Network Science **9** (4): 443–477, 2022
doi:10.1017/nws.2021.20

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Toward a generalized notion of discrete time for modeling temporal networks

Konstantin Kueffner[1,2] and Mark Strembeck[1,2,3]*

[1]Vienna University of Economics and Business, WU, Vienna, Austria; [2]Secure Business Austria Research Center (SBA), Vienna, Austria; and [3]Complexity Science Hub Vienna (CSH), Vienna, Austria
*Corresponding author. Email: mark.strembeck@wu.ac.at

Action Editor: Ulrik Brandes

Abstract

Many real-world networks, including social networks and computer networks for example, are temporal networks. This means that the vertices and edges change over time. However, most approaches for modeling and analyzing temporal networks do not explicitly discuss the underlying notion of time. In this paper, we therefore introduce a generalized notion of discrete time for modeling temporal networks. Our approach also allows for considering nondeterministic time and incomplete data, two issues that are often found when analyzing datasets extracted from online social networks, for example. In order to demonstrate the consequences of our generalized notion of time, we also discuss the implications for the computation of (shortest) temporal paths in temporal networks. In addition, we implemented an R-package that provides programming support for all concepts discussed in this paper. The R-package is publicly available for download.

Keywords: multilayer network; multivalued path; temporal network; temporal path; time; weighted network

1. Introduction

Temporal networks are a means for modeling and analyzing the temporal dimension of complex (networked) systems. However, most of the literature on temporal networks either does not explicitly discuss the underlying notion of time or uses a rather restrictive conception of time, for example, linear time. In this paper, we discuss a more general form of temporal networks, designed for modeling nonlinear flows of time. Thereby, our approach allows for considering nondeterministic and incomplete data when analyzing temporal networks—two issues that often appear when dealing with real-world data, such as data extracted from online social networks. This paper is a corrected and significantly extended version of Kueffner & Strembeck (2019).

We use multilayer networks to construct temporal networks alongside some generalized flow of time. In particular, we consider a temporal network, as a (temporal) sequence of networks, a notion that can easily be expressed via the well-known multilayer network concept. This approach allows to clearly separate time from other (temporally varying) attributes that are attached to the edges or vertices in a network. This strict separation, however, also requires to consider multivalued path lengths, the implications of which will be discussed as well.

Furthermore, some areas of application for our formalization of discrete time will be discussed in Section 4. Those areas range from its possible connection to modal logic, its application to planning problems, as well as its ability to encode inconsistent data.

In order to provide programming support for the concepts introduced in this paper, we also implemented a corresponding R-package.[1] Our R-package provides an extension to the `igraph`-package (Csardi & Nepusz, 2006) and thereby allows for the modeling and analysis of discrete temporal networks (for details see Appendix B).

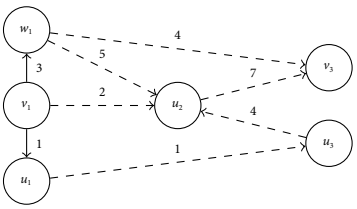
1.1 Basic definitions

For the purposes this paper, we use the notion of multilayer networks as specified in Definition 1.1, which understands multilayer networks as a family of graphs with their vertices being connected by inter-layer edges.

For the sake of readability, we mostly provide informal (or semi-formal), but intuitive definitions to introduce the different concepts. However, the corresponding formal definitions can be found in Appendix A.

Definition 1.1. *A weighted multilayer network is a triple* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \omega \rangle$ *such that for some arbitrary set of labels* I, $\mathcal{G} := (G_\alpha)_{\alpha \in I}$ *is a family of weighted graphs,* $\mathcal{R} := (R_{\alpha\beta})_{\alpha,\beta \in I}$ *is a family of relations, and* $\omega : E(\mathcal{M}) \to \mathbb{R}$ *assigns weights to the edges of the network.*

Example 1.1. *Consider the weighted multi-layer network* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \omega \rangle$.



The figure above corresponds to the following formal description. We have $\mathcal{G}$, *which is a family containing the weighted graphs* $G_1 := \langle \{u_1, v_1, w_1\}, \{(v_1, u_1), (v_1, w_1)\}, \omega_1 \rangle$, $G_2 := \langle \{u_2\}, \varnothing, \omega_2 \rangle$, *and* $G_3 := \langle \{u_3, v_3\}, \varnothing, \omega_3 \rangle$ *where* $\{(v_1, u_1) \mapsto 1, (v_1, w_1) \mapsto 3\}$. *We have* $\mathcal{R}$ *which is a family of relations such that only* $R_{1,2} := \{(v_1, u_2), (w_1, u_2)\}$, $R_{1,3} := \{(u_1, u_3), (w_1, v_3)\}$, $R_{2,3} := \{(u_2, v_3)\}$, *and* $R_{3,2} := \{(u_3, u_2)\}$ *are non-empty And we have* ω *which is defined by combining* $R_{1,2} := \{(v_1, u_2) \mapsto 2, (w_1, u_2) \mapsto 5\}$, $R_{1,3} := \{(u_1, u_3) \mapsto 1, (w_1, v_3) \mapsto 4\}$, $R_{2,3} := \{(u_2, v_3) \mapsto 7\}$, *and* $R_{3,2} := \{(u_3, u_2) \mapsto 4\}$, *as well as* ω_1, ω_2 *and* ω_3.

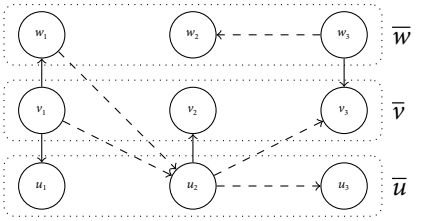
This is not the only possible notion of multilayer networks though. For example, one other formalism defines tuples for representing the layer membership of different vertices, while yet another represents a multilayer network solely as a tensor to be understood as an analog to the adjacency matrix of ordinary graphs. For further details, see, for example, De Domenico et al. (2013), Boccaletti et al. (2014), and Kivelä et al. (2014).

A multiplex is commonly understood as a multilayer network where all layers share the same set of vertices, that is, $\forall U, W \in V(\mathcal{G})\ U = W$ (Boccaletti et al., 2014; Kivelä et al., 2014). For the purposes of this paper, we relax the strong equality implied by those characterizations, as apart from some technical issues with respect to path properties, strong equality seems to be inappropriate for our purposes. For example, an object (vertex or edge) x at time t may not share the same properties as the same object (vertex or edge) x at time $t' \neq t$.

Definition 1.2. *A multilayer network with equivalence* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \equiv \rangle$ *is one where the vertices are partitioned into equivalence classes such that each equivalence class contains at most one vertex per graph. Such a multilayer network with equivalence is called a multiplex iff each equivalence class contains exactly one vertex per graph.*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Example 1.2. *Below we can see a multiplex with the equivalence classes are $\overline{u}$, $\overline{v}$, and $\overline{w}$ (the edge weights are suppressed for clarity):*



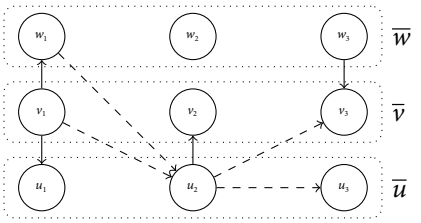
Note that the multiplex requirement enforces that each graph has the same number of vertices.

Moreover, the edges between network layers will be called *inter-layer edges*, while those within a single layer will be called *intra-layer edges*. Similarly, those edges between equivalence classes will be called *inter-class edges* and those within will be called *intra-class edges*.

Unfortunately, when modeling the progression of time, equivalence is necessary but not sufficient. For achieving the required expressiveness, we have to impose an additional order onto our structure. That is, if a graph G_α precedes a graph G_β according to some order $\preceq$, then the vertex v_α should precede v_β.

Definition 1.3. *A multilayer network with equivalence and order $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq \rangle$ is a multilayer network with equivalence such that the relations permit an ordering of the graphs in $\mathcal{G}$.*

Example 1.3. *Consider the multiplex*



Here, we can order the graphs such that $G_1 \preceq G_2 \preceq G_3$ without conflicting with the intra-graph edges, for example, the multiplex in Example 1.2 does not permit such an ordering due to the edge (w_3, w_2). Note that this ordering can also be used to construct an ordering over the vertices, for example, under such an ordering v_1 a predecessor of v_2.

Having defined the fundamental language of multilayer networks used in this paper, Section 2 now discusses the semantics and structure of time. Subsequently, Section 3 will use this notion of time for the definition of discrete temporal networks.

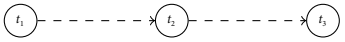
2. Generalized flow of time

We abstract from the notion of a temporal graph to discuss some more general observations regarding objects in a temporal dimension. When studying the notion of time, we distinguish between two classes of properties. Informally, the properties concerned with the *density* of time and the ones determining the *structure* of time. Many concepts mentioned in this section can be found in and/or build on the ideas presented by Venema (2017), Markosian et al. (2016), and Goranko & Galton (2015) and are heavily based on Kripke semantics (Van Ditmarsch et al., 2007).

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Definition 2.1. *Let $\Phi := \langle T, R, \iota \rangle$ be a structure representing the flow of time, where T is a set of points, R is a relation over T, and $\iota : T \to \wp(\mathcal{L})$ is a function assigning each point in time a set of properties P described in some language $\mathcal{L}$.*

Example 2.1. *The following graph depicts a flow of time where the properties are specified using propositional logic:*



with $\iota(t_1) := \{\neg a, \neg b\}$, $\iota(t_2) := \{a, \neg b\}$, and $\iota(t_3) := \{\neg a, b\}$, where

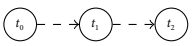
- *a means "Alice is home" and*
- *b means "Bob is home."*

There are some interesting conceptions of how points in time may relate, for example, cyclic or bidirectional notions of time. Both of which would allow, in one form or another, for the possibility of time travel. However, while theoretically possible (Gödel, 1949) and sometimes assumed within network theory, for example, in Taylor et al. (2017), such peculiarities shall not be considered in this paper. Hence, rather than allowing for an arbitrary relation R, we restrict ourself for all subsequent discussions to the set of structures building upon a directed notion of time.

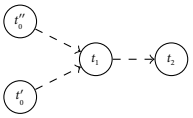
Definition 2.2. *Let $\Phi := \langle T, \preceq \rangle$ be a structure representing the flow of time, where T is a set of points and $\preceq$ is a (strict) partial order over T, see Venema (2017).*

By imposing further restriction onto $\preceq$, we can develop certain notions of time (see also Venema, 2017), for example:

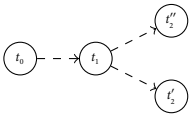
Example 2.2. *A flow of time is* linear *if its underlying order is total, for example,*



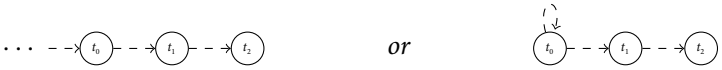
A flow of time is backward branching *if for a point t_1 representing the present there are two incomparable points in the past, for example,*



A flow of time is forward branching *if for a point t_1 representing the present there are two incomparable points in the future, for example,*

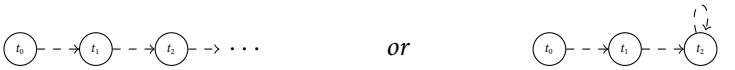


A flow of time is backward-serial *if there is always another point in the past, for example,*



https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

A flow of time is forward-serial *if there is always another point in the future, for example,*



The concept of *linear time* is highly intuitive: time flows within a straight line, there are no alternative time lines, no branching and no cycles, allowing us to work in a deterministic fashion. For example, one encounters this notion of time when dealing with normal time series. Moreover, a good part of the literature regarding temporal networks is concerned with linear time. However, one can easily find scenarios, where we deal with some kind of uncertainty or nondeterminism. Here, the notion of *possible worlds* can guide our reasoning. Kripke models, which provide the foundation for the semantics of modal logic, heavily rely on the concept of possible worlds (Venema, 2017; Van Ditmarsch et al., 2007).

In our case, branches can represent possible futures, thereby expressing the nondeterminism of the future. For example, having a linear flow of time in the past that branches into the future, expresses that we are certain what happened in the past, but we cannot predict the future with certainty, that is, there are multiple conceivable scenarios to account for. How those variants of the future are obtained precisely, may it be through statistical inference, by consulting experts with domain knowledge, or being a product of a simulation with randomness is currently not of our concern.

Considering possible worlds is especially useful when dealing with discrete objects such as graphs. That is, rather than introducing fuzzy edges (Sunitha & Mathew, 2013), we can, at least for our purposes, consider alternative worlds where an edge exists and some in which it does not. Similarly, backward branching could introduce the notion of unreliable data into our models. For example, if there two contradicting measurements of the same phenomenon at the same instance, one models those measurements as two incomparable elements within the flow of time (e.g., see Sections 3 and 4). Moreover, regardless of forward or backward branching, we do allow for collapsing flows of time. That is, two branches could meet at some point in time. Recall the unreliable data example mentioned above as an example for the applicability of such a structure which will be expanded upon in Example 4.5.

2.1 Paths in time

When discussing the density of time, three general notions are common: a flow of time can be *discrete*, *dense*, or *continuous*, thus existing in analogy to $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. However, since most measurements of real-world phenomena are processed by inherently discrete machines and this paper focuses on the analysis of discrete sequences of discrete objects, we will limit ourself to a discrete conception of time. Nevertheless, for analytical and predictive purposes, the other two models of time, especially the continuous one, should be investigated further in the context of temporal graphs (Venema, 2017).

In this paper, the restriction to discrete notions of time allows for a concrete conception of successors (i.e., we can formalize steps in time). Let $succ(t) \subseteq T$ therefore be the set of direct successors of the point in time $t \in T$. Similarly, let $pred(t)$ be the set of its direct predecessors. Clearly, if we have a linear flow of time, both sets contain at most one element. Using both the successor and the predecessor function, we can define the notion of a path in time. Namely, a *forward path* is simply a sequence of points in time such the next element in the sequence must be a successor of the previous one, that is, $\ldots, t, t', \ldots$ such that $t' \in succ(t)$. A *backward path in time* is defined in analog w.r.t. the predecessor function.

In general, if we refer to a *path in time*, it is either a forward or a backward path in time, and this prevents reachability between incomparable elements. We write $x \rightsquigarrow^+ y$ to indicate that there exists a forward path in time from the point x to the point y. Similarly, we write $x \rightsquigarrow^- y$ for backward path and $x \rightsquigarrow y$ for general paths.

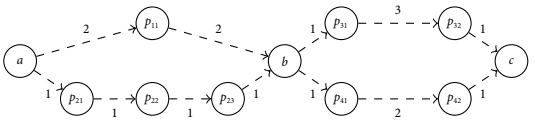
https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

The size of a path in time is determined by the elements in the path and can be understood as the amount of steps in time between two points. However, for our purposes, we require a more refined notion of distance. To accomplish this, we need to expand the flow in time.

Definition 2.3. *Let $\Phi := \langle T, \preceq, \omega \rangle$ be a flow of time with spacing, with the function ω being a function that assigns weights to the successor relations of the flow of time.*

Intuitively, ω stretches time, that is, it indicates the duration between two neighboring points in time. Using ω, we can define the *length* by summing up its weights. However, while the length of a path in time is a useful concept, it *cannot* serve as a measure of temporal distance.

Example 2.3. *Consider the flow of time $\Phi := \langle T, \preceq, \omega \rangle$:*



What is the distance δ between a and b and between b and c? If constructed based on the notion of a path, we can observe the following. For $|\cdot|$, we have $\delta(a,b) = 2$ or $\delta(a,b) = 4$ and for $|\cdot|_\omega$ we have $\delta_\omega(a,b) = 4$ or $\delta_\omega(a,b) = 4$. Analogously, we have $\delta(b,c) = 3$ or $\delta(b,c) = 3$ and $\delta_\omega(b,c) = 5$ or $\delta_\omega(b,c) = 4$.[2]

To resolve this ambiguity, we make use of a quasi-metric with infinity (Schroeder, 2006). A fairly natural choice for modeling noncyclic, directed flows of time. As one can easily travel into the future one instant at a time, while traveling into the past is, at least for the common man, impossible. Moreover, by assigning an infinite distance to backtracking within our flow of time, the distance between two branches also becomes infinite. One can easily check that this holds for paths with and without a join. More information about quasi-metrics can be found in Matthews (1994), Schroeder (2006).

For the remainder of this paper, we use the following quasi-metric.

Definition 2.4. *Let $\Phi := \langle T, \preceq, \omega \rangle$ be a flow of time with spacing. We call*

$$\delta_\omega(a,b) := \begin{cases} \min\limits_{a \rightsquigarrow b}(|a \rightsquigarrow b|_\omega) & a \preceq b \\ \infty & otw. \end{cases}$$

the distance between a and b (with respect to ω). Moreover, we call δ_1, where ω is the constant function 1, the step distance between a and b.

However, while this may allow us to speak about distance, in one form or another. It creates some semantic inconveniences or inconsistencies (see Example 2.4). Many of which can be reduced to the fact that, in their general form, flows in time can easily be used to express time as having varying density, that is, the spacing between points may vary within a single path or across branches. While sometimes useful, for example, consider relativity of time, a characterization of time without such properties is desirable as well.

Example 2.4. *Consider the flow of time as in Example 2.3. We can find two paths from a to b and two paths from b to c. For the latter, our notion of step distance causes no issues, as all paths between b and c have the same size, that is, $|\cdot|$. However, in the case a to b, two paths of different size can be found. Hence, if one wants to consider all points at a certain step distance from b, one obtains*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

$\delta_1(a,b) < \delta_1(a,p_{23})$ *but* $p_{23} < b$. *That is, the successor is closer than its predecessor. This can be resolved by adjusting the density of the flow of time across branches, for example,. by manipulating the spacing between two points.*

A flow of time, in which this issue cannot arise is a flow where every path between a join and a meet has the same length. Leading us to the definition of global homogeneousness.

Definition 2.5. *Let* $\Phi := \langle T, \preceq, \omega \rangle$ *be a quasi-metric flow of time. Then* Φ *is a globally homogeneous flow of time iff*

$$\delta_\omega(x,y) + \delta_\omega(y,z) = \infty \vee \delta_\omega(x,z) = \delta_\omega(x,y) + \delta_\omega(y,z)$$

If a flow of time satisfies this property, we can be sure that every path between two points has the same length according to δ_ω. However, it should be mentioned that the interpretation above is not the only possible notion of homogeneity (see, e.g., Venema, 2017).

3. Temporal networks

The notion of a temporal network, intuitively to be understood as a graph with an additional structure encoding the dimension of time, has been captured by multiple formalisms and is known under various names across different fields (Holme & Saramäki, 2012; Casteigts et al., 2012). The general approach of multilayer networks was introduced to unify formalisms that extend the ordinary notion of a graph, and this includes several formalisms concerned with capturing the notion of a temporal network.

In Holme & Saramäki (2012), an important distinction is made between *instant-based temporal graphs* (also called contact sequences) and *interval-based temporal graphs*. The prior understands time as a sequence of instants, where, for example, an edge is labeled with a sequence of time stamps indicating the instants at which this edge is present. This can be modeled with multilayer networks by introducing a new graph/layer for each time stamp that contains the edges active at that time and by connecting the corresponding vertices with inter-layer edges to simulate the progression of time. The latter, that is, interval-based temporal graphs, allow for the modeling of interactions with a duration. That is, rather than labeling edges with sets of time stamps, one labels edges with sets of intervals. Each of those intervals specifies a fragment of time during which the respective edge exists. Thereby, allowing the modeling of interactions with durations.

However, this paper is primarily concerned with discrete time; thus, one can conceive each interval as a countable sequence of instants. Meaning that while operating under a discrete notion of time, it suffices to discuss instant-based contact sequences only. To that extend, building on Definition 1.3, consider the following definition.

Definition 3.1. *Let* $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$ *be a weighted multilayer network with equivalence, we call it an instant-based temporal network iff*

- $\mathcal{G} := \{G_t \mid t \in T\}$ *for some labeling* T.
- $\mathcal{R}$ *is a collection of all successor relations with respect to* $\preceq$, *that is,* $\mathcal{R} := \{R_{t_i t_j} \mid G_{t_i}, G_{t_j} \in \mathcal{G} \wedge R_{t_i t_j} := \{(v,w) \mid v \in G_{t_i} \wedge w \in G_{t_j}\ Succ(G_{t_i}, G_{t_j}) \wedge v \equiv w\}\}$.
- ω *is defined such that it respects the intra-layer relation weights, that is, if* $e \in E(G_{t_i})$ *then* $\omega(e) := \omega_{t_i}(e)$, *while assigning positive weights to every inter-layer relation, that is, if* $e \in E(\mathcal{R})$ *then* $\omega(e) := k$ *for some* $k > 0$.

Moreover, we observe that a temporal graph can be understood as a flow of time with additional structure. That is, consider the flow in time $\mathcal{T} := \langle \mathcal{G}, \preceq, \iota, \omega \rangle$. Now for all $G_t \in \mathcal{G}$, we fix $\iota(G_t) = G_t$

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

where G_t is some weighted graph $G_t := \langle V_t, E_t, \omega_t \rangle$. By fixing the world at a certain point in time to being the same as its label, ι becomes redundant. Let $\equiv$ be in Definition 1.2 and let $\preceq$ extend to the vertices such that $\forall G_{t_i}, G_{t_j} \in \mathcal{G} \forall v \in G_{t_i} \forall w \in G_{t_j} \left(G_{t_i} \preceq G_{t_j} \implies (\overline{v} = \overline{w} \iff v \preceq w) \right)$. Then, $\mathcal{R}$ is just the set of successors with respect to our extended flow relation, where ω carries over.

3.1 Paths in temporal networks

The notion of a path in a static, non-weighted graph is a fairly simple concept, with its length being defined as its size, that is, its number of edges. In this case, a shortest path can be computed efficiently and has an fairly unambiguous semantics (Tang et al., 2009; Wu et al., 2014). Unfortunately, this property is already lost when considering weighted graphs in general. For example, allowing for negative weights can make the computation of the shortest path more expensive even without the presence of negative cycles (Cormen et al., 2009, pp. 646–658). If such cycles exist, the notion of shortest path already breaks down, that is, in a strongly connected graph the weight of every shortest path would be $-\infty$.

Moreover, even if one restricts the problem to simple graphs only, one is already faced with an NP-complete problem, a fact that can be shown by means of reduction from and to the longest simple path problem (Cormen et al., 2009, p. 1048). Additionally, as weights allow for the distinction between similarity and dissimilarity measures, the semantics of those weights has to be accounted for as well (Runkler, 2012; Segarra & Ribeiro, 2016). This is due to the fact that some measures only provide sensible results when they are similarity measures, for example, degree centrality, while others require dissimilarity measures, for example, closeness centrality (see, e.g., Goshtasby, 2012). Fortunately, it is possible to invert the semantic interpretation of the respective measure. One example of this would be $d_s(x, y) = \frac{1}{1+d_d(x,y)}$, with d_s and d_d being some measure of similarity and dissimilarity, respectively. Now consider a multilayer network, each layer having (possibly) different semantics. The same holds true for temporal networks, where we have a dissimilarity measure on inter-layer edges and another measure with different semantic on intra-layer edges. Hence, making the notion of a shortest path even more difficult (Runkler, 2012; Segarra & Ribeiro, 2016; Goshtasby, 2012).

Within the context of temporal networks, Wu et al. (2014) introduces a set of minimum temporal paths, consisting of

- the *earliest arrival path*, that is, starting from a_0 find the path ending in the smallest $b \in \overline{b}$;
- the *latest departure path*, that is, what is the largest $a \in \overline{a}$, while still being able to reach $\overline{b}$;
- the *fastest path*, that is, what is the shortest path between $\overline{a}$ and $\overline{b}$ minimizing the difference between ending time and starting time;
- the *shortest path*, that is, the path that is the shortest with respect to traversal time.

In Tang et al. (2009), a temporal graph is conceptualized as a sequence of graphs. However, by limiting the amount of hops within each static graph, they manage to encode some notion of time into each static graph. Moreover, it is not uncommon to make the distinction between the size of a path and its duration explicit (Holme & Saramäki, 2012; Casteigts et al., 2012; Michail, 2016). Analogously to flows in time, this distinction roughly corresponds to $|\cdot|_1$ and $|\cdot|_\omega$ (see Definitions 2.3 and 2.4). As already mentioned, when dealing with weighted temporal graphs, an additional dimension is introduced. Namely, we distinguish not only between temporal steps and temporal distance, but also between intra-layer steps and intra-layer distance. One approach would be to use some norm to collapse those two dimensions. However, motivated by the definitions found in Wu et al. (2014) and the issue of similarity and dissimilarity, we instead use the

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

concept of a path in time together with its notion of distance, as well the notion of a path in a regular graph.

Definition 3.2. *Let $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$ be a instant-based temporal network and let $v \in G_{t_i}$ and $w \in G_{t_j}$. Then the alternating sequence of regular paths and forward paths in time $\mathcal{T} \supseteq (v \rightsquigarrow_{\mathcal{T}} w) = (\overline{v}_{t_i} \rightsquigarrow_{\mathcal{T}} \overline{w}_{t_j}) := \overline{v}_{t_i} \rightsquigarrow^{+}_{\Phi} \overline{v}_{t_k} \rightsquigarrow \overline{w}_{t_k} \cdots \rightsquigarrow^{+}_{\Phi} \overline{u}_{t_j} \rightsquigarrow \overline{w}_{t_j}$ is called the temporal path from $\overline{v}_{t_i}$ to $\overline{w}_{t_j}$. We write $\lambda_{\Phi}(x \rightsquigarrow_{\mathcal{T}} y)$ to address all temporal edges in $x \rightsquigarrow_{\mathcal{T}} y$ and $\lambda_{\mathcal{G}}(x \rightsquigarrow_{\mathcal{T}} y)$ to address all in-graph edges in $x \rightsquigarrow_{\mathcal{T}} y$.*[3]

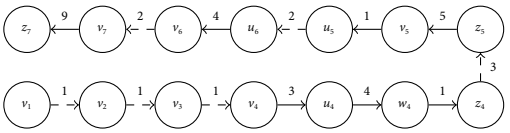
By distinguishing between the types of paths in temporal graphs, we allow for a separation of measures, that is, inter- and intra-length and size.

Definition 3.3. *Let $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$ be an instant-based temporal network, let $p := v \rightsquigarrow_{\mathcal{T}} w$ be the temporal path from v to w with $v, w \in V(\mathcal{T})$. Then, we define*

$$|p|_{\omega} := \left(\sum_{e \in \lambda_{\Phi}(p)} \omega(e), \sum_{e \in \lambda_{\mathcal{G}}(p)} \omega(e) \right)$$

as the length of p with respect to ω and $|p|_1$ as its size.

Example 3.1. *Consider the following path p, where dashed edges represent the temporal edges (i.e., inter-layer edges) and where solid edges represent normal edges (i.e., intra-layer edges):*



Given this one obtains

$$\lambda_{\Phi}(p) := \{v_1, v_2, v_3, v_4, z_4, z_5, u_5, u_6, v_6, v_7\}$$
$$\lambda_{\mathcal{G}}(p) := \{v_4, u_4, w_4, z_4, z_5, v_5, u_5, u_6, v_6, v_7, z_7\}$$

as well as $|p|_{\omega} := (10, 27)$.

By allowing for two dimensions with respect to path length and size, we have to define a specific order on those values. A natural choice for this is the so-called product order, that is, $(x_i, y_i) \leq (x_j, y_j) \iff x_i \leq x_j \wedge y_i \leq y_j$ (Dickson, 1913). However, one issue that immediately arises when using a product ordering is the issue of partiality (see Example 3.2). Moreover, there can be various natural interpretations of shortest path problems. For example, the problem $\overline{a} \rightsquigarrow_{\mathcal{T}} \overline{b}$ addresses the desire to compute the set of overall shortest distances between two equivalence classes, while the problem $\overline{a}_{t_i} \rightsquigarrow_{\mathcal{T}} \overline{b}$ restricts the same question to a certain starting point and $\overline{a} \rightsquigarrow_{\mathcal{T}} \overline{b}_{t_j}$ to a specific arrival date. Lastly, $\overline{a}_{t_i} \rightsquigarrow_{\mathcal{T}} \overline{b}_{t_j}$ is de facto an ordinary shortest path problem.[4]

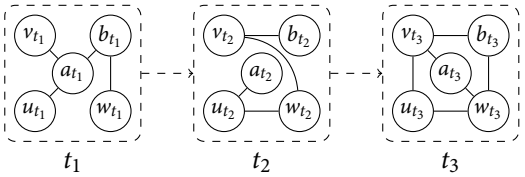
Example 3.2. *The temporal graph $\mathcal{T}$:*

where G_{t_1},G_{t_2}, and G_{t_3} are respectively

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Table 1. Compute all possible shortest paths between $\overline{a}$ and $\overline{b}$ and all elements within those equivalence classes

$\lvert \cdot \rightsquigarrow \cdot \rvert$	$\overline{b}_{t_1}$	$\overline{b}_{t_2}$	$\overline{b}_{t_3}$	$\overline{b}$
$\overline{a}_{t_1}$	$(0,1)$	$(1,1)$	$(2,1)$	$(0,1)$
$\overline{a}_{t_2}$	(∞,∞)	$(0,4)$	$(1,2)$	$(0,4),(1,2)$
$\overline{a}_{t_3}$	(∞,∞)	(∞,∞)	$(0,2)$	$(0,2)$
$\overline{a}$	$(0,1)$	$(1,1),(0,4)$	$(2,1),(0,2)$	$(0,1)$

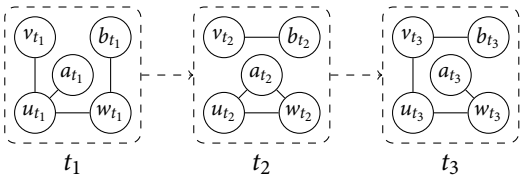


In this example, one can observe that each shortest path problem may lead to different and even multiple results.

Another significant observation is that from the shortest path between $\overline{x}$ and $\overline{y}$, $\overline{x} \rightsquigarrow \overline{z} \rightsquigarrow \overline{y}$, it is not possible to conclude that $\overline{x} \rightsquigarrow \overline{z}$ is the shortest path between $\overline{x}$ and $\overline{z}$ (see also Example 3.3). Thereby, prohibiting the safe use of Dijkstra's algorithm for some of the specified shortest temporal path problems.

Example 3.3. *The temporal graph $\mathcal{T}$:*

where G_{t_1},G_{t_2}, and G_{t_3} are respectively



Consider, $\overline{a} \rightsquigarrow \overline{w}$ and $\overline{a} \rightsquigarrow \overline{b}$. Clearly, the shortest path from $\overline{a}$ to $\overline{b}$ is $a_{t_1} - u_{t_1} - w_{t_1} - b_{t_1}$. Moreover, it is also easy to see that one of the shortest paths between $\overline{a}$ and $\overline{w}$ is $a_{t_2} - w_{t_2}$. Thereby, demonstrating that the latter is not a subpath of the prior.

While, in general, the multi-objective shortest path problem may need exponential runtime, we can do better due to its unique structure (Tarapata, 2007).

Proposition 3.4. *Let $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$ be a globally homogeneous, instant-based temporal multiplex then for $v, w \in V(\mathcal{T})$ the length (and size) of all shortest path is comparable and thus the same.*

Proof. We know there exists exactly one G_{t_i} and G_{t_j} in $\mathcal{G}$ such that $v \in G_{t_i} \wedge w \in G_{t_j}$. If there does not exits a path between v and w, we are done. Otherwise, by global homogeneity and due to $\preceq$

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

being directed, we obtain $\forall v \rightsquigarrow_{\mathcal{T}} w \, |v \rightsquigarrow_{\mathcal{T}} w|_\omega = (\delta_\omega(G_{t_i}, G_{t_j}), x)$. As they only differ in x, all of them are comparable and by minimality all shortest paths have the same length. □

Proposition 3.5. *Let $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$ be a globally homogeneous, instant-based temporal multiplex. The problem of finding the shortest path with respect to $|\cdot|_\omega$ over $\mathcal{T}$ from v to w for $v, w \in V(\mathcal{T})$ can be reduced in polynomial time to the problem of finding the shortest path between v and w in the weighted directed graph.*

Proof. Consider the construction $D := \langle V(\mathcal{T}), E(\mathcal{T}), \omega \rangle$. This transformation is linear. Show for $p_T := (v \rightsquigarrow_{\mathcal{T}} w) \subseteq \mathcal{T}$, we have $|v \rightsquigarrow_{\mathcal{T}} w|_\omega$ is minimal $\iff$ for $p_D := (v \rightsquigarrow_D w) \subseteq D$ we have $|v \rightsquigarrow_D w|_\omega$ is minimal. We observe $|\lambda_{\mathcal{G}}(p_T)|_\omega = \delta_\omega(v, w)$. Moreover, every path between v and w only differs in $|\lambda_\Phi(p_T)|_\omega$. Since, $|p_D|_\omega = \delta_\omega(v, w) + |\lambda_\Phi(p_T)|_\omega$, we obtain $|p_T|_\omega$ minimal must be equivalent with $|p_D|_\omega$ minimal. □

Hence, finding the shortest path between two distinct vertices in a temporal graph can be solved by applying a variant of Dijkstra's algorithm and it is thus $O((n+m) \cdot log(n))$, where $|\mathcal{G}| \leq n = |V(\mathcal{T})|$ and $m = |E(\mathcal{T})|$ (Barbehenn, 1998). By using this knowledge, we can implement algorithms for computing the other shortest path problems. Even a naive implementation, that is, one that computes all distances between $\overline{v}_t$ and every member of $\overline{w}$ is in $O((m+n) \cdot n \cdot log(n))$. Hence, searching for all minimal paths with respect to $|\cdot|_\omega$ is bounded by the same complexity, as finding minimal elements is at most $O(n^2)$. By applying the same algorithm for all $\overline{v}_t \in \overline{v}$, we obtain at most $O((m+n) \cdot n^2 \cdot log(n))$. This serves only as a rough estimate to show polynomial membership and to justify this approach from a computational complexity point of view.

4. Discussion

For the purposes of this paper, we decided against using the classical notion of multiplex network as a basis for our approach and instead defined a tailored variant that explicitly considers our generalized notion of time. This allows us to model a temporal path as a subnetwork of the main temporal network, with the temporal path retaining the properties of being a temporal network itself. Moreover, as discussed in Taylor et al. (2017), one can add isolated "ghost vertices" to each graph G_t to obtain the (classical) multiplex property. Unfortunately, this implies that for certain measures, the existence of such ghost vertices must be accounted for. While the relation which is used for modeling time is transitive, the corresponding inter-layer edges are neither transitive nor symmetric. Intuitively, they reflect the movement of a vertex through time. Therefore, by accepting the multiplex property, vertex persistence is ensured, allowing for a more simple semantics of movement through time, that is, a vertex cannot pop in and out of existence as it pleases.

Furthermore, allowing for partial orders, and thus deviating from the conceptions of time common in temporal logic (e.g., linear and branching flows of time), has several benefits. Firstly, from an applicative point of view, it allows for the modeling of inconsistent data, as presented in Example 4.5 (see below). Secondly, by allowing for a partial ordering, it is (to some extent) possible to extend the class of problems that can be modeled by this stricter notion of temporal networks, to those that can be modeled by a more general form of a temporal network, that is, a network where the conditions on the inter-layer edges $\mathcal{R}$ are relaxed. Those more general temporal networks allow for edges that connect vertices that are neither in the same graph nor in the same equivalence class, for example, edges such as $e := (\overline{v}_{t_i}, \overline{w}_{t_j})$ where $t_i \prec t_j$ and $\overline{v} \neq \overline{w}$ (Michail, 2016). Semantically those kinds of edges can be understood as the modeling of choice, that is, there are multiple ways to enter a subsequent point in time. Unfortunately, this does not map onto the framework presented above, as such additional edges do not adhere to the clean separation of the temporal and the graph internal semantics. In some cases, such general

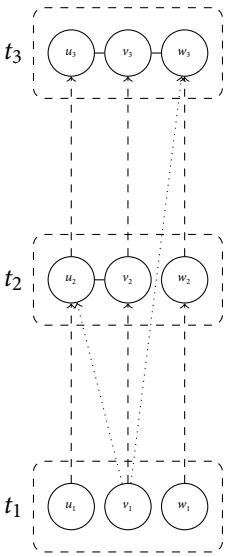
https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

temporal networks can be encoded as a temporal network as defined in this paper. As an example, an encoding that preserves distance with respect to a weight function ω will be presented below.

Let $\mathcal{T}^+ := \langle \mathcal{G}, \mathcal{R}^+, \equiv, \preceq, \omega \rangle$ be a relaxed version of a globally homogeneous temporal network. Meaning that $\mathcal{R}^+ := \mathcal{R} \cup \mathcal{R}^*$ where $\mathcal{R}$ is the usual collection of intra-class edge relations and $\mathcal{R}^*$ is the collection of inter-class, inter-layer edge relations. Let τ be a transformation from $\mathcal{T}^+ := \langle \mathcal{G}, \mathcal{R}^+, \equiv, \preceq, \omega \rangle$ to $\mathcal{T}' := \langle \mathcal{G}', \mathcal{R}', \equiv', \preceq', \omega' \rangle$ such that for all edges in $(\overline{v}_{t_i}, \overline{w}_{t_j}) \in E(\mathcal{R}^*)$ a new graph, that is, a new time step, between G_{t_i} and G_{t_j} will be added. Let this graph be called $G_{t_{v_i w_j}}$. This graph must contain the required amount of vertices as to satisfy the multiplex criteria, that is, exactly one vertex in the graph has to intersect with exactly one of the existing equivalence classes. Therefore, each equivalence class can be expanded by a single vertex from $G_{t_{v_i w_j}}$. Additionally, the graph $G_{t_{v_i w_j}}$ contains only a single edge, that is, the edge between $\overline{v}_{t_{v_i w_j}}$ and $\overline{w}_{t_{v_i w_j}}$, and this single edge simulates the inter-class component of the respective inter-layer, inter-class edge. Hence, its weight corresponds to the inter-class weight of the edge $(\overline{v}_{t_i}, \overline{w}_{t_j})$. Moreover, the predecessor relation is extended to account for $G_{t_{v_i w_j}}$ such that $G_{t_i} \prec' G_{t_{v_i w_j}} \prec' G_{t_j}$, while being incomparable with any other graph in $\mathcal{G}'$. Adding $G_{t_{v_i w_j}}$ implies that $\mathcal{R}'$ contains a set of inter-layer edges between G_{t_i} and $G_{t_{v_i w_j}}$, as well as between $G_{t_{v_i w_j}}$ and G_{t_j}. The edges within those sets are drawn according to the updated equivalence classes. This requires the extension of the function ω'. Meaning, it preforms identical to ω over $\mathcal{R}$ and for each newly added inter-layer edge the ω' positions $G_{t_{v_i w_j}}$ simply on the halfway point between G_{t_i} and G_{t_j}.

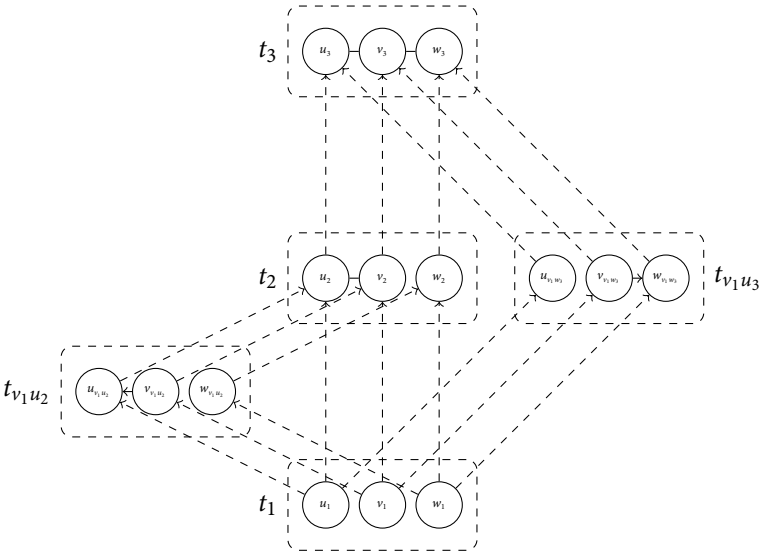
This roughly sketched transformation can be observed in Example 4.1.

Example 4.1. *Consider the following general temporal network. The inter-layer, intra-class edges, that is, $E(\mathcal{R})$ are dashed, while the inter-layer, inter-class edges, that is, $E(\mathcal{R}^*)$ are dotted and the normal intra-layer, intra-class edges, that is, $E(\mathcal{G})$ are solid:*



For example, we assume $\omega(v_1, u_2) := (4, 2)$, $\omega(v_1, w_3) := (8, 1)$ (remember that for the inter-class, inter-layer edges the weights have to be multivalued) and all other edges are assigned the weight 4. *After the transformation, one obtains the following temporal network:*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



where

- $\omega'(v_{v_1u_2}, u_{v_1u_2}) := 2$ *and* $\omega'(v_{v_1w_3}, w_{v_1w_3}) := 1$;
- *for all* $x \in \{u, v, w\}$ *it holds that* $\omega'(x_1, x_{v_1u_2}) = \omega'(x_{v_1u_2}, x_2) := 2$ *and* $\omega'(x_1, x_{v_1w_3}) = \omega'(x_{v_1w_3}, x_3) := 4$;
- *all other edges are assigned the weight* 4.

The transformation sketched above preserves distances between existing vertices. Consider a path p in $\mathcal{T}^+$. If p does not contain an edge $(\overline{v}_{t_i}, \overline{w}_{t_j}) \in E(\mathcal{R}^*)$, the same path exists in $\mathcal{T}'$ as well. Otherwise, let $t_n := t_{v_{t_i}w_{t_j}}$. Now for an edge $(\overline{v}_{t_i}, \overline{w}_{t_j})$ in p that is also in $E(\mathcal{R}^*)$, p will have the form $p = p_1, (\overline{v}_{t_i}, \overline{w}_{t_j}), p_2$, where p_1 and p_2 are the (possibly empty) paths before and after the edge $(\overline{v}_{t_i}, \overline{w}_{t_j})$. The edge $(\overline{v}_{t_i}, \overline{w}_{t_j})$ can now be replaced by $(\overline{v}'_{t_i}, \overline{v}'_{t_n}), \overline{v}'_{t_n}, (\overline{v}'_{t_n}, \overline{w}'_{t_n}), \overline{w}'_{t_n}, (\overline{w}'_{t_n}, \overline{w}'_{t_j})$, resulting in the path:

$$p' = p_1, (\overline{v}'_{t_i}, \overline{v}'_{t_n}), \overline{v}'_{t_n}, (\overline{v}'_{t_n}, \overline{w}'_{t_n}), \overline{w}'_{t_n}, (\overline{w}'_{t_n}, \overline{w}'_{t_j}), p_2$$

After exhaustive application of this substitution, one obtains a path in $\mathcal{T}'$. Let the temporal component of $\omega(\overline{v}_{t_i}, \overline{w}_{t_j})$ be d_Φ and the intra-layer component be $d_\mathcal{G}$. By definition, $d_\Phi = \frac{\alpha}{2} + \frac{\alpha}{2} = \omega'(\overline{v}'_{t_i}, \overline{v}'_{t_n}) + \omega'(\overline{w}'_{t_n}, \overline{w}'_{t_j})$ and $d_\mathcal{G} = \omega'(\overline{v}'_{t_n}, \overline{w}'_{t_n})$. Lastly, each newly added graph contains only a single edge. Hence, all paths p' in $\mathcal{T}'$ that contain a sequence $(\overline{v}'_{t_i}, \overline{v}'_{t_n}), \overline{v}'_{t_n}, (\overline{v}'_{t_n}, \overline{v}'_{t_j})$ have a length equivalent path in $\mathcal{T}^+$ that only uses intra-class edges from $\overline{v}_{t_i}$ to $\overline{v}_{t_j}$. To conclude, one can use this procedure to transform this general form of temporal networks into the one presented in this paper, without distorting the distances between vertices.

Future work may discuss possible ramifications with respect to common network measures. For example, by only considering the multivalued notion of distance on linear time, one can already detect different behaviors of some network measures. That is, consider the path-based centrality measures closeness and betweenness. As betweenness relies on counts of shortest paths, its extension to a multivalued path length is straightforward. Even in linear time, the various shortest path problems result in analogous betweenness measures, that is, with and without restriction on the start and end time. In contrast, while closeness centrality has a similar relationship with respect to time, any ranking of vertices will be a partial ordering. That is, since the length of a path is incorporated into this measure, its multivalued nature will be carried over. Lastly, while

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

interesting, a discussion of network measures on nonlinear models of time is, unfortunately, far beyond the scope of this paper.

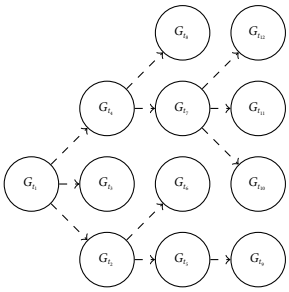
4.1 Areas of application

In this section, we briefly discuss possible areas of application for our approach, as well as possible advantages that may emerge when reasoning about temporal networks in a more abstract manner.

4.1.1 Connections to modal logic

Firstly, we present an example of how this more abstracted perspective toward temporal networks allows for the use of temporal logic to reason about temporal networks. This particular case is concerned with centrality measures. One utility provided by those measures is that they can be used to impose an ordering onto vertices based on their perceived importance. In the subsequent example, we use temporal logic to construct an ordering of vertices based on their orderings imposed by multiple centrality measures at multiple points in time. That is, we use propositional logic to encode information about the structure of the finite network at each point in time. The structural information, in this case the ordering of vertices based on centrality measures, is used in a temporal logic sentence to formally and unambiguously compose a new ordering of vertices that explicitly engages with the temporal dimension.

Example 4.2. *Consider the following flow of time* $\Phi := \langle T, R, \iota \rangle$*, where finite graphs are encoded in propositional logic at each point in time:*



Moreover, let $C_{E_{xy}}$ *be the a predicate indicating that the eigenvector centrality* C_E *of vertex x is smaller than the eigenvector centrality of vertex y, that is,* $C_E(x) < C_E(y)$*. Similarly,* $C_{K_{xy}}$ *indicates the same for the Katz centrality* C_K*. Furthermore, those predicates evaluate to true at the following states:*

$$\Phi, G_{t_x} \vDash C_{E_{ab}} \qquad x \in \{2, 3, 4, 6, 8, 9, 10, 11, 12\}$$
$$\Phi, G_{t_x} \vDash C_{K_{ab}} \qquad x \in \{2, 3, 4, 7, 9, 12\}$$

Consider the subsequent sentence in propositional temporal logic:

$$C_{N_{ab}} \leftrightarrow (A\Diamond(A\Box\, C_{E_{ab}}) \wedge A\bigcirc C_{K_{ab}} \wedge E\Box(C_{K_{ab}} \rightarrow C_{E_{ab}}))$$

The right part of the equivalence expresses the following:

- $A\Diamond(A\Box\, C_{E_{ab}})$ *requires that on all branches at some point in the future* $(A\Diamond)$*, it must be the case that on all branches in all subsequent points in the future* $(A\Box)$ *the inequality* $C_E(a) < C_E(b)$ *is satisfied.*
- $A\bigcirc C_{K_{ab}}$ *requires that on all branches in the next point in time* $(A\bigcirc)$*, it must be the case that* $C_K(a) < C_K(b)$*.*
- $E\Box(C_{K_{ab}} \rightarrow C_{E_{ab}})$ *requires that in all subsequent points on some branch* $(E\Box)$*, it must be the case that* $C_K(a) < C_K(b)$ *implies* $C_E(a) < C_E(b)$*.*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

$A\Diamond(A\Box\, C_{E_{ab}})$ is satisfied, because $C_{E_{ab}}$ holds in 3, 8, 9, 10, 11, *and* 12. *$A\bigcirc\, C_{K_{ab}}$ is satisfied, because $C_{K_{ab}}$ holds in* 2, 3, *and* 4. *And $E\Box(C_{K_{ab}} \to C_{E_{ab}})$ is satisfied, because $C_{K_{ab}}$ holds in* 2 *and* 9, *and $C_{E_{ab}}$ holds in* 2 *and* 9 *as well. Given this $\Phi, G_{t_1} \vDash C_{N_{ab}}$ holds. Hence, according to the "new" centrality measure C_N vertex a is less important than the vertex b, see also Kröger (2012).*

There are two apparent applications of logic in temporal networks. Namely, as a specification and as a query language. To illustrate this, consider a trace in a communication network consisting of client, router, and backup router. From a monitoring perspective, desirable properties of that trace could be "It must always be true that a backup router can only receive messages, if the corresponding (main) router is down." or "If the router shuts down, the backup router has to be active in the next time step." Second, temporal logic as a querying language. Here, we might be interested in the points in time where a critical server is operating at maximum capacity, for example, such a question would amount to checking whether the server has a high betweenness centrality and a degree centrality equal to its capacity.

Both examples are restricted to a linear flow of time. However, if we would be aware of the protocol applied in the communication network, we would be able to model the development of the network by utilizing branching, that is, from each state there are several possible future states due to potential nondeterministic properties of the protocol. Hence, to verify whether a particular network satisfies a desirable property specified using temporal logic, we simply construct the temporal network given the rules of the protocol and check whether the property holds.

Another example for the use of temporal logic in network verification is Panda et al. (2015); here, they use logic to specify reachability invariants such as "It must always be true that a send message is always received." Moreover, a nice example for utilizing temporal logic as a query language is provided in Monteiro et al. (2008) , where it is applied to query dynamic cellular interaction networks. Lastly, it should be mentioned that the intersections between logic and network science are not restricted to temporal networks of course, for example, Plotkin et al. (2016) apply verification techniques to switching networks, or Seligman et al. (2011) who use a modal logic to reason about knowledge in social networks.

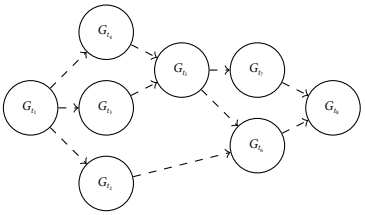
4.1.2 Planning problems

Given the task of finding an optimal strategy for some planning problem, it is possible to map choices and their consequences onto a tree-like structure. That is, by fixing ones move, the set of possible subsequent decisions will be restricted accordingly, with each of those decisions being represented as a branch in the decision tree (see, e.g., Van Benthem, 2014). Within the context of temporal networks, this is mirrored by future branching flows in time. Additionally, if there is the desire to express that two strategies have identical outcomes after a certain point, one can use flows in time with joins. That is, one can use branching to model historic dependencies, and the joining of branches to weaken those historic dependencies.

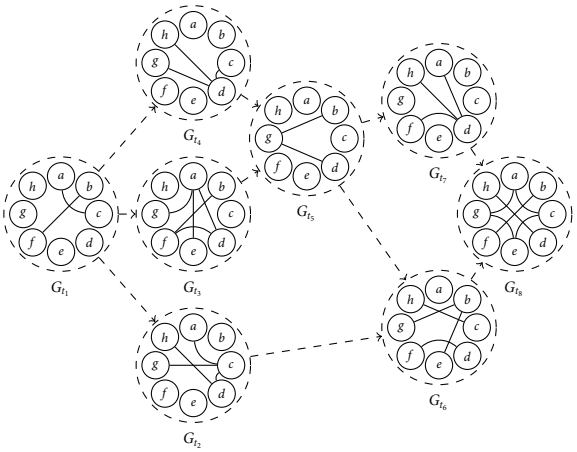
Example 4.3. *A group of people $P := \{a, b, c, d, e, f, g, h\}$ have to pass a token t from person a to person b such that the duration and the risk, for example, the risk of being detected or the risk of damaging the token is minimized. Both, time and risk are encoded by the weight function. Moreover, the set of possible moves is limited and context-dependent. We can model the set of possible transactions via weighted graphs and express context dependency by arranging the graphs within a suitable nonlinear flow of time around which a corresponding weighted, instant-based temporal multiplex is build. Moreover, using this encoding, a possible optimal strategy corresponds to a minimal temporal path problem.*

Consider the following weighted, instant-based temporal multiplex $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega \rangle$:

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



with the respective graphs being (note that the graphs depicted here do not represent the actual flow of time, because the description of the world is incomplete, that is, the token position is not encoded.):



Moreover, for the purpose of this example, let $\omega(a_{t_2}, c_{t_2}) := 2$, $\omega(c_{t_2}, d_{t_2}) := 8$, $\omega(a_{t_3}, d_{t_3}) := 8$, $\omega(d_{t_3}, f_{t_3}) := 6$, $\omega(a_{t_3}, g_{t_3}) := 3$, $\omega(d_{t_5}, g_{t_5}) := 2$, $\omega(b_{t_5}, g_{t_5}) := 4$, $\omega(b_{t_6}, e_{t_6}) := 4$, $\omega(b_{t_6}, g_{t_6}) := 4$ *and* 1 *for all remaining edges.*

As mentioned above, if one accepts this temporal network as is, one will encounter an issue with respect to computational complexity. That is, we now have two paths with different durations between two points in time. In particular, one can reach G_{t_6} *from* G_{t_1} *through* $G_{t_1} - G_{t_4} - G_{t_5} - G_{t_6}$, *through* $G_{t_1} - G_{t_3} - G_{t_5} - G_{t_6}$, *and through* $G_{t_1} - G_{t_2} - G_{t_6}$. *Meaning that, if one leaves the weight of the temporal edges uniform, global homogeneity is lost. Hence, the property required for an efficient computation of the shortest path is no longer satisfied. We can restore global homogeneity by simply setting* $\omega(G_{t_2}, G_{t_6}) = 2$.

Having resolved this issue, one can now move on toward computing the multiple minimal paths within this temporal network. This, however, requires additional thought, as within a temporal dimension there are multiple minimal path problems to choose from. Since the entry point of the token game is fixed, we can reject the minimal path problem:

$$\min_{(|\overline{a} \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega})_{\Phi}} \min_{(|\overline{a} \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega})_{\mathcal{G}}} |\overline{a} \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega}$$

Moreover, since the problem at hand requires flexibility with respect to the end point, we have to settle for

$$\min_{(|\overline{a}_{t_1} \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega})_{\Phi}} \min_{(|\overline{a}_{t_1} \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega})_{\mathcal{G}}} |\overline{a}_1 \rightsquigarrow_{\mathcal{T}} \overline{b}|_{\omega}$$

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

To compute the solution to this problem, one first has to compute all shortest path from $\overline{a}_{t_1}$ to each element in $\overline{b}$, that is,

$$|p_1|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_3} - \overline{d}_{t_3} - \overline{f}_{t_3} - \overline{b}_{t_3}|_\omega = (1, 15)$$

$$|p_2|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_3} - \overline{g}_{t_3} - \overline{g}_{t_5} - \overline{b}_{t_5}|_\omega = (2, 7)$$

$$|p_3|_\omega = |\overline{a}_{t_1} - \overline{c}_{t_1} - \overline{c}_{t_4} - \overline{d}_{t_4} - \overline{g}_{t_4} - \overline{g}_{t_5} - \overline{b}_{t_5}|_\omega = (2, 7)$$

$$|p_4|_\omega = |\overline{a}_{t_1} - \overline{c}_{t_1} - \overline{c}_{t_2} - \overline{g}_{t_2} - \overline{g}_{t_6} - \overline{b}_{t_6}|_\omega = (3, 5)$$

$$|p_5|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_3} - \overline{e}_{t_3} - \overline{e}_{t_5} - \overline{e}_{t_6} - \overline{b}_{t_6}|_\omega = (3, 5)$$

$$|p_6|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_3} - \overline{g}_{t_3} - \overline{g}_{t_5} - \overline{b}_{t_5} - \overline{b}_{t_7}|_\omega = (3, 7)$$

$$|p_7|_\omega = |\overline{a}_{t_1} - \overline{c}_{t_1} - \overline{c}_{t_4} - \overline{d}_{t_4} - \overline{g}_{t_4} - \overline{g}_{t_5} - \overline{b}_{t_5} - \overline{b}_{t_7}|_\omega = (3, 7)$$

$$|p_8|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_3} - \overline{a}_{t_5} - \overline{a}_{t_7} - \overline{d}_{t_7} - \overline{f}_{t_7} - \overline{f}_{t_8} - \overline{b}_{t_8}|_\omega = (4, 3)$$

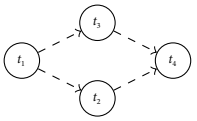
$$|p_9|_\omega = |\overline{a}_{t_1} - \overline{a}_{t_4} - \overline{a}_{t_5} - \overline{a}_{t_7} - \overline{d}_{t_7} - \overline{f}_{t_7} - \overline{f}_{t_8} - \overline{b}_{t_8}|_\omega = (4, 3)$$

Now, since $(3, 5) \leq (3, 7)$, we can reject the paths p_6 and p_7. Hence, with the remaining being minimal paths, we have found the set of optimal paths, which reflect the set of optimal strategies for passing the token from a to b. There are two important things to consider. Firstly, we can observe that p_2 and p_3 enter G_{t_5} on the same position. That is, those two paths represent two different strategies leading to exactly the same outcome, as both of them end in the same world. Hence, rather than just obtaining the same minimal distance, one actually reaches the exact same state of the world. This is reflected by the fact that p_6 and p_7 are merely p_2 and p_3 with identical extensions. By contrast, p_4 and p_5 do not produce the same outcome. Meaning that, while those two path share the same temporal distance and risk, they do not end in precisely the same world (note that it merely seems as such, because the token positions are not encoded in our image).

4.1.3 Modeling of belief

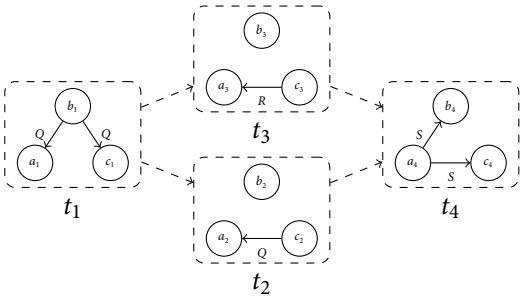
By considering nonlinear flows in time, one can reconcile uncertainty and discrete objects. One of such objects in question are logical formulas. Such formulas may be used to model and express the belief of agents. Such nondeterminism is especially important, if one approximates human reasoning by relying on nonclassical forms of logic, such as non-monotonic reasoning. For more on non-monotonic reasoning, see, for example, Bochman (2011). Within some non-monotonic formalisms, for example, default logic, one can obtain multiple different maximal consistent sets of conclusions from the same description of the world (Gottlob, 1992). Hence, one can use branches to encode the mutually exclusive sets of beliefs of an agent into a temporal graph.

Example 4.4. *Let $P := \{a, b, c\}$ be a set of people. Consider the following flow of time:*



Within each point in time, we are going to embed the following graphs (note that this merely represents a fragment of the actual network, as the internal beliefs of each agent are not encoded within the representations below):

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



Firstly, we limit ourself to two mutually exclusive beliefs Q and R, that is, $Q \wedge R \vdash \bot$, as well as the belief S, where $R \rightarrow S$ holds with our theory. For example, Q could be the belief that there have been humans on the moon, whereas R would be the belief in the conspiracy that there have never been humans on the moon. Clearly, if one beliefs in the conspiracy R, one must also accept the conspiracy that the moon lading was fake, that is, $R \rightarrow S$.

Secondly, the expression of a belief is represented by an edge. Thirdly, an assumption underlying this model is that, if an agent advocates for a belief, this belief must have been accepted as true by the respective agent. By contrast, the recipient may or may not adopt the communicated belief. Fourthly, every agent starts with a consistent sets of beliefs. Lastly, a belief can be obtained externally through communication, that is, another agent has to communicate this belief to the agent in question, and internally through deductive reasoning. In this case, if an agent beliefs R, the agent must also accept S.

In this example, we want to decide whether agent c adopted the belief Q at t_1. Assume that the initial sets of beliefs are $\{\varnothing\}$ for a, $\{\{Q\}\}$ for b and $\{\{R\}\}$ for c. Hence, after closing each set of beliefs deductively, one obtains at point t_1 that S must be held by c as well. From there, beliefs are communicated, resulting in $a_2 = a_3 = \{Cl(\{Q\}), Cl(\varnothing)\}$, $b_2 = b_3 = \{Cl(\{Q\})\}$, $c_2 = \{Cl(\{Q\})\}$, and $c_3 = \{Cl(\{R, S\})\}$ at t_2 and t_3, respectively.[5] Hence, one step away from the origin, we know that agent c has either of two consistent sets of belief, that is, $Cl(\{R, S\})$ and $Cl(\{Q\})$, which are separated into two possible worlds. That is, we have encoded all possible consistent set of beliefs of the agent c into the structure of the network. Lastly, in t_4, we observe that agent a communicates belief S. Knowing that S was never communicated and was not previously held by a, the only possible method of acquisition would be internal reasoning, which requires the assumption of R. However, there exists only one possible point in time, explaining the adoption of the belief R by agent a and that is t_3. Hence, we have to accept t_3 as reality and discard t_2. If this example would not focus solely on c, a similar encoding and elimination ought to be done for agent a as well. Moreover, as Q and R are mutually exclusive, the only possible way for c to communicate R is, if it rejected Q at t_1. Thereby, answering the problem stated above.

This example tries to highlight the benefits of encoding a temporal network in such a manner that it can easily be translated into various logical formalisms, here, in particular for the modeling of agents and beliefs in a network setting using non-monotonic inference mechanisms such as answer set programming. In fact, there have been several investigations into the utility of such inference mechanisms for the modeling of social behavior. For example, De Vos & Vermeir (2003) used logic programming to model the development of beliefs in a network; Costantini & Tocchio (2004) introduced DALI a specific language for the modeling of agents and their reaction to events in a rudimentary temporal context, that is, all events that occur to an agent are time-stamped, which allows an agent to remember whether it reacted to an incoming event. However, a number of other publications exist that use non-monotonic logic for the study of multi-agent systems, for example, Buccafurri & Caminiti (2005) and Cliffe et al. (2005).

The approach presented in Lopes et al. (2015) is remotely related to the idea sketched in the example above. That is, they introduce Network Optimized Datalog to allow for a system

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

administrator to verify his/her own beliefs over a dynamic network structure, for example, "is it true that that internal controllers cannot be accessed from the Internet?."

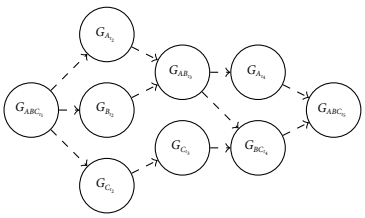
4.1.4 Inconsistent data

Lastly, we discuss a formal approach for dealing with inconsistent data. That is, we use branching in the flow of time to model different measurements of the same reality at the same point in time, while the joining of branches can be used to model agreement of those measurements. Moreover, by encoding inconsistencies of the data into our formal framework, we can use formalisms such as temporal logic to identify inconsistent measurement sequences in an automated manner.

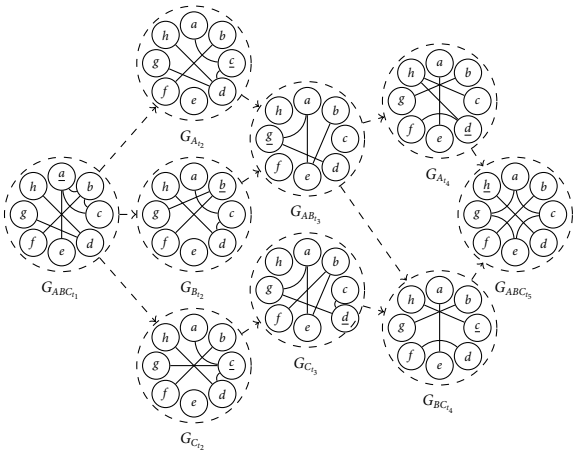
Example 4.5. *A group of people* $P := \{a, b, c, d, e, f, g, h\}$ *have to pass a token t from person to person. There are three data sources. Each data point containing the position of the token, as well as the interactions between people through which the token may be passed. Moreover, we know that the following rules apply*

(1) A token can only be passed over an edge.
(2) A token cannot be passed to the same person twice.
(3) After each token exchange, time moves forward by one step.

Consider the following weighted, instant-based temporal multiplex $\mathcal{T} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq, \omega_1 \rangle$ *representing the measurements, where joined points in time represent agreement among the respective data sources, while branching encodes disagreements between sources:*



with the corresponding graphs being (note the underline is representing the respective position of the token):



https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

If we now consider each data source separately, we recognize that, given the rules of the system, neither is sufficient to explain the phenomena of transferring the token from person a to person h. That is, source A reports that in step $G_{A_{t_2}}$ to $G_{AB_{t_3}}$ the token was transferred from person c to person g without a connecting edge. Similarly, according to source B, the token jumped from g to c in step $G_{AB_{t_3}}$ to $G_{BC_{t_4}}$. By contrast, source C reported that person c received the token twice. How to resolve those issues is highly case-specific. Considering probabilities, or methods to fuzzify the underlying problem, even though possible, would compromise the inherently discrete nature of the token. Another line of thought would be to select the edges from all data sources in such a manner that the resulting model explains the behavior of the system, for example, by taking the union of all edges at a given point in time. Another method could be to find all possible paths from the first point in time to the last point in time, such that the laws of the system are satisfied. The underlying idea being, that if two data sources agree at one instant, both possible futures can be considered as future steps. Hence, in this case, we would obtain the valid sequence:

At this point, it is important to note that we are not interested in arguing for a particular method of aggregation. The main objective of the previous example is to demonstrate some benefits of thinking about temporal networks in a more abstract manner. Moreover, as mentioned above, it may serve as a small step towards logic-based methods of consolidating data sources with in the realm of temporal networks.

4.2 Tool support

In addition to the theoretical work presented above, we implemented a corresponding R-package[6] (Wickham, 2015). The primary purpose of this package is to provide a data structure that allows for convenient storage and manipulation of flows of time and temporal networks based on such flows of time (as discussed in Sections 2 and 3). The package, called `tempnetwork`, is mostly comprised of two S3-classes (Wickham, 2014, sec. 1, ch. 7). The first, called `tempflow`, is concerned with the storage and manipulation of flows of time. The second, called `tempnetwork`, builds upon the `tempflow`-class in a similar manner as the presented temporal networks build upon flows of time. Meaning, it uses the structure provided by the `tempflow`-object to construct a temporal network from a suitable family of graphs and a suitable equivalence relation. The `tempnetwork`-package was implemented as an extension of the `igraph`-package (Csardi & Nepusz, 2006). However, particular care was given to limit the dependencies to the `igraph`-package only. A brief documentation and some examples are given in Appendix B.

5. Related work

The concept of a temporal network was developed in parallel across various fields, resulting in multiple formalisms expressing the same or a similar idea (see, for example, Tang et al., 2009; Michail, 2016; Kempe et al., 2002, and Kostakos, 2009). To obtain an overview over such a diverse body of literature, the reader is referred to Holme & Saramäki (2012), which provides an extensive summary of the concepts related to temporal networks. Moreover, Casteigts et al. (2012) not only provides an overview for some formalisms but also attempts to unify those formalisms.

Another attempt of unifying this diverse ecosystem of formalisms is the notion of a multilayer network. In this context, Kivelä et al. (2014) and Boccaletti et al. (2014) can be recommended, if one is interested in two comprehensive surveys of the field. However, for a quick introduction into the topic, Aleta & Moreno (2018) and Tomasini (2015) seem to be more convenient. Naturally, even if the notion of a multilayer network is an attempt at unification, there still exist

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

several formalizations of the same concept, see, for example, Solá et al. (2013), Wang et al. (2017), Spatocco et al. (2018), Fenu & Higham (2017) and Kivelä & Porter (2018), Tomasini (2015) and Cozzo et al. (2016). Particularly interesting are De Domenico et al. (2013), Solé-Ribalta et al. (2016), as well as Wang et al. (2017) and Aleta & Moreno (2018) who discuss the tensor notation of multilayer networks.

The notion of a path is central not only in graph theory, but also in multilayer networks and thus also in temporal networks. Two insightful papers that specifically discuss paths in temporal networks are Tang et al. (2009) and Wu et al. (2014). In particular, Wu et al. (2014) presents various possible conceptions of what a minimal temporal path may be, and Tang et al. (2009) discusses what kinds of metrics could be used for measuring distances in temporal graphs. Moreover, as the notion of path is integral for the computation of some centrality measures, the discussion of what a minimal path in a temporal network may be, thus projects forward and influences the definitions of some common centrality measures. For example, Taylor et al. (2017) tries to define an eigenvector centrality that differentiates between inter- and intra-graph edges. For a more computational perspective on temporal paths and temporal networks in general (Michail, 2016) can be recommended.

While some definitions of temporal networks neglect to study the temporal dimension in a rigorous and explicit manner, there are two definitions that do study the temporal dimension explicitly. Namely, temporal even graphs (Mellor, 2018) and stream graphs (Latapy et al., 2018). The prior is a statical encoding of a temporal graph, that is well suited for computing reachability. This property was exploited in Badie-Modiri et al. (2020) to estimate in- or out-reachability with limited waiting times. A temporal event graph is computed by transforming the events in the temporal graph (i.e., interactions between two vertices at some point in time) into vertices and connecting them based on temporal adjacency, that is, two events are connected if they share at least one vertex and they occur within Δ time of each other. Hence, the connectivity of the event graph correlates positively with Δ. The event graph framework was extended to weighted temporal event graphs in Kivelä et al. (2018). The latter cleanly separates both the temporal and the structural aspects of a temporal network. This is accomplished by essentially generalizing important graph theoretical concepts such that they can accommodate temporal aspects.

A stream graph (Latapy et al., 2018) can be understood as a graph where vertices as well as edges are annotated with time stamps. If the set of time stamps is continuous, both edges and vertices now are able to persist over time intervals. Moreover, stream graphs are used to to investigate the behavior of multi-valued paths, for example, Latapy et al. (2018) introduce interesting concepts such as the shortest fastest path and the fastest shortest path.

In particular, the link graphs introduced in Latapy et al. (2018), which are stream graphs with temporally invariant vertices, share similarities with our approach. In fact, it is easy to see that if we restrict the flow of time to be linear, our approach is subsumed by link graphs, in particular, because we do not allow continuous time. However, to the best of our knowledge, the stream graph approach does not account for the nondeterminism as discussed in this paper. Hence, despite the similarities, such as a clean separation of the temporal and structural components of the temporal network and the investigations into multi-valued paths, our work remains noticeable distinct and both approaches complement each other.

As our approach also makes a (small) step toward the combined study of networks, logic, and temporal logic (see Section 4), the reader may be interested in Seligman et al. (2011), Ahuja & Malhi (2016), and Pardo et al. (2018) all of which discuss possible uses of logic in the study of networks. Furthermore, for a quick introduction into temporal logic, the reader is referred to Venema (2017), Burgess (1979), and Goranko & Galton (2015). Moreover, for a detailed review of some of the many variants of temporal logic, see Kröger (2012). As temporal logic is intimately intertwined with modal logic, the reader may also be interested in Van Benthem et al. (2010) which provides an introduction into modal logic, as well as a good overview over its areas of application.

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

6. Conclusion

Most approaches for the analysis of temporal networks do not explicitly discuss the underlying conception of time. Moreover, weighted temporal networks are still uncommon in the literature, and a direct discussion about how to reconcile the two semantic dimensions seems to be even more rare. In order to tackle those issues, this paper discussed time as a formal structure, thereby explicitly engaging with some of the underlying assumptions of time on which a temporal network may operate. The utility of which was investigated by presenting a variety of examples, each reflecting a possible area of application. Furthermore, we implemented a corresponding R-package (for details, see Appendix B) which provides tool support for the concepts introduced in this paper and thereby enables the modeling and analysis of discrete temporal network.

Together with our approach, we discussed some of the pitfalls that arise when dealing with nondeterministic time. Moreover, our generalized abstraction of time promotes a clean separation of the semantics of time and the semantic interpretation of the network itself (i.e., the semantics of the vertices and edges in the corresponding network). To retain a clean separation of dimensions, we introduced the notion of multivalued (temporal) paths, a variant of paths that, on the one hand, enables a more in-depth understanding of temporal networks without sacrificing semantic integrity, while on the other hand introducing several new and interesting technical challenges, such as the computation of multivalued centrality measures for temporal networks. In addition, investigations into the connections between stream graphs (Latapy et al., 2018) and interval temporal logic (Cau et al., 2006) could further strengthen the ties between logic and temporal graphs.

Competing interests. None

Notes

1 The R-package is publicly available from https://nm.wu.ac.at/nm/strembeck/RTempNet/tempnetwork.zip.
2 $|\cdot|$ is shorthand for $|x \rightsquigarrow y|$ for some path $x \rightsquigarrow y$.
3 $a \rightsquigarrow_X b$ is shorthand for $a \rightsquigarrow b \subseteq X$.
4 For a definition of $\overline{a}_{t_i}$, see Definition 1.2.
5 $Cl(X) := \{\varphi \mid X \vDash \varphi\}$ is the deductive closure of X.
6 The source code is publicly available at https://nm.wu.ac.at/nm/strembeck/RTempNet/tempnetwork.zip.

References

Ahuja, M., & Malhi, H. (2016). Predicting links in complex network using fuzzy logic. *International Journal of Computer Trends and Technology*, *36*, 158–162.

Aleta, A., & Moreno, Y. (2018). Multilayer networks in a nutshell. *Annual Review of Condensed Matter Physics, 10*, 45–62.

Badie-Modiri, A., Karsai, M., & Kivelä, M. (2020). Efficient limited-time reachability estimation in temporal networks. *Physical Review E*, *101*(5), 052303.

Barbehenn, M. (1998). A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, *47*(2), 263.

Boccaletti, S., Bianconi, G., Criado, R., Del Genio, C. I., Gómez-Gardenes, J., Romance, M., Sendina-Nadal, I., Wang, Z., & Zanin, M. (2014). The structure and dynamics of multilayer networks. *Physics Reports*, *544*(1), 1–122.

Bochman, A. (2011). Logic in nonmonotonic reasoning. In *Nonmonotonic reasoning. Essays celebrating its 30th anniversary* (pp. 25–61). Rickmansworth, England: College Publications.

Buccafurri, F., & Caminiti, G. (2005). A social semantics for multi-agent systems. In *International conference on logic programming and nonmonotonic reasoning* (pp. 317–329). Berlin, Heidelberg: Springer.

Burgess, J. P. (1979). Logic and time 1. *The Journal of Symbolic Logic*, *44*(4), 566–582.

Casteigts, A., Flocchini, P., Quattrociocchi, W., & Santoro, N. (2012). Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, *27*(5), 387–408.

Cau, A., Moszkowski, B., & Zedan, H. (2006). Interval temporal logic. Retrieved from http://www. cms. dmu. ac. uk/čau/itlhomepage/itlhomepage.html.

Cliffe, O., De Vos, M., & Padget, J. (2005). Specifying and analysing agent-based social institutions using answer set programming. In *International conference on autonomous agents and multiagent systems* (pp. 99–113). Berlin, Heidelberg: Springer.

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. Cambridge, MA and London, England: MIT Press.
Costantini, S., & Tocchio, A. (2004). The dali logic programming agent-oriented language. In *European workshop on logics in artificial intelligence* (pp. 685–688). Berlin, Heidelberg: Springer.
Cozzo, E., de Arruda, G. F., Rodrigues, F. A., & Moreno, Y. (2016). Multilayer networks: Metrics and spectral properties. In *Interconnected networks* (pp. 17–35). Berlin, Heidelberg: Springer.
Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*, *1695*(5), 1–9.
De Domenico, M., Solé-Ribalta, A., Cozzo, E., Kivelä, M., Moreno, Y., Porter, M. A., Gómez, S., & Arenas, A. (2013). Mathematical formulation of multilayer networks. *Physical Review X*, *3*(4), 041022.
De Vos, M., & Vermeir, D. (2003). Logic programming agents playing games. In *Research and development in intelligent systems XIX* (pp. 323–336). Berlin, Heidelberg: Springer.
Dickson, L. E. (1913). Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, *35*(4), 413–422.
Fenu, C., & Higham, D. J. (2017). Block matrix formulations for evolving networks. *SIAM Journal on Matrix Analysis and Applications*, *38*(2), 343–360.
Gödel, K. (1949). An example of a new type of cosmological solutions of Einstein's field equations of gravitation. *Reviews of Modern Physics*, *21*(3), 447.
Goranko, V., & Galton, A. (2015). Temporal logic. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (winter 2015 ed.). Stanford, CA: Metaphysics Research Lab, Stanford University.
Goshtasby, A. A. (2012). Similarity and dissimilarity measures. In *Image registration* (pp. 7–66). Berlin, Heidelberg: Springer.
Gottlob, G. (1992). Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, *2*(3), 397–425.
Holme, P., & Saramäki, J. (2012). Temporal networks. *Physics Reports*, *519*(3), 97–125.
Kempe, D., Kleinberg, J., & Kumar, A. (2002). Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, *64*(4), 820–842.
Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J. P., Moreno, Y., & Porter, M. A. (2014). Multilayer networks. *Journal of Complex Networks*, *2*(3), 203–271.
Kivelä, M., Cambe, J., Saramäki, J., & Karsai, M. (2018). Mapping temporal-network percolation to weighted, static event graphs. *Scientific Reports*, *8*(1), 1–9.
Kivelä, M. & Porter, M. A. (2018). Isomorphisms in multilayer networks. *IEEE Transactions on Network Science and Engineering*, *5*(3), 198–211.
Kostakos, V. (2009). Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, *388*(6), 1007–1023.
Kröger, F. (2012). *Temporal logic of programs*, vol. 8. Berlin, Heidelberg: Springer Science & Business Media.
Kueffner, K. & Strembeck, M. (2019). A generalized notion of time for modeling temporal networks. In *Proceedings of the 4th international conference on complexity, future information systems and risk (COMPLEXIS)*. Setúbal, Portugal.
Latapy, M., Viard, T., & Magnien, C. (2018). Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, *8*(1), 1–29.
Lopes, N. P., Bjørner, N., Godefroid, P., Jayaraman, K., & Varghese, G. (2015). Checking beliefs in dynamic networks. In *12th {USENIX} symposium on networked systems design and implementation ({NSDI} 15)* (pp. 499–512). Berkeley, CA.
Markosian, N., Sullivan, M., & Emery, N. (2016). Time. In E. N. Zalta (ed.), *The Stanford encyclopedia of philosophy* (fall 2016 ed.). Stanford, CA: Metaphysics Research Lab, Stanford University.
Matthews, S. G. (1994). Partial metric topology. *Annals of the New York Academy of Sciences*, *728*(1), 183–197.
Mellor, A. (2018). The temporal event graph. *Journal of Complex Networks*, *6*(4), 639–659.
Michail, O. (2016). An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, *12*(4), 239–280.
Monteiro, P. T., Ropers, D., Mateescu, R., Freitas, A. T., & De Jong, H. (2008). Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, *24*(16), i227–i233.
Panda, A., Argyraki, K., Sagiv, M., Schapira, M., & Shenker, S. (2015). New directions for network verification. In *1st summit on advances in programming languages (SNAPL 2015)*. Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
Pardo, R., Sánchez, C., & Schneider, G. (2018). Timed epistemic knowledge bases for social networks. In *International symposium on formal methods* (pp. 185–202). Berlin, Heidelberg: Springer.
Plotkin, G. D., Bjørner, N., Lopes, N. P., Rybalchenko, A., & Varghese, G. (2016). Scaling network verification using symmetry and surgery. *ACM SIGPLAN Notices*, *51*(1), 69–83.
Runkler, T. A. (2012). *Data analytics*. Berlin, Heidelberg: Springer.
Schroeder, V. (2006). Quasi-metric and metric spaces. *Conformal Geometry and Dynamics of the American Mathematical Society*, *10*(18), 355–360.
Segarra, S., & Ribeiro, A. (2016). Stability and continuity of centrality measures in weighted graphs. *IEEE Transactions on Signal Processing*, *64*(3), 543–555.
Seligman, J., Liu, F., & Girard, P. (2011). Logic in the community. In *Indian conference on logic and its applications* (pp. 178–188). Berlin, Heidelberg: Springer.

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Solá, L., Romance, M., Criado, R., Flores, J., García del Amo, A., & Boccaletti, S. (2013). Eigenvector centrality of nodes in multiplex networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *23*(3), 033131.
Solé-Ribalta, A., De Domenico, M., Gómez, S., & Arenas, A. (2016). Random walk centrality in interconnected multilayer networks. *Physica D: Nonlinear Phenomena*, *323*, 73–79.
Spatocco, C., Stilo, G., & Domeniconi, C. (2018). A new framework for centrality measures in multiplex networks. arXiv preprint arXiv:1801.08026.
Sunitha, M., & Mathew, S. (2013). Fuzzy graph theory: A survey. *Annals of Pure and Applied mathematics*, *4*(1), 92–110.
Tang, J., Musolesi, M., Mascolo, C., & Latora, V. (2009). Temporal distance metrics for social network analysis. In *Proceedings of the 2nd ACM workshop on Online social networks* (pp. 31–36). New York City, NY: ACM.
Tarapata, Z. (2007). Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science*, *17*(2), 269–287.
Taylor, D., Myers, S. A., Clauset, A., Porter, M. A., & Mucha, P. J. (2017). Eigenvector-based centrality measures for temporal networks. *Multiscale Modeling & Simulation*, *15*(1), 537–574.
Tomasini, M. (2015). An introduction to multilayer networks. *BioComplex Laboratory, Florida Institute of Technology, Melbourne, USA* (pp. 1–14).
Van Benthem, J. (2014). *Logic in games*. Cambridge, MA: MIT Press.
Van Benthem, J., van Benthem, J. F., van Benthem, J. F., Mathématicien, I., & van Benthem, J. F. (2010). *Modal logic for open minds*. Stanford, CA: Center for the Study of Language and Information Stanford.
Van Ditmarsch, H., van Der Hoek, W., & Kooi, B. (2007). *Dynamic epistemic logic*, vol. 337. Berlin, Heidelberg: Springer Science & Business Media.
Venema, Y. (2017). *Temporal logic* (pp. 203–223), chapter 10. Hoboken, NJ: John Wiley & Sons, Ltd.
Wang, D., Wang, H., & Zou, X. (2017). Identifying key nodes in multilayer networks based on tensor decomposition. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *27*(6), 063108.
Wickham, H. (2014). *Advanced R*. Chapman & Hall/CRC The R Series. Boca Raton, FL: CRC Press.
Wickham, H. (2015). *R packages: organize, test, document, and share your code*. London, UK: O'Reilly Media, Inc.
Wu, H., Cheng, J., Huang, S., Ke, Y., Lu, Y., & Xu, Y. (2014). Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, *7*(9), 721–732.

Appendix A. Formal definitions

A.1 Basic definitions

Definition A.1. *A weighted multilayer network is a triple* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \omega \rangle$ *such that for some arbitrary set of labels* I*:*

- $\mathcal{G} := (G_\alpha)_{\alpha \in I}$ *is a family of weighted graphs* $G_\alpha := \langle V_\alpha, E_\alpha, \omega_\alpha \rangle$ *such that the respective vertex sets are pairwise distinct;*
- $\mathcal{R} := (R_{\alpha\beta})_{\alpha,\beta \in I}$ *is a family of relations, such that* $\forall \alpha, \beta \in I\, \alpha \neq \beta$ *we have* $R_{\alpha\beta} \subseteq V(G_\alpha) \times V(G_\beta)$*;*
- $\omega : E(\mathcal{M}) \to \mathbb{R}$ *assigns weights to the edges of the network, such that* $\omega(e) := \omega_\alpha(e)$ *if* $e \in E(G_\alpha)$*.*

We define $\mathcal{V}(\mathcal{G}) := \{V(G) \mid G \in \mathcal{G}\}$ *and* $V(\mathcal{G}) := \bigcup_{V \in \mathcal{V}(\mathcal{G})} V$*. As well as* $\mathcal{E}(\mathcal{G}) := \{E(G) \mid G \in \mathcal{G}\}$ *and* $E(\mathcal{G}) := \bigcup_{E \in \mathcal{E}(\mathcal{G})} E$*. Moreover, let* $E(\mathcal{R}) := \bigcup_{R \in \mathcal{R}} R$*. Lastly,* $V(\mathcal{M}) := V(\mathcal{G})$ *and* $E(\mathcal{M}) := E(R) \cup E(\mathcal{G})$*.*[7]

Definition A.2. *Let* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \equiv \rangle$ *be a multilayer network with equivalence such that* $\forall \alpha \in I \forall v, w \in V(G_\alpha)\ v \equiv w \Rightarrow v = w$*. Moreover, the equivalence classes generated by* $\equiv$ *are denoted as* $\overline{v} := \{w \mid w \in V(\mathcal{G})\ v \equiv w\}$ *and we write* $\overline{v}_\alpha := v \in V_\alpha \cap \overline{v}$*. Such a multilayer network with equivalence is called a multiplex iff* $\forall v \in V(\mathcal{G})\ |\overline{v}| = |\mathcal{G}|$*. Lastly, we write* $\mathcal{V}(\mathcal{M})$ *to reference the set of all equivalence classes*[8]*.*

Definition A.3. *Let* $\mathcal{M} := \langle \mathcal{G}, \mathcal{R}, \equiv, \preceq \rangle$ *be a multilayer network with equivalence and order. If* $\preceq$ *is a partial ordering on* $\mathcal{G}$*. Moreover,* $G_\alpha \preceq G_\beta \iff (\forall v \in G_\alpha \forall w \in G_\beta (v \equiv w \Rightarrow v \preceq w)) \wedge (v \not\equiv w \Rightarrow v \nparallel w))$*.*

A.2 Generalized flow of time

Definition A.4. *Let* $\Phi := \langle T, R, \iota \rangle$ *be a structure representing the flow of time, where* T *is a set of points,* R *is a relation over* T*, that is,* $R \subset T \times T$ *and* $\iota : T \to \wp(\mathcal{L})$ *is a function assigning each point in time a set of properties* P *described in some language* $\mathcal{L}$*. If the properties at a point in time are not relevant, we write* $\Phi := \langle T, R \rangle$*.*

Definition A.5. *Let* $\Phi := \langle T, \preceq \rangle$ *be a structure representing the flow of time, where* T *is a set of points and* $\preceq$ *is a (strict) partial order over* T *(see Venema, 2017).*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

For more on partial orders see, for example, Matthews (1994). Moreover, we write $x \prec y := x \preceq y \wedge x \neq y$ as a shorthand.

By imposing further restriction onto $\preceq$, we can develop certain notions of time, for example, a flow of time is (see Venema, 2017):

- *linear* if its underlying order is total, that is $\forall x, y \in T\, x \preceq y \vee y \preceq x$;
- *backward branching* if for a point a representing the present there are two incomparable points in the past, that is, $\exists x, y \in T\, x \preceq a \wedge y \preceq a \wedge x \nparallel y$;
- *forward branching* if for a point a representing the present, there are two incomparable points in the future, that is, $\exists x, y \in T\, a \preceq x \wedge a \preceq y \wedge x \nparallel y$;
- *backward-serial* if there is always another point in the past, that is, $\forall x \in T \exists y \in T\, y \prec x$;
- *forward-serial* if there is always another point in the future, that is, $\forall x \in T \exists y \in T\, x \prec y$.

A.3 Paths in time

Definition A.6. *Let $\Phi := \langle T, \preceq \rangle$ be a flow of time, then $t + 1 := succ(t)$ is called the (direct) successor function iff*

$$succ : T \to \wp(T) \quad succ(t) \mapsto \{t' \mid t' \in T \wedge Succ(t, t')\}$$

for the successor predicate $Succ(a, b) := a \prec b \wedge \forall x \in T\, x \preceq a \vee b \preceq x$. Moreover, $t - 1 = pred(t) := \{t' \mid t' \in T \wedge succ(t') = t\}$ is called the (direct) predecessor function

Definition A.7. *Let $\Phi := \langle T, \preceq \rangle$ be a flow of time, then*

- *a forward path in time $a \rightsquigarrow^+ b$ between a and b is a sequence $a \rightsquigarrow^+ b = t_1 \rightsquigarrow^+ t_n := t_1, t_2, \ldots, t_{n-1}, t_n$ where $\forall i \in \{1, \ldots n-1\}\, t_{i+1} \in succ(t_i)$;*
- *a backward path in time $a \rightsquigarrow^- b$ is defined in analog to $a \rightsquigarrow^+ b$ with respect to the predecessor function pred;*
- *if $a \preceq b$, a path in time is $a \rightsquigarrow b := a \rightsquigarrow^+ b$; and if $b \preceq a$, a path in time is $a \rightsquigarrow b := a \rightsquigarrow^- b$.*

The size of a path in time is determined by the elements in the path and denoted as $|a \rightsquigarrow b|$ and can be understood as the amount of steps in time between a and b in $a \rightsquigarrow b$.

Definition A.8. *Let $\Phi := \langle T, \preceq, \omega \rangle$ be a flow of time with spacing, with the function:*

$$\omega : \mathcal{S}(\Phi) \to \mathbb{R}^+,\ x \mapsto \omega(x)$$

where $\mathcal{S}(\Phi) := \{(x, y) \mid x, y \in T \wedge Succ(x, y)\}$ contains all steps in time. The length of a path in time $a \rightsquigarrow b$ with respect to ω is thus $|a \rightsquigarrow b|_\omega := \sum_{e \in \mathcal{S}(a \rightsquigarrow b)} \omega(e)$.

Intuitively, ω stretches time, that is, it assigns a duration onto a step in time. However, while the length of a path in time is a useful concept, it *cannot* serve as a measure of temporal distance.

Definition A.9. *Let $\Phi := \langle T, \preceq, \omega \rangle$ be a flow of time with spacing. We call*

$$\delta_\omega(a, b) := \begin{cases} \min\limits_{a \rightsquigarrow b} (|a \rightsquigarrow b|_\omega) & a \preceq b \\ \infty & otw. \end{cases}$$

the distance between a and b (with respect to ω). Moreover, we call δ_1, where ω is the constant function 1, the step distance between a and b.

Proposition A.10. *$\Phi := \langle T, \preceq \rangle$ and $\Phi := \langle T, \preceq, \omega \rangle$ are quasi-metric flows of time.*

Proof. Since δ_1 is a special case of δ_ω, we only consider the latter. Firstly, $\delta_\omega(x, y) \geq 0$ holds as $\forall (x, y) \in \mathcal{S}(\Phi)\, \omega(x, y) \in \mathbb{R}^+$. Secondly, $\delta_\omega(x, y) = 0 = \delta(y, x) \iff x = y$ holds because $x = y \iff |x \rightsquigarrow y| = 0$, especially with respect to ω, and the empty sum is always 0. Finally, $\delta_\omega(x, z) \leq \delta_\omega(x, y) + \delta_\omega(y, z)$. If $z \prec x$, then $\delta_\omega(x, z) = \infty$. Since there are no circles, it must be that either $\delta_\omega(x, y) = \infty$ or $\delta_\omega(y, z) = \infty$. For $y \prec x$ or $z \prec y$, we have that in either case backtracking would be required, leading to a shift in direction, which by definition is not a valid path in time. If $x \prec y \prec z$, assume the contrary. Thus, $\delta_\omega(x, z) > \delta_\omega(x, y) + \delta_\omega(y, z)$. Hence, $\exists x \rightsquigarrow y, y \rightsquigarrow z$, resulting in $x \rightsquigarrow y \rightsquigarrow z$ such that $|x \rightsquigarrow z|_\omega > |x \rightsquigarrow y \rightsquigarrow z|_\omega$. However, by definition, $|x \rightsquigarrow z|_\omega$ is minimal, thus arriving at the desired contradiction. □

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

A.4 Linear and homogeneous flows of time

Definition A.11. *Let* $\Phi := \langle T, \preceq, \omega \rangle$ *be a quasi-metric flow of time. Then* Φ *is a globally-homogeneous flow of time iff* $\forall x, y, z \in T$*:*

$$\delta_\omega(x, y) + \delta_\omega(y, z) = \infty \vee \delta_\omega(x, z) = \delta_\omega(x, y) + \delta_\omega(y, z)$$

Notice, that every linear flow of time satisfies this property. Lastly, we have to check whether global homogeneousness resolves the issue.

Proposition A.12. *Let* $\Phi := \langle T, \preceq, \omega \rangle$ *be a flow of time, then* $\forall x, y, z \in T\ (x \prec y \prec z \implies \delta_\omega(x, y) < \delta_\omega(x, z))$ *iff it is globally homogeneous*

Proof. If $\delta_\omega(x, y) + \delta_\omega(y, z) = \infty$, $\forall x \rightsquigarrow z\ (y \notin x \rightsquigarrow z)$. Hence, we only consider $x \preceq y \preceq z$. If either of $\preceq$ are equal, we are done. Otherwise, we have $x \prec y \prec z$. Since, $\delta_\omega(x, z) = \delta_\omega(x, y) + \delta_\omega(y, z)$, thus $\delta_\omega(x, z) - \delta_\omega(y, z) = \delta_\omega(x, y)$ and since $0 < \delta_\omega(y, z)$, $\delta_\omega(x, y) < \delta_\omega(x, z)$ follows □

Definition A.13. $\Phi := \langle T, \preceq, \omega \rangle$ *is a local homogeneous flow of time iff* ω *is the constant function* x *with* $x \in \mathbb{R}^+$.

Definition A.14. $\Phi := \langle T, \preceq, \omega \rangle$ *is a homogeneous flow of time iff* Φ *is locally and globally homogeneous.*

Definition A.15. *Let* $\Phi := \langle T, \preceq, \omega \rangle$ *be a globally-homogeneous flow of time, then for some point of origin* $a \in T$ *and some* $x \in \mathbb{R}$ *we have*

$$\Phi(a, x)_\omega = \begin{cases} \{y \mid y \in T \wedge \delta_\omega(a, y) = |x|\} & 0 \leq x \\ \{y \mid y \in T \wedge \delta_\omega(y, a) = |x|\} & 0 > x \end{cases}$$

Appendix B. An R-package for discrete temporal networks

The source code is publicly available at https://nm.wu.ac.at/nm/strembeck/RTempNet/tempnetwork.zip Before delving deeper into the respective classes, a precursory remark. Across most functions it is often the case that multiple parameters expect vectors of equal length as input. In such a case, the respective entries from each vector form a tuple that is further used to specify objects in the respective structures and/or to assign values to those objects. That is, for a provided family of vectors $(v_i)_{i \in \{1,\ldots,n\}} = (v_{i1}, \ldots, v_{im})_{i \in \{1,\ldots,n\}}$ all of length m, it is assumed that the entries at the j^{th} position relate, and thus could be read as the tuple $(v_{1j}, \ldots, v_{nj})$. To demonstrate the sketched behavior, consider the following example.

Example B.1. *Consider the simple, undirected graph* $G := \langle \{a, b, c\}, \{(a, b), (b, c), (c, a)\} \rangle$*. Let* f *be a function that can be used to assign numeric values to individual edges from the graph G. That is,* f *takes as input a graph, two vectors containing vertex names and a vector containing numeric values. Now consider the function call* $f(G, (a, c, b), (b, a, c), (2, 3, 4))$ *which assigns the number* 2 *to the edge* (a, b)*, the number* 3 *to the edge* (c, a)*, and the number* 4 *to the edge* (b, c).

B.1 The `tempflow`-class

The primary objective of the `tempflow` class is to provide a simple and convenient data structure to encourage the work with the concept of flows of time, as presented in Sections 2 and 3. Fundamentally, the class and its associated functions can be separated into five distinct parts: the structure itself, a set of getter functions, a set of setter function, a set of functions serving as predicates intended to check basic properties about the given `tempflow`-object, and a set of functions that allow for some basic structural modifications of the given `tempflow`-object.

`tempflow(inpGraph, setDefaultValues, safe)` As previously mentioned, the S3-class `tempflow` is merely a wrapper over the `igraph`-class. In this particular case, this can be taken literally. That is, the named list underlying the S3-class contains only a single entry. Moreover, each entry in such a list will be called *field*. This field is named `graph` and stores an `igraph`-object. Hence, the constructor of this class requires an `igraph`-object, which must be passed using the parameter `inpGraph`. However, this particular `igraph`-object must adhere to certain constraints, similar to the relation of flows of time (as discussed in this paper). Namely,

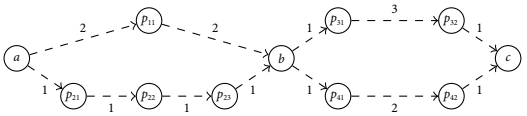
- the `igraph`-object has to be a directed, acyclic graph;[9]
- each edge in the `igraph`-object must have a weight greater or equal to 0;
- the `name` attribute of each vertex in the `igraph`-object must be unique string value.

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

Unfortunately, checking those conditions can cause a significant computational overhead. Hence, the parameter `safe` was introduced which if set to `FALSE`, forces the constructor to forgo those checks. Lastly, if the parameter `setDefaultValues` takes on the value `TRUE`, invalid weight or name assignments will be replaced by a respective default assignment. Where by default, all edges are assigned the weight 1 and the name of each vertex is a string representation of their current vertex-id (as assigned within the `igraph`-object).

Before moving on toward presenting a selection `tempflow`-functions, a small interlude. There exists one particular `tempflow`-object that is used as a running example for the rest of this section. This flow, originally taken from Example 2.3, is defined as follows.

Definition B.1. *Let* $\Phi := \langle T, \preceq, \omega \rangle$ *be the following flow of time encoded as a* `tempflow`*-object:*



`get_step_weights(tempFlow, srcPointList, dstPointList, safe)` This function serves as a representative for the set of getter functions, as most of them follow a similar structure. Firstly, the two primary parameters are `srcPointList` and `dstPointList`. Any input passed through those two parameters is only considered valid if it satisfies the following conditions:

- Either parameter must be passed a vector (or one-dimensional list) containing only strings or only integers;
- If a string vector is passed, each value must correspond with the name of a point in the provided `tempflow`-object (as provided through the parameter `tempFlow`);
- If a vector containing only integers is provided, each value must correspond with the id of a point in the provided `tempflow`-object.

Moreover, to characterize a step in time, two (not necessarily distinct) points are required. Hence, the vectors given through `srcPointList` and `dstPointList` must be of the same length in any valid input of this function. This is necessary as the steps addressed by any valid input are characterized as follows. If `srcPointList` $:= (p_i)_{0<i\leq n}$ and `dstPointList` $:= (q_i)_{0<i\leq n}$, then the steps $(p_i, q_i)_{0<i\leq n}$ are considered. Given this input pattern, it is also expected that the characterized steps exists in the provided `tempflow`-object. Fortunately, the function will ensure that those criteria are upheld, if the parameter `safe` takes on the value `TRUE`. Meaning that, if one wants to reduce computational overhead, this safety mechanism can be disabled. Given a valid input, this function will return the step weights of the characterized steps from the provided `tempflow`-object. Furthermore, if either `srcPointList` or `dstPointList` remain `NULL`, the function returns the step weights of all steps in the provided `tempflow`-object.

Example B.2. *Consider the* `tempflow`*-object* Φ *from Definition B.1 and the function call* `set_step_weights`(Φ, s, d) *where* $s := (a, p_{11}, p_{21}, p_{31}, p_{32})$ *and* $d := (p_{11}, b, p_{22}, p_{32}, c)$*. The specified function call will result in the output* $(2, 2, 1, 3, 1)$.

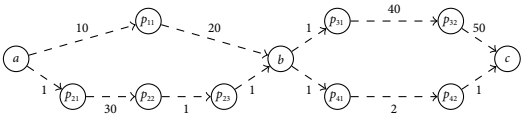
`set_step_weights(tempFlow, srcPointList, dstPointList , weightList, safe)` Due to their similar structure, this function serves as a representative for the whole setter function block. As for the required properties of the parameters `srcPointList` and `dstPointList`, the reader may be referred to the discussion of the previous function, leaving only the primary parameter `weightList` to be elaborated upon. In a similar fashion, and as implemented in the `igraph`-package, a valid value for this parameter is either a vector with the same length as the vectors provided through `srcPointList` or `dstPointList` or it is of length 1. Irrespective of the length, a valid input for the `weightList` parameter must always be comprised of nonnegative numerical values. Again, the checks ensuring the validity of the input can be disabled via the `safe` parameter. Finally, the default behavior of `srcPointList` and `dstPointList` remains equal to the `get_step_weights`-function. Additionally, if the parameter `weightList` takes on the value `NULL`, all specified steps will be assigned the value 1.[10] For an illustrative example, depicting the behavior of the function consider the following.

Example B.3. *Consider the* `tempflow`*-object* Φ *from Definition B.1. Moreover, consider the following function call:*

$$\texttt{set\_step\_weights}(\Phi, s, d, w)$$

where $s := (a, p_{11}, p_{21}, p_{31}, p_{32})$*,* $d := (p_{11}, b, p_{22}, p_{32}, c)$*, and* $w := (10, 20, 30, 40, 50)$*. The output of which is the following* `tempflow`*-object:*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



`is_branch_globally_homogeneous(tempFlow, startPoint, endPoint, considerWeights, considerLoops, safe)` Apart from a `tempflow`-object, this function takes two string or integer values representing points in the provided `tempflow`-object. Then, it reduces this `tempflow`-object to a flow containing all points in between the points specified through `startPoint` and `endPoint`, finally checking whether the resulting `tempflow`-object is globally homogeneous. Moreover, if `considerWeights` is `TRUE`, the function will account for weights when calculating the distances between points. Furthermore, if `considerLoops` is `FALSE`, all paths that contain loops will be neglected in the calculation.

Example B.4. *Consider the* `tempflow`*-object* Φ *from Definition B.1. Hence, the function call:*

$$\texttt{is\_branch\_globally\_homogeneous}(\Phi, a, b)$$

returns `TRUE`*, while*

$$\texttt{is\_branch\_globally\_homogeneous}(\Phi, a, b, \texttt{FALSE})$$

returns `FALSE`*. By contrast,*

$$\texttt{is\_branch\_globally\_homogeneous}(\Phi, b, c)$$

returns `FALSE` *and*

$$\texttt{is\_branch\_globally\_homogeneous}(\Phi, b, c, \texttt{FALSE})$$

returns `TRUE` *(see also Section 2).*

`is_branch_locally_homogeneous(tempFlow, startPoint, endPoint, safe)` Apart from a `tempflow`-object, this function takes two string or integer values representing points in the provided `tempflow`-object. Then, it reduces this `tempflow`-object to a flow only containing the points in between `startPoint` and `endPoint`, finally checking whether the resulting `tempflow`-object is locally homogeneous.

Example B.5. *Consider the* `tempflow`*-object* Φ *from Definition B.1. Hence, the function call:*

$$\texttt{is\_branch\_locally\_homogeneous}(\Phi, a, p_{23})$$

returns `TRUE`*, while*

$$\texttt{is\_branch\_locally\_homogeneous}(\Phi, a, b)$$

returns `FALSE` *(see also Section 2).*

`slice_flow(tempFlow, startPointList, endPointList, safe)` The purpose of this function is to ease the process of restricting the flow of time to relevant slices of time. For this function to behave properly, the validity conditions of the input must be ensured. That is, the parameter `startPointList` and `endPointList` must be given vectors containing either string or integer values only. If a string vector is passed, each entry must correspond with the point name of a point in the provided `tempflow`-object. Otherwise, it must correspond with the point id of a point in the `tempflow`-object. If the parameter `safe` takes on the value `TRUE`, it will be ensured that the provided input is valid. Finally, given a valid input, the `tempflow`-object will be restricted to any points that lie in between any start–end point pair. That is, given a flow in time $\Phi := \langle T, R \rangle$, the output of the function is a flow Φ', which is the flow Φ restricted to the point set P, where P is defined as:

$$P := \{p \mid \forall (s, e) \in S \times E \wedge p \in T \, s \leq p \leq e\}$$

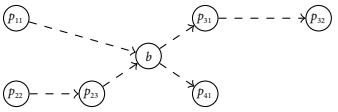
for $S :=$ `startPointList` and $E :=$ `endPointList`. To illustrate this, consider the following example.

Example B.6. *Consider the* `tempflow`*-object* Φ *from Definition B.1. Now consider the following function call:*

$$\texttt{slice\_flow}(\Phi, (p_{11}, p_{22}), (p_{32}, p_{41}))$$

The output of which is the following:

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



`compute_tempdistance(tempFlow, startPoint, endPoint, considerWeights, assumeReflexivity, safe)` This function returns the distance between two points in the `tempflow`-object. Firstly, if the parameter `safe` takes on the value `TRUE`, the validity of the input will be ensured. In this particular case, validity means that the parameter `startPoint` and `endPoint` are either strings representing the point names of existing points or integers representing the point ids of existing points. As discussed in Section 2, the distance between two points is determined by the shortest path in time. Whether the length of a path is determined by taking the sum of the weights of its steps, or merely by counting the number of its steps, depends on the parameter `considerWeights`. Furthermore, if there is no path connecting the start point and the end point, the distance is considered to be infinite. For an illustrative example, depicting the behavior of the function consider the following.

Example B.7. *Consider the* `tempflow`*-object* Φ *from Definition B.1 and the function call:*

$$\texttt{compute\_tempdistance}(\Phi, a, c, \texttt{TRUE}),$$

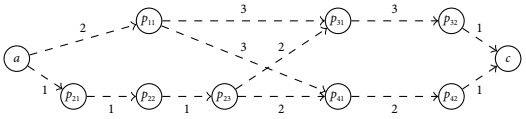
which produces the output 8*. Moreover, the function call:*

$$\texttt{compute\_tempdistance}(\Phi, a, c, \texttt{FALSE})$$

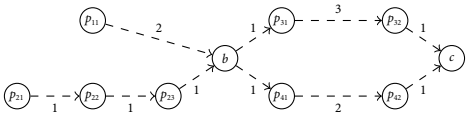
produces the output 5*.*

`remove_point(tempFlow, removePoint, safe)` This function allows for the removal of a point from the provided `tempflow`-object, while ensuring that the `tempflow`-object remains connected.[11] Again the validity of its input, that is, the value of `removePoint` is either a string or an integer corresponding to the point name or point id of a point in the `tempflow`-object, will be checked if the parameter `safe` is set to `TRUE`. In an initial step, the point will be removed from the given `tempflow`-object. The main purpose of this function, however, is not the removal of points, but the reconnecting of its predecessor and successors. This occurs in the subsequent step and is accomplished by connecting all predecessors of the removed point to its successors while at the same time aggregating the weights of the corresponding steps through addition. That is, let $t :=$ `removePoint` and let $pred(t)$ be the set of t's predecessors and $succ(t)$ the set of its successors and let ω be a weight function reflecting the weight attributes of the `tempflow`-object, then the set of newly added steps is going to be $pred(t) \times succ(t)$ with weight attributes being assigned as follows $\forall (a, b) \in pred(t) \times succ(t)\ \omega((a, t)) + \omega((t, b))$. Lastly, in the case of either $pred(t)$ or $succ(t)$ being empty, this function behaves as virtually identical to the function `delete_points`.

Example B.8. *Consider the* `tempflow`*-object* Φ *from Definition B.1 and the function call* `remove_point`(Φ, b)*, which produces the following output:*



By contrast, the function call `remove_point`(Φ, a) *produces the following as output:*



`insert_point(tempFlow, newPoint, predPointList, sucPointList, weightList, pointAttrList, stepAttrList, removeEdges, safe)` This function allows for the insertion of a point in between two sets of points.[12] Starting with the parameter `newPoint`, the value passed to this parameter will become the name of the newly inserted point. Hence, it must be unique and a string. As for the parameter `predPointList` and `sucPointList`, the usual conditions hold. That is, they must contain string or integer values corresponding with the point names or point ids of points in the `tempflow`-object. The parameter `weightList`, which provides the possibility of determining the weights of the newly inserted steps, has familiar validity conditions. Namely, it can only take a vector that contains nonnegative numeric values and is either of length 1 or is equal to the sum of the lengths of the vectors passed through `predPointList` and `sucPointList`. Similar length restrictions are also enforced with respect to vectors contained in the parameter `stepAttrList`, which should be

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

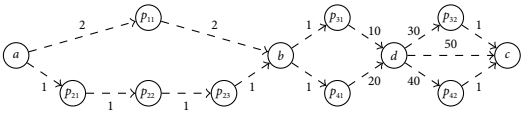
structured as required by the function `igraph::add.edges`. By contrast, while similarly expecting a structure as required by the function `igraph::add.vertices`, the vectors passed through the parameter `pointAttrList` should only be of length 1. Finally, all those conditions will be checked as long as `safe` is `TRUE`.

As implied by its name, this function inserts a point, named after the value passed through `newPoint`, such that it predecessors will be the points specified by the parameter `predPointList` and such that is successors are those specified in `sucPointList`. Furthermore, if the parameter `removeEdges` is set to `TRUE`, any step intersecting with the set `predPointList` × `sucPointList` will be removed. Moreover, the weights are assigned as follows. Let $t :=$ `newPoint`, let $P :=$ `predPointList` $= (p_i)_{i\in\{1,\ldots,n\}}$, let $S :=$ `sucPointList` $= (s_i)_{i\in\{1,\ldots,m\}}$, and let $W :=$ `weightList` $= (w_i)_{i\in\{1,\ldots,n+m\}}$. After executing the function, the resulting `tempflow`-object will contain the step (p_i, t) with the weight w_i for some $i \in \{1, \ldots, n\}$, as well as the step (t, s_j) with the weight w_{n+j} for some $j \in \{1, \ldots, m\}$. The behavior of this function can be observed in the following example.

Example B.9. *Consider the* `tempflow`*-object* Φ *from Definition B.1 and the function call:*

$$\texttt{insert\_point}(\Phi, d, (p_{31}, p_{41}), (p_{32}, p_{42}, c), \texttt{weightList} = (10, 20, 30, 40, 50)),$$

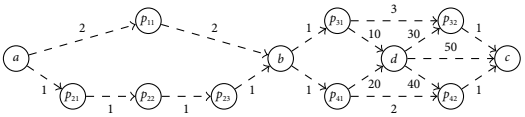
which produces the following as output:



By contrast, the function call:

$$\texttt{insert\_point}(\Phi, d, (p_{31}, p_{41}), (p_{32}, p_{42}, c), \texttt{weightList} = (10, 20, 30, 40, 50), \texttt{removeEdges} = \texttt{FALSE}),$$

produces the following output:



B.2 The `tempnetwork`-class

This class builds upon the `tempflow`-class. While the `tempflow`-class'es primary objective was to provide a data structure for encoding flows of time, the `tempnetwork`-class intends to do the same for the temporal networks presented in this paper. Not only does it support the same (or similar) operations as the `tempflow`-class, but its functions can again be divided into a set of getter functions, a set of setter functions and a set of functions that allow for some basic structural modifications of the given `tempnetwork`-object.

`tempnetwork(tempFlow, graphList, equivalenceRelation, setDefaultGraphNames, setDefaultFlowValues, storeTempGraph,safe)` This function returns an S3-class `tempnetwork`-object, containing five fields, that is, entries in the named list underlying the S3-class:

- The field `tempFlow`, accessible through the function `tF`, stores a non-empty `tempflow`-object;
- The field `graphList`, accessible through the function `gL`, stores a non-empty, named list of `igraph`-objects, where
  - the indices are identical to the names of the points in the provided `tempflow`-object;
  - every vertex must have a string as a name (which must be unique with respect to its own graph);
  - all vertex sets have the same size;
  - all vertex sets must be identical, if no equivalence relation is provided.
- The field `equivRel`, accessible through the function `eR`, stores a named list of named lists, which should approximate an equivalence relation composed out of equivalence classes, where
  - the indices of the whole equivalence relation must be unique;

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

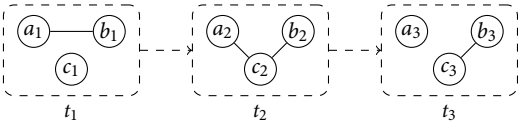
  - the number of equivalence classes must be equal to the number of vertices in any graph of the provided graph list;
  - the indices of each equivalence class are identical to the names of the points from the provided `tempflow`-object;
  - for every vertex v in the graph at point t from the given graph list, there must exist an equivalence class in the equivalence relation that has the value v at index t;
  - for every point t in the given `tempflow`-object, if one aggregates all entries indexed by point t over all equivalence classes, one obtains the vertex set of the graph at point t;
  - for every point t in the given `tempflow`-object, there can not be two different equivalence classes that have the same entry at point t.
- The field `tempGraph`, accessible through the function `tG`, stores a graph encoding of the `tempnetwork`-object as an `igraph`-object. However, this graph is computed lazily, that is, it will be computed only, if one explicitly calls for it.
- The field `storeTempGraph`, accessible through the function `stG`, is used to specify whether the graph encoding of the `tempnetwork`-object is stored in the `tempGraph`-parameter after it is computed. However, the moment some function changes the underlying `tempnetwork`-object the graph is discarded, even if this flag is set to `TRUE`.

Most of the fields correspond to an input parameter of the constructor. However, some small differences exist. Firstly, it is possible to pass an `igraph`-object through the `tempFlow`-parameter, as long as it satisfies all relevant conditions for the construction of a `tempflow`-object. That is, the required `tempflow`-object will be constructed from the given `igraph`-object, which is precisely the reason why the parameter `setDefaultFlowValues` exits. Secondly, if the parameter `equivalenceRelation` remains `NULL`, it will be assumed that no equivalence relation was given, thereby requiring from the graph list that all vertex sets must be identical. Thirdly, if the parameter `setDefaultGraphNames` is `TRUE`, any graph in the given graph list containing a vertex without a name will be replaced by an isomorphic graph where the name of each vertex is a string corresponding to its index in the respective `igraph`-object. Moreover, any graph containing a vertex with a name that is not a string will be coerced into a string. Lastly, the parameter `safe` can be used to enable or disable the checks that ensure that all input conditions are satisfied.

The following `tempnetwork`-object will be used as a running example throughout the remainder of this section.

Definition B.2. *Consider the following* `tempnetwork`*-object* $\mathcal{T}$ *where* $\texttt{tF}(\mathcal{T})$ *is*

$\texttt{gL}(\mathcal{T})$ *(already aligned in accordance with* $\texttt{tF}(\mathcal{T})$*) is*



and where $\texttt{eR}(\mathcal{T})$ *is*

$$\begin{aligned} &\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3\},\\ &\ b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3\},\\ &\ c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\} \end{aligned}$$

`insert_point(tempNetwork, newPoint, predPointList, sucPointList, graphList, equivRelation, weightList, pointAttrList, stepAttrList, removeEdges, safe)` This function is an example of a function that is already defined for a `tempflow`-object but requires a slight alteration in order to remain viable for `tempnetwork`-objects. That is, its purpose is identical to its `tempflow` counterpart. However, rather that returning a `tempflow`-object, this function returns a `tempnetwork`-object. Furthermore, since this function adds a point to the `tempflow`-object of the given `tempnetwork`-object, both the graph list and the equivalence relation must be updated as well, hence, requiring one to provide an `igraph`-object for the newly added point. Moreover, each equivalence class must be extended by the vertices of the newly added graph.

To be more precise, consider the following. Firstly, the parameter `graphList` requires either a single `igraph`-object or a named list containing a single `igraph`-object indexed by the name of the new point. Secondly, the parameter `equivRelation` requires a named list of named lists:

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

- where the outer-level names must be identical to the outer-level names of the already existing equivalence relation;
- where the inner-level names must equal the name of the newly added point;
- where the values must correspond to the vertex names of the newly added graph.

Moreover, if both the existing equivalence class and the parameter `equivRelation` are `NULL`, then it is required that the names of the vertices must be identical across all graphs (including the given one). Otherwise, the given equivalence relation will be merged with the already existing one (which may or may not have to be constructed). Using those two parameters, the existing graph list and equivalence classes are extended. Consider the following as an example.

Example B.10. *Consider some* `tempnetwork`*-object* $\mathcal{T}$ *such that*

$$\mathtt{gL}(\mathcal{T}) = \{t_1 \mapsto G_1,\ t_2 \mapsto G_2,\ t_3 \mapsto G_3\}$$

and

$$\begin{aligned}\mathtt{eR}(\mathcal{T}) = \{&a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3\},\\ &b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3\},\\ &c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\}\end{aligned}$$

Moreover, assume the new point to be t_4*, the new graph to be* $G_4 := \langle\{a_4, b_4, c_4\}, E\rangle$ *and the provided equivalence class fragment to be* $\{a \mapsto \{t_4 \mapsto a_4\}, b \mapsto \{t_4 \mapsto b_4\}, c \mapsto \{t_4 \mapsto c_4\}\}$*. Resulting in the new graph list* $\{t_1 \mapsto G_1,\ t_2 \mapsto G_2,\ t_3 \mapsto G_3\ t_4 \mapsto G_4\}$ *and the new equivalence relation:*

$$\begin{aligned}&\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3, t_4 \mapsto a_4\},\\ &b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3, t_4 \mapsto b_4\},\\ &c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3, t_4 \mapsto c_4\}\}\end{aligned}$$

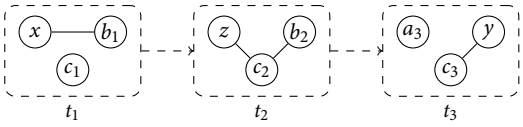
after the function call (see also Section 2).

`set_vertex_names(tempNetwork, pointNameList, tVertexNameList, newVertexNameList, changeEquivRel, safe)` This function changes the names of vertices in a graph from the graph list of the given `tempnetwork`-object. Apart from the `tempnetwork`-object, this function has the parameter `pointNameList`, which takes a vector containing strings that represent points in the `tempflow`-object of the given `tempnetwork`-object. Moreover, the vector passed to `tVertexNameList` contains strings representing the names of the equivalence classes. Those two input vectors are sufficient to identify the graph and the name of the vertex that ought to be changed. Hence, the last major parameter is `newVertexNameList`, which takes a vector containing the new names for each of the specified vertices. If `NULL`, a vector containing the string encoding of the indices of the specified vertices in their respective `igraph`-objects will be computed. Lastly, the parameter `changeEquivRel` specifies whether the names of the given vertices should also be changed in the equivalence relation of the `tempnetwork`-object.

Example B.11. *Consider* $\mathcal{T}$ *to be the* `tempnetwork`*-object from Definition B.2. Now consider the following function call:*

$$\mathcal{T}' := \mathtt{set\_vertex\_names}(\mathcal{T}, (t_1, t_3, t_2), (a, b, a), (x, y, z))$$

resulting in $\mathtt{gL}(\mathcal{T}')$



and where $\mathtt{eR}(\mathcal{T}')$ *is*

$$\begin{aligned}&\{a \mapsto \{t_1 \mapsto x,\ t_2 \mapsto z,\ t_3 \mapsto a_3\},\\ &b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto y\},\\ &c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\}\end{aligned}$$

`add_tvertices(tempNetwork, tVertexNameList, gVertexNameList, pointNameList, safe)` This function adds a series of vertices to every graph in the graph list of the given `tempnetwork`-object and creates corresponding

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

equivalence classes and adds them to the equivalence relation of the given `tempnetwork`-object. To accomplish this, the following parameters are required.

Firstly, apart from the obligatory `tempnetwork`-object passed through the parameter `tempNetwork`, this function has the parameter `tVertexNameList`, which takes a vector of strings representing the names of the new equivalence classes. It must be either of the same length as the vector `gVertexNameList` or its length must be the length of the vector `gVertexNameList` divided by n, where n is the number of points in the `tempflow`-object of the given `tempnetwork`-object. If the latter is the case, it will be assumed that the first entry of this vector corresponds to the first n entries in the `gVertexNameList`-vector.

Secondly, the parameter `gVertexNameList` takes a vector of strings representing the vertices to be added to the graphs in `gL(tempNetwork)`. As every graph must have the same number of vertices, the length of this vector must be a multiple of the number of points in `tF(tempNetwork)`. By default, it is assumed that those have the same name as the equivalence classes specified in the `tVertexNameList`-vector.

Thirdly, the parameter `pointNameList` takes a vector of strings representing the names of the points in `tF(tempNetwork)`. The length of this vector must be identical to the length of the `gVertexNameList`-vector, as this vector is used to place the new vertex in the appropriate graph. By default, the points as ordered by the underlying `igraph`-object are selected and replicated until the resulting vector has the same length as the vector `gVertexNameList`.

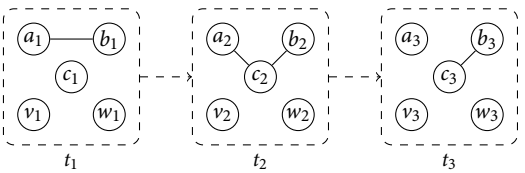
Example B.12. *Consider $\mathcal{T}$ to be the* `tempnetwork`*-object from Definition B.2. Now consider the following function call:*

$$\begin{aligned}\mathcal{T}' := \texttt{add\_tvertices}(&\mathcal{T}, (v, w),\\ &(v_1, v_2, v_3, w_3, w_1, w_2),\\ &(t_1, t_2, t_3, t_3, t_1, t_2))\end{aligned}$$

which is equivalent to

$$\begin{aligned}\mathcal{T}' := \texttt{add\_tvertices}(&\mathcal{T},\\ &(v, v, v, w, w, w),\\ &(v_1, v_2, v_3, w_3, w_1, w_2),\\ &(t_1, t_2, t_3, t_3, t_1, t_2))\end{aligned}$$

resulting in $\texttt{gL}(\mathcal{T}')$*:*



and where $\texttt{eR}(\mathcal{T}')$ *is*

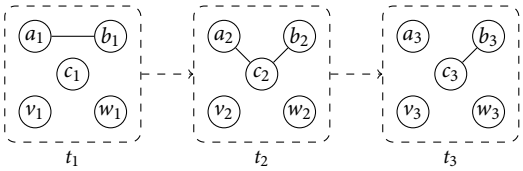
$$\begin{aligned}&\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3\},\\ &\ b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3\},\\ &\ c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\\ &\ v \mapsto \{t_1 \mapsto v_1,\ t_2 \mapsto v_2,\ t_3 \mapsto v_3\}\\ &\ w \mapsto \{t_1 \mapsto w_1,\ t_2 \mapsto w_2,\ t_3 \mapsto w_3\}\}\end{aligned}$$

`remove_tvertices(tempNetwork, tVertexNameList, safe)` This function takes a list of equivalence class names and creates a new `tempnetwork`-object by removing the specified equivalence classes from the equivalence relation of the given `tempnetwork`-object and by removing all vertices contained in in the specified equivalence classes from their respective graphs in the graph list of the given `tempnetwork`-object.

Apart from the parameter `tempNetwork`, which takes a `tempnetwork`-object, this function requires a vector of strings representing the names of equivalence classes which is passed through the parameter `tVertexNameList`.

Example B.13. *Consider the following* `tempnetwork`*-object $\mathcal{T}$ where* $\texttt{gL}(\mathcal{T})$ *(already aligned in accordance with* $\texttt{tF}(\mathcal{T})$ *) is*

https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press



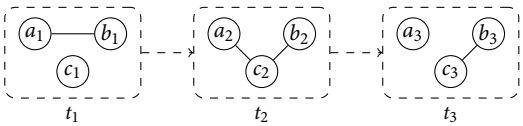
and where $\texttt{eR}(\mathcal{T})$ *is*

$$\begin{aligned}&\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3\},\\ &\ b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3\},\\ &\ c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\\ &\ v \mapsto \{t_1 \mapsto v_1,\ t_2 \mapsto v_2,\ t_3 \mapsto v_3\}\}\\ &\ w \mapsto \{t_1 \mapsto w_1,\ t_2 \mapsto w_2,\ t_3 \mapsto w_3\}\}\end{aligned}$$

Now consider the following function call:

$$\mathcal{T}' := \texttt{remove\_tvertices}(\mathcal{T}, (v, w))$$

where $\mathcal{T}'$ *is identical to the* `tempnetwork`*-object* $\mathcal{T}$*, as defined in Definition B.2:*



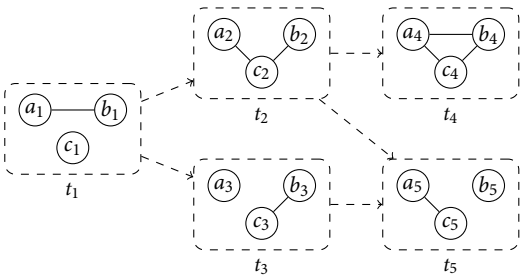
and where $\texttt{eR}(\mathcal{T}')$ *is*

$$\begin{aligned}&\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3\},\\ &\ b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3\},\\ &\ c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3\}\}\end{aligned}$$

`construct_underlying_graph(tempNetwork, connectGraphs, forceIGraph)` This function constructs a graph representation of the given `tempnetwork`-object. This is accomplished by renaming all vertices, where the new name for each vertex is constructed by concatenating the name of the point, the name of the equivalence class, and the name of the vertex into a single string. Additionally, the vertex attributes `vertexName` and `pointNametempVertexName` are added to each vertex and filled according to its indices in the equivalence relation. After this, all edited graphs are merged (using `igraph::disjoint_union`).

Moreover, if `connectGraphs` is `TRUE`, for each equivalence class, all contained vertices are connected in the exact same manner as the points in the `tempflow`-object of the given `tempnetwork`-object. Furthermore, if the field `storeTempGraph` in the given `tempnetwork` is `TRUE` and the parameter `forceIGraph` is `FALSE`, the function will return a new `tempnetwork`-object, where the computed graph is stored in the field `tempGraph`. Otherwise, the computed `igraph`-object will be returned directly.

Example B.14. *Consider the following* `tempnetwork`*-object* $\mathcal{T}$ *where* $\texttt{gL}(\mathcal{T})$ *(already aligned in accordance with* $\texttt{tF}(\mathcal{T})$ *) is*



https://doi.org/10.1017/nws.2021.20 Published online by Cambridge University Press

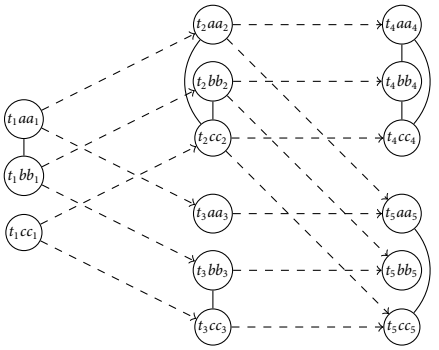
and where $\texttt{eR}(\mathcal{T})$ *is*

$$\{a \mapsto \{t_1 \mapsto a_1,\ t_2 \mapsto a_2,\ t_3 \mapsto a_3,\ t_4 \mapsto a_4,\ t_5 \mapsto a_5\},$$
$$b \mapsto \{t_1 \mapsto b_1,\ t_2 \mapsto b_2,\ t_3 \mapsto b_3,\ t_4 \mapsto b_4,\ t_5 \mapsto b_5\},$$
$$c \mapsto \{t_1 \mapsto c_1,\ t_2 \mapsto c_2,\ t_3 \mapsto c_3,\ t_4 \mapsto c_4,\ t_5 \mapsto c_5\}\}$$

Now consider the following function call:

$$\texttt{construct\_underlying\_graph}(\mathcal{T}, \texttt{forceIGraph} = \texttt{TRUE})$$

resulting in



with the following vertex attributes (here only a small sample is depicted):

$\{t_1aa_1 \mapsto \{vertexName \mapsto a_1,$
$\quad pointName \mapsto t_1,$
$\quad tempVertexName \mapsto a\},$
$t_2bb_2 \mapsto \{vertexName \mapsto b_2,$
$\quad pointName \mapsto t_2,$
$\quad tempVertexName \mapsto b\}$
$t_5cc_5 \mapsto \{vertexName \mapsto c_5,$
$\quad pointName \mapsto t_5,$
$\quad tempVertexName \mapsto c\}\}.$

`compute_distance(tempNetwork, startTVertex, endTVertex, startPoint, endPoint, considerWeights, cutGraph, safe)` This function computes the multi-valued distance (or distances), see Section 3.1, between two vertices in the graph constructed by `construct_underlying_graph`. To address those vertices, one is required to provide the name of their equivalence classes, as well as their associated points in the given `tempflow`-object. If it is the case that the start and end vertex are both completely specified, then this function will return a vector with its first entry representing the distance traveled on inter-layer edges and its second representing the distance traveled on intra-layer edges.

If a vertex is not completely specified, that is, `startPoint` or `endPoint` is not given, then this function computes the multi-valued distances with respect to every vertex in the given equivalence class. That is, the function assumes that `startPoint == NULL` implies `startPoint == tP(tempNetwork)`, analog for `endPoint`. Given this assumption, it will call itself recursively compute the base case for each of those entries and aggregate the results into a list.

Example B.15. *Consider Example 3.2. The following equivalences hold*

$$\texttt{compute\_distance}(\mathcal{T}, a, b) = |\overline{a} \rightsquigarrow \overline{b}|$$
$$\texttt{compute\_distance}(\mathcal{T}, a, b, t_1) = |\overline{a}_{t_1} \rightsquigarrow \overline{b}|$$
$$\texttt{compute\_distance}(\mathcal{T}, a, b, \texttt{endPoint=}t_3) = |\overline{a} \rightsquigarrow \overline{b}_{t_3}|$$
$$\texttt{compute\_distance}(\mathcal{T}, a, b, t_1, t_3) = |\overline{a}_{t_1} \rightsquigarrow \overline{b}_{t_3}|$$

Cite this article: Kueffner K. and Strembeck M. (2022). Toward a Generalized Notion of Discrete Time for Modeling Temporal Networks. *Network Science* **9**, 443–477. https://doi.org/10.1017/nws.2021.20