u, v\} \in E\}$

Description of the 2SAT formula ϕ_{2SAT}. The formula ϕ_{2SAT} captures all conditions apart from the "non-cyclic" condition of the problem variant TTO (presented in Table 1). The formal description of ϕ_{2SAT} is as follows. Consider a triangle of (G, λ) on the vertices u, v, w, where $\lambda(u, v) = t_1$, $\lambda(v, w) = t_2$, $\lambda(w, v) = t_3$, and $t_1 \leq t_2 \leq t_3$. If $t_1 < t_2 = t_3$ then we add to ϕ_{2SAT} the clauses $(x_{uw} \vee x_{wv}) \wedge (x_{vw} \vee x_{wu})$; note that these clauses are equivalent to $x_{wu} = x_{wv}$. If $t_1 \leq t_2 < t_3$ then we add to ϕ_{2SAT} the clauses $(x_{wv} \vee x_{uw}) \wedge (x_{uv} \vee x_{wu})$; note that these clauses are equivalent to $(x_{vw} \Rightarrow x_{uw}) \wedge (x_{vu} \Rightarrow x_{wu})$. Now consider a path of length 2 that is induced by the vertices u, v, w, where $\lambda(u, v) = t_1$, $\lambda(v, w) = t_2$, and $t_1 \leq t_2$. If $t_1 = t_2$ then we add to ϕ_{2SAT} the clauses $(x_{vu} \vee x_{wv}) \wedge (x_{vw} \vee x_{uv})$; note that these clauses are equivalent to $(x_{uv} = x_{wv})$. Finally, if $t_1 < t_2$ then we add to ϕ_{2SAT} the clause $(x_{vu} \vee x_{wv})$; note that this clause is equivalent to $(x_{uv} \Rightarrow x_{wv})$.

Brief outline of the algorithm. In the *initialization phase*, we exhaustively check which truth values are *forced* in $\phi_{\text{3NAE}} \wedge \phi_{\text{2SAT}}$ by using the subroutine Initial-Forcing. During the execution of Initial-Forcing, we either replace the formulas ϕ_{3NAE} and ϕ_{2SAT} by the equivalent formulas $\phi_{\text{3NAE}}^{(0)}$ and $\phi_{\text{2SAT}}^{(0)}$, respectively, or we reach a contradiction by showing that $\phi_{\text{3NAE}} \wedge \phi_{\text{2SAT}}$ is unsatisfiable.

▶ **Observation 8.** *The temporal graph (G, λ) is transitively orientable if and only if $\phi_{3NAE}^{(0)} \wedge \phi_{2SAT}^{(0)}$ is satisfiable.*

The *main phase* of the algorithm starts once the formulas $\phi_{\text{3NAE}}^{(0)}$ and $\phi_{\text{2SAT}}^{(0)}$ have been computed. Then we iteratively try assigning to each variable x_i the truth value 1 or 0. Once we have set $x_i = 1$ (resp. $x_i = 0$) during the iteration $j \geq 1$ of the algorithm, we call algorithm Boolean-Forcing (see Algorithm 3) as a subroutine to check which implications this value of x_i has on the current formulas $\phi_{\text{3NAE}}^{(j-1)}$ and $\phi_{\text{2SAT}}^{(j-1)}$ and which other truth values of variables are forced. The correctness of Boolean-Forcing can be easily verified by checking all subcases of Boolean-Forcing. During the execution of Boolean-Forcing, we either replace the current formulas by $\phi_{\text{3NAE}}^{(j)}$ and $\phi_{\text{2SAT}}^{(j)}$, or we reach a contradiction by showing that, setting $x_i = 1$ (resp. $x_i = 0$) makes $\phi_{\text{3NAE}}^{(j-1)} \wedge \phi_{\text{2SAT}}^{(j-1)}$ unsatisfiable. If each of the truth assignments $\{x_i = 1, x_i = 0\}$ leads to such a contradiction, we return that (G, λ)

Algorithm 2 Initial-Forcing

Input: A 2-SAT formula ϕ_{2SAT} and a 3-NAE formula ϕ_{3NAE}

Output: A 2-SAT formula $\phi^{(0)}_{\text{2SAT}}$ and a 3-NAE formula $\phi^{(0)}_{\text{3NAE}}$ such that $\phi^{(0)}_{\text{2SAT}} \wedge \phi^{(0)}_{\text{3NAE}}$ is satisfiable if and only if $\phi_{\text{2SAT}} \wedge \phi_{\text{3NAE}}$ is satisfiable, or the announcement that $\phi_{\text{2SAT}} \wedge \phi_{\text{3NAE}}$ is not satisfiable.

1: $\phi^{(0)}_{\text{3NAE}} \leftarrow \phi_{\text{3NAE}}$; $\phi^{(0)}_{\text{2SAT}} \leftarrow \phi_{\text{2SAT}}$ {initialization}
2: **for** every variable x_i appearing in $\phi^{(0)}_{\text{3NAE}} \wedge \phi^{(0)}_{\text{2SAT}}$ **do**
3:   **if** Boolean-Forcing $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}, x_i, 1\right) =$ "NO" **then**
4:     **if** Boolean-Forcing $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}, x_i, 0\right) =$ "NO" **then**
5:       **return** "NO" {both $x_i = 1$ and $x_i = 0$ invalidate the formulas}
6:     **else**
7:       $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}\right) \leftarrow$ Boolean-Forcing $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}, x_i, 0\right)$
8:   **else**
9:     **if** Boolean-Forcing $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}, x_i, 0\right) =$ "NO" **then**
10:       $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}\right) \leftarrow$ Boolean-Forcing $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}, x_i, 1\right)$
11: **for** every clause $\texttt{NAE}(x_{uv}, x_{vw}, x_{wu})$ of $\phi^{(0)}_{\text{3NAE}}$ **do**
12:   **for** every variable x_{ab} **do**
13:     **if** $x_{ab} \overset{*}{\Rightarrow}_{\phi^{(0)}_{\text{2SAT}}} x_{uv}$ and $x_{ab} \overset{*}{\Rightarrow}_{\phi^{(0)}_{\text{2SAT}}} x_{vw}$ **then** {add $(x_{ab} \Rightarrow x_{uw})$ to $\phi^{(0)}_{\text{2SAT}}$}
14:       $\phi^{(0)}_{\text{2SAT}} \leftarrow \phi^{(0)}_{\text{2SAT}} \wedge (x_{ba} \vee x_{uw})$
15: Repeat lines 2 and 11 until no changes occur on $\phi^{(0)}_{\text{2SAT}}$ and $\phi^{(0)}_{\text{3NAE}}$
16: **return** $\left(\phi^{(0)}_{\text{3NAE}}, \phi^{(0)}_{\text{2SAT}}\right)$

is a *no*-instance. Otherwise, if at least one of the truth assignments $\{x_i = 1, x_i = 0\}$ does not lead to such a contradiction, we follow this truth assignment and proceed with the next variable.

As we prove in our *main technical result* of this section (Theorem 9), $\phi^{(j-1)}_{\text{3NAE}} \wedge \phi^{(j-1)}_{\text{2SAT}}$ is satisfiable if and only if $\phi^{(j)}_{\text{3NAE}} \wedge \phi^{(j)}_{\text{2SAT}}$ is satisfiable. Note that, during the execution of the algorithm, we can *both add and remove* clauses from $\phi^{(j)}_{\text{2SAT}}$. On the other hand, we can *only remove* clauses from $\phi^{(j)}_{\text{3NAE}}$. Thus, at some iteration j, we obtain $\phi^{(j)}_{\text{3NAE}} = \emptyset$, and after that iteration we only need to decide satisfiability of $\phi^{(j)}_{\text{2SAT}}$ which can be done efficiently [6].

We are now ready to present in the next theorem our main technical result of this section.

▶ **Theorem 9.** *For every iteration $j \geq 1$ of the algorithm, $\phi^{(j)}_{3NAE} \wedge \phi^{(j)}_{2SAT}$ is satisfiable if and only if $\phi^{(j-1)}_{3NAE} \wedge \phi^{(j-1)}_{2SAT}$ is satisfiable.*

Using Theorem 9, we can now conclude this section with the next theorem.

▶ **Theorem 10.** TTO *can be solved in polynomial time.*

Algorithm 3 BOOLEAN-FORCING

Input: A 2-SAT formula ϕ_2, a 3-NAE formula ϕ_3, and a variable x_i of $\phi_2 \wedge \phi_3$, and a truth value VALUE $\in \{0,1\}$

Output: A 2-SAT formula ϕ_2' and a 3-NAE formula ϕ_3', obtained from ϕ_2 and ϕ_3 by setting $x_i =$ VALUE, or the announcement that $x_i =$ VALUE does not satisfy $\phi_2 \wedge \phi_3$.

```
1: φ'_2 ← φ_2;   φ'_3 ← φ_3
2: while φ'_2 has a clause (x_uv ∨ x_pq) and x_uv = 1 do
3:    Remove the clause (x_uv ∨ x_pq) from φ'_2
4: while φ'_2 has a clause (x_uv ∨ x_pq) and x_uv = 0 do
5:    if x_pq = 0 then return "NO"
6:    Remove the clause (x_uv ∨ x_pq) from φ'_2;   x_pq ← 1
7: for every variable x_uv that does not yet have a truth value do
8:    if x_uv ⇒*_{φ''_2} x_vu, where φ''_2 = φ'_2 \ φ_2 then x_uv ← 0
9: for every clause NAE(x_uv, x_vw, x_wu) of φ'_3 do {synchronous triangle on vertices u, v, w}
10:   if x_uv ⇒*_{φ'_2} x_vw then {add (x_uv ⇒ x_uw) ∧ (x_uw ⇒ x_vw) to φ'_2}
11:      φ'_2 ← φ'_2 ∧ (x_vu ∨ x_uw) ∧ (x_wu ∨ x_vw)
12:      Remove the clause NAE(x_uv, x_vw, x_wu) from φ'_3
13:   if x_uv already got the value 1 or 0 then
14:      Remove the clause NAE(x_uv, x_vw, x_wu) from φ'_3
15:      if x_vw and x_wu do not have yet a truth value then
16:         if x_uv = 1 then {add (x_vw ⇒ x_uw) to φ'_2}
17:            φ'_2 ← φ'_2 ∧ (x_wv ∨ x_uw)
18:         else {x_uv = 0;  in this case add (x_uw ⇒ x_vw) to φ'_2}
19:            φ'_2 ← φ'_2 ∧ (x_wu ∨ x_vw)
20:      if x_vw = x_uv and x_wu does not have yet a truth value then
21:         x_wu ← 1 − x_uv
22:      if x_vw = x_wu = x_uv then return "NO"
23: Repeat lines 2, 4, 7, and 9 until no changes occur on φ'_2 and φ'_3
24: if both x_uv = 0 and x_uv = 1 for some variable x_uv then return "NO"
25: return (φ'_2, φ'_3)
```

Proof sketch. First recall by Observation 8 that the input temporal graph (G,λ) is transitively orientable if and only if $\phi_{3\text{NAE}}^{(0)} \wedge \phi_{2\text{SAT}}^{(0)}$ is satisfiable.

Let (G,λ) be a *yes*-instance. Then, by iteratively applying Theorem 9 it follows that $\phi_{3\text{NAE}}^{(j)} \wedge \phi_{2\text{SAT}}^{(j)}$ is satisfiable, for every iteration j of the algorithm. Recall that, at the end of the last iteration k of the algorithm, $\phi_{3\text{NAE}}^{(k)} \wedge \phi_{2\text{SAT}}^{(k)}$ is empty. Then the algorithm gives the arbitrary truth value $x_i = 1$ to every variable x_i which did not yet get any truth value yet. This is a correct decision as all these variables are not involved in any Boolean constraint of $\phi_{3\text{NAE}}^{(k)} \wedge \phi_{2\text{SAT}}^{(k)}$ (which is empty). Finally, the algorithm orients all edges of G according to the corresponding truth assignment. The returned orientation F of (G,λ) is temporally transitive as every variable was assigned a truth value according to the Boolean constraints

Algorithm 4 Temporal transitive orientation.

Input: A temporal graph (G, λ), where $G = (V, E)$.
Output: A temporal transitive orientation F of (G, λ), or the announcement that (G, λ) is temporally not transitively orientable.

Execute Algorithm 1 to build the Λ-implication classes $\{A_1, A_2, \ldots, A_s\}$ and the Boolean variables $\{x_{uv}, x_{vu} : \{u, v\} \in E\}$
if Algorithm 1 returns "NO" **then return** "NO"
Build the 3NAE formula ϕ_{3NAE} and the 2SAT formula ϕ_{2SAT}
if Initial-Forcing $(\phi_{3NAE}, \phi_{2SAT}) \neq$ "NO" **then** {Initialization phase}
  $\left(\phi_{3NAE}^{(0)}, \phi_{2SAT}^{(0)}\right) \leftarrow$ Initial-Forcing $(\phi_{3NAE}, \phi_{2SAT})$
else {$\phi_{3NAE} \wedge \phi_{2SAT}$ leads to a contradiction}
  return "NO"
$j \leftarrow 1$; $F \leftarrow \emptyset$ {Main phase}
while a variable x_i appearing in $\phi_{3NAE}^{(j-1)} \wedge \phi_{2SAT}^{(j-1)}$ did not yet receive a truth value **do**
  if Boolean-Forcing $\left(\phi_{3NAE}^{(j-1)}, \phi_{2SAT}^{(j-1)}, x_i, 1\right) \neq$ "NO" **then**
    $\left(\phi_{3NAE}^{(j)}, \phi_{2SAT}^{(j)}\right) \leftarrow$ Boolean-Forcing $\left(\phi_{3NAE}^{(j-1)}, \phi_{2SAT}^{(j-1)}, x_i, 1\right)$
  else {$x_i = 1$ leads to a contradiction}
    if Boolean-Forcing $\left(\phi_{3NAE}^{(j-1)}, \phi_{2SAT}^{(j-1)}, x_i, 0\right) \neq$ "NO" **then**
      $\left(\phi_{3NAE}^{(j)}, \phi_{2SAT}^{(j)}\right) \leftarrow$ Boolean-Forcing $\left(\phi_{3NAE}^{(j-1)}, \phi_{2SAT}^{(j-1)}, x_i, 0\right)$
    else
      return "NO"
  $j \leftarrow j + 1$
for $i = 1$ to s **do**
  if x_i did not yet receive a truth value **then** $x_i \leftarrow 1$
  if $x_i = 1$ **then** $F \leftarrow F \cup A_i$ **else** $F \leftarrow F \cup \overline{A_i}$
return the temporally transitive orientation F of (G, λ)

throughout the execution of the algorithm.

Now let (G, λ) be a *no*-instance. We will prove that, at some iteration $j \leq 0$, the algorithm will "NO". Suppose otherwise that the algorithm instead returns an orientation F of (G, λ) after performing k iterations. Then clearly $\phi_{3NAE}^{(k)} \wedge \phi_{2SAT}^{(k)}$ is empty, and thus clearly satisfiable. Therefore, iteratively applying Theorem 9 implies that $\phi_{3NAE}^{(0)} \wedge \phi_{2SAT}^{(0)}$ is also satisfiable, and thus (G, λ) is temporally transitively orientable by Observation 8, which is a contradiction to the assumption that (G, λ) be a *no*-instance.

Lastly, we prove that our algorithm runs in polynomial time. The Λ-implication classes of (G, λ) can be clearly computed in polynomial time. Our algorithm calls a subroutine Boolean-Forcing at most four times for every variable in $\phi_{3NAE}^{(0)} \wedge \phi_{2SAT}^{(0)}$. Boolean-Forcing iteratively adds and removes clauses from the 2SAT part of the formula, while it can only remove clauses from the 3NAE part. Whenever a clause is added to the 2SAT part, a clause of the 3NAE part is removed. Therefore, as the initial 3NAE formula has at most polynomially-many clauses, we can add clauses to the 2SAT part only polynomially-many times. Hence, we have an overall polynomial running time. ◀

4 Temporal Transitive Completion

We now study the computational complexity of Temporal Transitive Completion (TTC). In the static case, the so-called *minimum comparability completion* problem, i.e. adding the smallest number of edges to a static graph to turn it into a comparability graph, is known to be NP-hard [24]. Note that minimum comparability completion on static graphs is a special case of TTC and thus it follows that TTC is NP-hard too. Our other variants, however, do not generalize static comparability completion in such a straightforward way. Note that for Strict TTC we have that the corresponding recognition problem Strict TTO is NP-complete (Theorem 3), hence it follows directly that Strict TTC is NP-hard. For the remaining two variants of our problem, we show in the following that they are also NP-hard, giving the result that all four variants of TTC are NP-hard. Furthermore, we present a polynomial-time algorithm for all four problem variants for the case that all edges of underlying graph are oriented, see Theorem 12. This allows directly to derive an FPT algorithm for the number of unoriented edges as a parameter.

▶ **Theorem 11.** *All four variants of* TTC *are NP-hard, even when the input temporal graph is completely unoriented.*

We now show that TTC can be solved in polynomial time, if all edges are already oriented, as the next theorem states.

▶ **Theorem 12.** *An instance* $(\mathcal{G}, F, k)$ *of* TTC *where* $\mathcal{G} = (G, \lambda)$ *and* $G = (V, E)$*, can be solved in* $O(m^2)$ *time if* F *is an orientation of* E*, where* $m = |E|$*.*

Using Theorem 12 we can now prove that TTC is fixed-parameter tractable (FPT) with respect to the number of unoriented edges in the input temporal graph $\mathcal{G}$.

▶ **Corollary 13.** *Let* $I = (\mathcal{G} = (G, \lambda), F, k)$ *be an instance of* TTC*, where* $G = (V, E)$*. Then* I *can be solved in* $O(2^q \cdot m^2)$*, where* $q = |E| - |F|$ *and* m *the number of time edges.*

5 Deciding Multilayer Transitive Orientation

In this section we prove that Multilayer Transitive Orientation (MTO) is NP-complete, even if every edge of the given temporal graph has at most two labels. Recall that this problem asks for an orientation F of a temporal graph $\mathcal{G} = (G, \lambda)$ (i.e. with exactly one orientation for each edge of G) such that, for every "time-layer" $t \geq 1$, the (static) oriented graph defined by the edges having time-label t is transitively oriented in F. As we discussed in Section 2, this problem makes more sense when every edge of G potentially has multiple time-labels, therefore we assume here that the time-labeling function is $\lambda : E \to 2^{\mathbb{N}}$.

▶ **Theorem 14.** MTO *is NP-complete, even on temporal graphs with at most two labels per edge.*

References

1 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.

2 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017.

3 Eleni C. Akrida, George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, Paul G. Spirakis, and Viktor Zamaraev. How fast can we reach a target vertex in stochastic temporal graphs? *Journal of Computer and System Sciences*, 114:65–83, 2020. An extended abstract appeared at ICALP 2019.

4 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020.

5 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539, 2021.

6 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

7 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 149:1–149:14, 2016.

8 Matthias Bentert, Anne-Sophie Himmel, Hendrik Molter, Marco Morik, Rolf Niedermeier, and René Saitenmacher. Listing all maximal k-plexes in temporal graphs. *ACM Journal of Experimental Algorithmics*, 24(1):13:1–13:27, 2019.

9 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020.

10 Robert Bredereck, Christian Komusiewicz, Stefan Kratsch, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Assessing the computational complexity of multilayer subgraph detection. *Network Science*, 7(2):215–241, 2019.

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

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

13 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865762`.

14 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865764`.

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

16 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. In *31st International Symposium on Algorithms and Computation (ISAAC)*, pages 30:1–30:18, 2020.

17 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 134:1–134:14, 2019.

18 Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondřej Suchý. Cluster editing in multi-layer and temporal graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2018.

19 J. Enright, K. Meeks, G.B. Mertzios, and V. Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.

20 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021.

21 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 444–455, 2015.

22 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020.

23 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2nd edition, 2004.

24 S Louis Hakimi, Edward F Schmeichel, and Neal E Young. Orienting graphs to optimize reachability. *Information Processing Letters*, 63(5):229–235, 1997.

25 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.

26 Petter Holme and Jari Saramäki. *Temporal network theory*, volume 2. Springer, 2019.

27 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

28 Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. *Physical Review E*, 85(2):026107, 2012.

29 Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 536–545, 1994.

30 Ross M. McConnell and Jeremy P. Spinrad. Linear-time transitive orientation. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 19–25, 1997.

31 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

32 David B McDonald and Daizaburo Shizuka. Comparative transitive and temporal orderliness in dominance networks. *Behavioral Ecology*, 24(2):511–520, 2013.

33 George B. Mertzios. The recognition of simple-triangle graphs and of linear-interval orders is polynomial. *SIAM Journal on Discrete Mathematics*, 29(3):1150–1185, 2015.

34 George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 657–668, 2013.

35 George B Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154, pages 27:1–27:14, 2020.

36 George B. Mertzios, Hendrik Molter, Malte Renken, Paul G. Spirakis, and Philipp Zschoche. The complexity of transitively orienting temporal graphs. *arXiv preprint arXiv:2102.06783*, 2021.

37 George B Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 7667–7674, 2019.

38 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, January 2018.

39 Robert Moskovitch and Yuval Shahar. Medical temporal-knowledge discovery via temporal abstraction. In *Proceedings of the AMIA Annual Symposium*, page 452, 2009.

40 Robert Moskovitch and Yuval Shahar. Fast time intervals mining using the transitivity of temporal relations. *Knowledge and Information Systems*, 42(1):21–48, 2015.

41 V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. Graph metrics for temporal networks. In *Temporal Networks*. Springer, 2013.

42 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

43 Jeremy P. Spinrad. On comparability and permutation graphs. *SIAM Journal on Computing*, 14(3):658–670, 1985.

44 Jeremy P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute Monographs*. American Mathematical Society, 2003.

45 Xavier Tannier and Philippe Muller. Evaluating temporal graphs built from texts via transitive reduction. *Journal of Artificial Intelligence Research (JAIR)*, 40:375–413, 2011.

46 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

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

48 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.