Bentert *et al. Applied Network Science* (2020) 5:73
https://doi.org/10.1007/s41109-020-00311-0

Applied Network Science

RESEARCH Open Access

Efficient computation of optimal temporal walks under waiting-time constraints

Matthias Bentert*, Anne-Sophie Himmel, André Nichterlein and Rolf Niedermeier

An extended abstract of this work appeared in the proceedings of the 8th International Conference on Complex Networks and their Applications (2019) (Himmel et al. 2019). This full version considers more optimization criteria, provides missing proofs, and presents extended experimental results.

*Correspondence:
matthias.bentert@tu-berlin.de
Anne-Sophie Himmel is supported by the DFG, project FPTinP (NI 369/16).
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Berlin, Germany

Abstract

Node connectivity plays a central role in temporal network analysis. We provide a broad study of various concepts of walks in temporal graphs, that is, graphs with fixed vertex sets but arc sets changing over time. Taking into account the temporal aspect leads to a rich set of optimization criteria for "shortest" walks. Extending and broadening state-of-the-art work of Wu et al. [IEEE TKDE 2016], we provide an algorithm for computing shortest walks that is capable to deal with various optimization criteria and any linear combination of these. It runs in $O(|V| + |E| \log |E|)$ time where $|V|$ is the number of vertices and $|E|$ is the number of time-arcs. A central distinguishing factor to Wu et al.'s work is that our model allows to, motivated by real-world applications, respect waiting-time constraints for vertices, that is, the minimum and maximum waiting time allowed in intermediate vertices of a walk. Moreover, other than Wu et al. our algorithm also allows to search for walks that pass multiple subsequent time-arcs in one time step, and it can deal with a richer set of optimization criteria. Our experimental studies indicate that our richer modeling can be achieved without significantly worsening the running time when compared to Wu et al.'s algorithms.

Keywords: Temporal networks, Temporal paths, Shortest path computation, Waiting policies, Infectious disease spreading

Introduction

Computing shortest paths in networks is arguably among the most important graph algorithms. It is relevant in numerous application contexts and used as a subroutine in a highly diverse set of applications. While the respective problem has been studied for static graphs for decades, over the last years there has been an intensified interest in studying shortest path computations in *temporal graphs*—graphs where the vertex set remains static, but the arc set may change over time (in discrete time steps).

Two natural motivating examples for the relevance of path and walk (in the later one may visit a vertex multiple times) computations in temporal graphs are as follows. First, Wu et al. (2016) discussed applications in flight networks where every node represents an airport and each arc represents a flight connection with a departure time. Clearly, a "shortest" path may then relates to a most convenient flight connection between two cities. Second, understanding the spread of infectious diseases is a major challenge to global health. Herein, nodes represent persons and time-labeled arcs represent contacts between persons where a virus can be transmitted. "Shortest" path (walk) analysis here may help to find measures against disease spreading (Newman 2018, Chapter 17). Notably, in both examples one might need to also take into account issues such as different concepts of "shortest"—also called optimal—paths (walks) or waiting times in nodes; this will be an important aspect of our modeling.

Our main reference point is the work of Wu et al. (2016) on efficient algorithms for optimal *temporal path* computation. These algorithms are also implemented in the temporal graph library of Apache Flink (Lightenberg et al. 2018). We extend their model with respect to two aspects. First, we additionally consider waiting-time constraints[1] for the network nodes, that is, a minimum time a walk must spend in a node before continuing and a maximum time a walk can stay in a node before it has to continue. Minimum waiting times can be used to model that passengers on an airport need some time to get from an arriving flight to the terminal for their next flight and maximum waiting times can be used to model recover times of patients after which they can no longer infect other people. Importantly, maximum-waiting-time constraints can enforce cycles from one node to another; refer to Fig. 1 for a simple example. Hence, we need to take into account optimal *temporal walks* from one node to another (in Wu et al.'s model without waiting-time constraints there is always an (optimal) temporal path visiting no node twice because no cycles are necessary).

Actually, if one insists on paths (without repeated nodes) instead of walks, then even deciding whether there exists a path between two nodes becomes NP-hard (Casteigts et al. 2019).

The second extension to Wu et al.'s work lies in an increased number of optimality criteria (different notions of optimal walks) and the fact that we do not only deal with optimizing one criterion but a linear combination of any of these, thus addressing potentially richer modeling needs in real-world applications. For example, in the above-mentioned flight network a traveling person might want to use our algorithm to optimize a linear combination of cost and travel duration. Note that trying to find walks under constraints (e.g., travel duration at most t *and* cost at most c for given $c, t \in \mathbb{N}$) leads to NP-hard computational problems, even in the static case (Ahuja et al. 1993).

Related Work. The theory of temporal graphs is a relatively young but active field of research (Holme 2015; Holme and Saramäki 2012; Holme and Saramäki 2013; Holme and Saramäki 2019). Many of the basic concepts of temporal graphs such as temporal connectivity (Kempe et al. 2002; Nicosia et al. 2012; Axiotis and Fotakis 2016; Mertzios et al. 2019) or s-z-separation (Zschoche et al. 2020; Fluschnik et al. 2020) are based on the notion of temporal paths and walks. The concept of optimal temporal walks plays

[1] Waiting-time constraints play a particularly important role in standard spreading models of infectious diseases such as the SIS-model (Susceptible-Infected-Susceptible) (Newman 2018, Chapter 17).

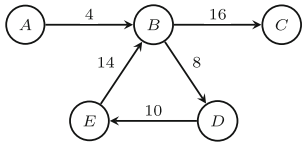


Fig. 1 A temporal graph (with time-labeled arcs called time-arcs) with maximum waiting time four in which the only temporal walk from *A* to *C* visits *B* twice

a central role in the definition of temporal graph metrics such as eccentricity, diameter, betweenness and closeness centrality (Pan and Saramäki 2011; Santoro et al. 2011; Kim and Anderson 2012; Buß et al. 2020).

An early algorithm for computing optimal temporal walks is due to Xuan et al. (2003). They computed temporal walks under different optimization criteria, namely *foremost*, *fastest*, and *minimum hop-count*. Wu et al. (2016) followed up by introducing algorithms for computing optimal walks for the optimization criteria *foremost*, *reverse-foremost*, *fastest*, and *shortest* on temporal graphs with no waiting-time constraints. Their algorithms run in linear and quasi-linear time with respect to the number of time-arcs, provided that transmission times on time-arcs are greater than zero. Concerning time-dependent multicriteria optimal path computation, there has been research in the related field of route planning (Bast et al. 2016).

The study of minimum- and maximum-waiting-time constraints in vertices has not received much attention in the context of temporal walks even though they are considered as important extensions to the temporal walk model (Holme and Saramäki 2012; Pan and Saramäki 2011). One of the first papers to consider waiting-time constraints in the context of paths is due to Dean (Dean 2004). Dean showed that time-dependent shortest path problems with waiting constraints can be solved in polynomial time. Dean's setting is closely related to waiting-time constraints for optimal temporal walks. In more recent work, Modiri et al. (2019) and Kivelä et al. (2018) studied the changes in reachability when introducing maximum-waiting time constraints for temporal walks using so-called *event graphs*. Lastly, Casteigts et al. (2019) have very recently shown that finding optimal temporal paths under maximum-waiting-time constraints is NP-hard.

In the so-called multistage setting, where multiple graphs over the same vertex set are considered without a (temporal) ordering, paths have been studied by Fluschnik et al. (2020).

Our Contributions. We analyze the running time complexity of computing optimal temporal walks under waiting-time constraints. To this end, we develop and (theoretically and empirically) analyze an algorithm for finding an optimal walk from a source vertex to each vertex in the temporal graph under waiting-time constraints. Our algorithm runs in quasi-linear time in the number of time-arcs. This implies that the additional consideration of waiting-time constraints on temporal walks does not increase the asymptotic computational complexity of finding optimal temporal walks. Moreover, our algorithm can compute optimal walks not only for single optimality criteria but also for any linear combination of these. In experiments on real-world social network data sets, we demonstrate that in terms of efficiency our algorithm can compete with state-of-the-art algorithms by Wu et al. (2016). Their algorithms only run on temporal graphs without

waiting-time constraints, which do optimize only one criterion (and not a linear combination). Additionally, our algorithm allows transmission times being zero (hence, allowing to pass multiple arcs in one time step) while the algorithms of Wu et al. request transmission times on arcs to be greater than zero.

Organization of the Paper. In "Modeling of optimal temporal walks" section, we discuss temporal graphs, temporal walks, and the various corresponding optimality criteria, starting with an extensive motivating example in the context of disease spreading. In "Formal definitions" section, we introduce definitions and notations used throughout the paper. We continue in "Transformation" section by presenting two simple linear-time transformations to eliminate transmission times and minimum waiting times in the temporal graph without loosing any modeling power. In "Algorithm" section, we design and analyze an algorithm for computing optimal walks under maximum-waiting-time for any linear combination of these. Finally, in "Experimental results" section, we demonstrate the efficiency of our algorithm on real-world data sets. We compare our running times with the running times of the algorithms of Wu et al. (2016). We further examine the impact of different maximum-waiting-time values on the existence and structure of optimal temporal walks.

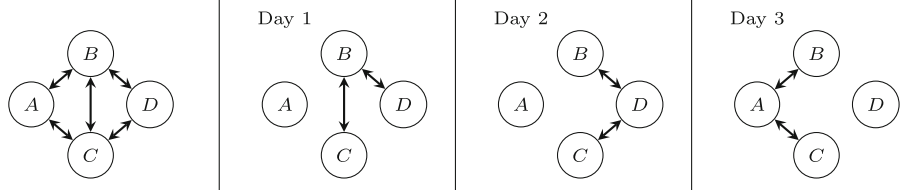
Modeling of optimal temporal walks

Before we introduce our basic concepts relating to temporal graphs and walks, we start with a more extensive discussion of a motivating example from the disease spreading context.

Disease Spreading Motivating Example. Pandemic spread of an infectious disease is a great threat to global health, potentially associated with high mortality rates as well as economic fallout (Salathé et al. 2010). Understanding the dynamics of infectious disease spreading within human proximity networks could facilitate the development of mitigation strategies.

A large part of the study required to understand the dynamics of infectious diseases is the analysis of transmission routes through proximity networks (Salathé et al. 2010). Classical graph theory can be used to model the main structure of a network: Each person in the network is represented by a node and an arc between two nodes indicates the possibility of an infection from one person to the other. However, the time component plays a crucial role in the analysis of transmission routes of a potential disease as shown in the following example:

Example 1 *Studying a proximity network as shown in Fig. 2a, there are several transmission routes from A to D, e.g.,* $A \to B \to D$ *and* $A \to C \to D$*, by which a disease could have spread. If we extend our model by the points in time of proximity contacts in Fig. 2b to d, then we reach the conclusion that a disease could not have spread from A to D. The proximity contacts* $A \xrightarrow{3} B$ *and* $A \xrightarrow{3} C$ *occurred on day three whereas the contacts* $B \xrightarrow{1} D$ *and* $C \xrightarrow{2} D$ *occurred on days one and two, respectively. Thus, A could have only infected B and C after proximity contact with D.* ■



(a) Proximity network (b) Proximity network at day 1 (c) Proximity network at day 2 (d) Proximity network at day 3

Fig. 2 A proximity network modeled as a static graph (**a**) and a closer look at the days in which the proximity contacts occur (**b**-**d**)

In addition to what has been said so far, the infectious period of a disease also has to be taken into account when computing potential transmission routes through the network, implying the minimum time a person has to be infected before she becomes contagious herself and the maximum time a person can be infected before she is no longer contagious:

Example 2 *If person B was infected by person A on day four ($A \xrightarrow{4} B$) and the infectious period of the disease starts after one day and ends after the fourth day, then person B could not have infected person C she met on day ten ($B \xrightarrow{10} C$). Hence, person C could not have been infected by the disease via the transmission route $A \xrightarrow{4} B \xrightarrow{10} C$.* ■

Temporal Graphs. These are capable of modeling both properties elaborated in the two examples above. Temporal graphs are already a frequently used model in the prediction and control of infectious diseases (Masuda and Holme 2013; Holme 2016). Temporal graphs—also referred to as temporal networks (Nicosia et al. 2013; Holme and Saramäki 2012), evolving graphs (Xuan et al. 2003), or time-varying graphs (Santoro et al. 2011; Casteigts et al. 2012)—are graphs where the arc set changes over time; thus, they can capture the dynamics within a proximity network.

In this paper, we will consider the following temporal graph model: A temporal graph consists of a *set of vertices*, a *set of time-arcs*, and a *lifetime*. A time-arc is a (directed) arc between two vertices that is associated with a *time stamp* at which the contact occurs and a *transmission time* that indicates the amount of time to traverse the arc. Furthermore, each vertex v is assigned an individual *minimum waiting time* $\alpha(v)$ and *maximum waiting time* $\beta(v)$.

The application areas of temporal graphs are numerous: In addition to human and animal proximity networks, they are used in communication networks, traffic networks, and distributed computing, to name a few application areas (Holme and Saramäki 2012; Holme and Saramäki 2013; Holme and Saramäki 2019).

In our running disease spreading example, we are interested in transmission routes of an infectious disease. These transmission routes can revisit a person in the proximity network due to possible reinfection (Barabási 2016). Hence, the transmission routes can contain cycles which needs to be considered in the choice of concepts representing these routes.

Temporal Walks & Optimal Temporal Walks. Within the temporal graph model, temporal walks—also called journeys (Xuan et al. 2003; Nicosia et al. 2013)—are the fundamental concept that represents the transmission routes in our running example.

A temporal walk is a sequence of time-arcs which connects a sequence of vertices and which are non-decreasing in time. In our model, a temporal walk additionally ensures that it remains the minimum waiting time in each intermediate vertex and does not exceed the maximum waiting time in any intermediate vertex of the walk.

Example 3 *Continuing Example 2, a valid temporal walk (transmission route) from A to D could be the following:* $A \xrightarrow{4} B \xrightarrow{8} D$. *Person A could have infected B on day four. Due to the infectious period of four days, B was still contagious on day eight when she had contact with person C. This does not hold on a route* $A \xrightarrow{4} B \xrightarrow{10} C$ *as discussed in Example 2. If B was infected at time step 4, then she was not contagious anymore at time step 10.* ■

A *temporal path* is a temporal walk where all vertices are pairwise distinct. Maximum-waiting-time constraints have significant impact on temporal walks. In a temporal graph with constraints on the maximum waiting time, one can be forced to make detours because the maximum waiting time in a vertex is exceeded. As a consequence, there can be two vertices A and C such that any temporal walk from A to C is not a path, as shown in Fig. 1.

Observation 1 *Let* $\mathcal{G} = (V, E, T, \beta)$ *be a temporal graph with maximum-waiting-time constraints. Then there can exist two vertices* $s, z \in V$ *such that no s-z-walk is a temporal path.*

We are interested in temporal walks within our proximity network in general, but wish to place emphasis on temporal walks that optimize certain properties. A plethora of criteria can be optimized as a consequence of the time aspect. Possible criteria (with the chosen names in brackets) include: arrival time (*foremost*), departure time (*reverse-foremost*), duration (*fastest*), transmission time (*shortest*), number of time-arcs (*minimum hop-count*), time-arc cost (*cheapest*), probability (*most-likely*), and waiting time (*minimum waiting time*). Next, we provide examples for all properties from their respective fields of application.

Foremost. A foremost walk is a temporal walk that has the earliest arrival time. Computing a foremost walk from a source vertex to all vertices in the proximity network determines the speed with which an infectious disease could spread.

Reverse-Foremost. A reverse-foremost walk is a temporal walk that exhibits the latest possible departure time. Computing a reverse-foremost walk from a source vertex to all vertices in the proximity network estimates the latest possible point in time at which an infectious disease could start spreading and still permeates the entire network.

Fastest. A fastest walk is a temporal walk which exhibits the minimum travel duration, that is, the minimum difference between departure and arrival times. For an appropriate motivation, we leave proximity networks and consider the field of flight networks. Airports represent vertices, time-arcs represent flights from one airport to another. The time stamp of a time-arc indicates the departure time of a flight,

the transmission time of the arc indicates the flight duration. The minimum waiting time in the vertices signifies the minimum time required to spend in an airport to catch a connecting flight. Within flight networks, the duration is often the criterion passengers aim to minimize in order to streamline their journeys.

Shortest. A shortest walk is a temporal walk that minimizes the sum of transmission times on the time-arcs. In the context of flight networks, a shortest walk is a flight connection with the minimum time spent airborne.

Minimum Hop-Count. A minimum-hop-count walk is a temporal walk which minimizes the number of time-arcs. Within a flight network, passengers also aim to minimize their number of connecting flights to avoid lengthy boarding procedures and the risk of missing connecting flights.

Cheapest. For a given cost function on the time-arcs, a cheapest walk is a temporal walk with the minimum sum of costs over all time-arcs. The benefits of the minimization of this property within flight networks are obvious: Weighing long travel times and multiple connections against the cheapest fare is the oldest consideration in the book for many air travelers.

Most-Likely. For given probabilities on the time-arcs, a most-likely walk is a temporal walk with the highest probability. One application lies in disease spreading: For every contact there is a certain likelihood for an infectious disease to be transmitted depending on the proximity of the persons or the body contact between them. Thus, a most-likely walk is a transmission route with the highest probability for the infectious disease to be spread under the simplifying assumption of stochastic independence. The respective probabilities of the time-arcs within the walk are multiplied.

Minimum Waiting Time. The minimum-waiting-time walk is a temporal walk whose total sum of waiting times over all intermediate vertices is minimum. Routing packets through a router network prioritizes minimum waiting times of packages in the routers to improve the overall performance of the network.

Formal definitions

In this section, we formally introduce the most important concepts related to temporal graphs, temporal walks, and formalize our optimality criteria. We start with some basic mathematical definitions. We refer to an interval $[a, b]$ as a contiguous ordered set of discrete time steps: $[a, b] := \{n \mid n \in \mathbb{N} \wedge a \leq n \leq b\}$, where $a, b \in \mathbb{N}$. Further, $[a] := [1, a]$. Given a function $f\colon A \to B$, we write $f \equiv c$ if $f(a) = c$ for all $a \in A$ and $c \in B$.

Temporal Graphs. A temporal graph is a graph whose arc set changes over time.

Definition 1 (Temporal Graph) *A temporal graph $\mathcal{G} = (V, E, T, \alpha, \beta)$ is a five-tuple consisting of*

- *a lifetime $T \in \mathbb{N}$,*
- *a vertex set V,*
- *a time-arc set $E \subseteq V \times V \times \{1, \dots, T\} \times \{0, \dots, T\}$,*
- *a minimum waiting time $\alpha\colon V \to \{0, \dots, T\}$, and*
- *a maximum waiting time $\beta\colon V \to \{0, \dots, T\}$.*

We remark that the central modeling aspect in our work—the minimum and maximum waiting times—is so far not commonly part of the definition of temporal graphs. A time-arc $(v, w, t, \lambda) \in E$ is a directed connection from v to w with *time stamp* t and *transmission time* λ, that is, a transmission from v to w starting at time step t and taking λ time steps to cross the arc. The *departure time* in vertex v is t; the *arrival time* in vertex w is then $t + \lambda$. The two waiting-time functions $\alpha\colon V \to \mathbb{N}$ and $\beta\colon V \to \mathbb{N}$ assign each vertex a minimum and maximum waiting time, respectively. The minimum waiting time $\alpha(v)$ is the minimum time a person has to stay in a vertex v before she can move on in the temporal graph. The maximum waiting time $\beta(v)$ is the maximum time a person can stay in a vertex v before she is no longer allowed to move further in the graph. A temporal graph $\mathcal{G} = (V, E, T, \alpha, \beta)$ is called *instantaneous* if $\alpha(v) = 0$ for all $v \in V$ and $\lambda = 0$ for all $(v, w, t, \lambda) \in E$. Then, for the ease of presentation, we neglect α and we write arcs as triples $(v, w, t) \in E$ for instantaneous graphs.

In Table 1, we introduce some notation for temporal graphs.

Temporal Walk. A temporal walk is a walk in a temporal graph such that the time stamps of the visited time-arcs of a temporal walk are non-decreasing in time. Additionally, the transmission time and the waiting-time constraints have to be taken into account.

Definition 2 (Temporal Walk) *Given a temporal graph* $\mathcal{G} = (V, E, T, \alpha, \beta)$ *and two vertices* $s, z \in V$*, a temporal walk of length* k *from* s *to* z *is a sequence* $((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k}$ *of time-arcs such that* $s = v_0$, $z = v_k$, *and* $t_i + \lambda_i + \alpha(v_i) \leq t_{i+1} \leq t_i + \lambda_i + \beta(v_i)$ *for all* $i \in [k-1]$.

A *temporal path* is a temporal walk where all vertices are pairwise distinct. We use the notation $P = ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k}$ for a temporal walk or a temporal path, that is, a sequence of k time-arcs. The value of k is *the length of* P and for each $i \in [k]$ the tuple $(v_{i-1}, v_i, t_i, \lambda_i)$ is a time-arc from v_{i-1} to v_i starting in time step t_i. The value of λ_i is the time needed to traverse the time-arc, that is, one arrives at time step $t_i + \lambda_i$ at v_i.

Optimal Temporal Walk. Due to the time aspect, there are several, potentially contradicting criteria that can be optimized in a temporal walk. We formally define the criteria that were already motivated in "Modeling of optimal temporal walks" section.

Table 1 Frequently used notation for a temporal graph $\mathcal{G}$. The first part shows global variables, the second part shows frequently used local variables

$\mathbb{N}$	the natural numbers (including 0) $\{0, \ldots\}$
V	the vertex set of $\mathcal{G}$
E	the time-arc set of $\mathcal{G}$
$[T]$	the time interval of $\mathcal{G}$
α	the minimum waiting time with $\alpha : V \to \mathbb{N}$
β	the maximum waiting time with $\beta : V \to \mathbb{N}$
V_t	the vertex subset $V_t \subseteq V$ at time t, that is, $V_t := \{v, w \mid (v, w, t, \lambda) \in E\}$
E_t	the time-arc subset at time t, that is, $E_t := \{(v, w) \mid (v, w, t, \lambda) \in E\}$
G_t	the directed static graph $G_t := (V_t, E_t)$
(v, w, t, λ)	a time-arc from u to v with time stamp t and transmission time λ
k	usually the number of time-arcs in an optimal walk
P	a walk; often $P = ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k}$ indicates the optimal walk

Definition 3 (Optimal Temporal Walk) *Let $\mathcal{G} = (V, E, T, \alpha, \beta)$ be a temporal graph, let $c\colon E \to \mathbb{N}$ be a cost function, and let $s, z \in V$ be two vertices. A temporal walk $P = ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k}$ from s to z is called optimal if it minimizes or maximizes a certain value among all temporal walks from s to z:*

criterion	*min / max*	*optimization value*
foremost	*min*	$t_k + \lambda_k$
reverse-foremost	*max*	t_1
fastest	*min*	$(t_k + \lambda_k) - t_1$
shortest	*min*	$\sum_{i=1}^{k} \lambda_i$
minimum hop-count	*min*	k
cheapest	*min*	$\sum_{i=1}^{k} c((v_{i-1}, v_i, t_i, \lambda_i))$
most-likely	*max*	$\prod_{i=1}^{k} c((v_{i-1}, v_i, t_i, \lambda_i))$
minimum waiting time	*min*	$\sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i)$

Note that the *most-likely* criterion can easily be transformed into *cheapest*. For the *most-likely* criterion, the cost values of the time-arcs represent probabilities, implying $c(e) \in [0, 1]$ for all $e \in E$. Hence, maximizing $\prod_{i=1}^{k} c((v_{i-1}, v_i, t_i, \lambda_i)$ is equivalent to minimizing

$$\sum_{i=1}^{k} -\log c((v_{i-1}, v_i, t_i, \lambda_i)).$$

Hence, we neglect considering the *most-likely* criterion separately. We further call a temporal walk $P = ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k}$ from s to z an optimal temporal walk with respect to a *linear combination* with $\delta_1, \ldots, \delta_7 \in \mathbb{Q}_0^+$ if it minimizes

$$\begin{aligned}
\mathrm{lin}(P) = \ & \delta_1 \cdot (t_k + \lambda_k) && \textit{Foremost} \\
& + \delta_2 \cdot (-t_1) && \textit{Reverse-Foremost} \\
& + \delta_3 \cdot (t_k + \lambda_k - t_1) && \textit{Fastest} \\
& + \delta_4 \cdot \sum_{i=1}^{k} \lambda_i && \textit{Shortest} \\
& + \delta_5 \cdot \sum_{i=1}^{k} c((v_{i-1}, v_i, t_i, \lambda_i)) && \textit{Cheapest} \\
& + \delta_6 \cdot k && \textit{Minimum Hop-Count} \\
& + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - (t_i + \lambda_i)) && \textit{Minimum Waiting Time}
\end{aligned}$$

among all temporal walks from s to z.

Transformation

To simplify the presentation of the forthcoming algorithm in "Algorithm" section for computing optimal temporal walks, we design it to run only on *instantaneous* temporal graphs, that is, temporal graphs with no transmission times ($\lambda = 0$ for all $(v, w, t, \lambda) \in E$) and no minimum-waiting-time constraints ($\alpha(v) = 0$ for all $v \in V$). This is no restriction since we can eliminate these with the following transformation.

Transformation 1 (Remove α and λ.) *Let $\mathcal{G} = (V, E, T, \alpha, \beta)$ be a temporal graph and let $c\colon E \to \mathbb{N}$ be a cost function. Transform $(\mathcal{G}, c)$ into $(\mathcal{G}', c', c_\lambda, ind, A)$ where*

- *$\mathcal{G}'$ is an instantaneous temporal graph $\mathcal{G}' = (V', E', T, \beta')$ with*
  - *$V' = V \cup V^E$ with $V^E := \{v_e \mid e \in E\}$,*
  - *$E' = E^O \cup E^I$ with $E^O := \{(v, v_e, t) \mid e = (v, u, t, \lambda) \in E\}$ and $E^I = \{(v_e, u, t + \lambda + \alpha(u)) \mid e = (v, u, t, \lambda) \in E\}$,*
  - *$\beta'\colon V' \to \mathbb{N}$ with $\beta'(v) := \beta(v)$ for all $v \in V$, else $\beta'(v) := T$,*
- *$c'\colon E' \to \mathbb{N}$ is a cost function with $c'(e) := c(\hat{e})$ for all $e = (v, v_{\hat{e}}, t) \in E^O$, else $c'(e) := 0$,*
- *$c_\lambda\colon E' \to \mathbb{N}$ is a transmission-cost function with $c_\lambda(e) := \ell$ for $e = (v_{\hat{e}}, w, t + \ell + \alpha(w)) \in E^I$ and $\hat{e} = (v, w, t, \ell) \in E$, else $c_\lambda(e) := 0$,*
- *$ind\colon V' \to \{0, 1\}$ is a vertex-index function with $ind(v) := 1$ if $v \in V$, else $ind(v) := 0$, and*
- *$A(v)\colon V' \to \{0, 1\}$ is an auxiliary function with $A(v) := \alpha(v)$ if $v \in V$, else $A(v) := 0$.*

We now show that any temporal graph can be transformed by Transformation 1 into an equivalent instantaneous temporal graph in linear time such that any optimal temporal walk in the instantaneous temporal graphs directly corresponds to an optimal temporal walk in the original graph and vice versa. To this end, we have to slightly adapt the formula for the linear combination as shown in the following proposition.

Proposition 1 *Let $\mathcal{G} = (V, E, T, \alpha, \beta)$ be a temporal graph, let $c\colon E \to \mathbb{N}$ be a cost function, and let $s, z \in V$. Let further $(\mathcal{G}' = (V', E', T', \beta'), c', c_\lambda, ind, A)$ be the result of applying Transformation 1 to $(\mathcal{G}, c)$.*

For $\delta_1, \ldots, \delta_7 \in \mathbb{Q}_0^+$, there exists a temporal walk $P := ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^k = (e_i)_{i=1}^k$ from s to z in $\mathcal{G}$ which is optimal with respect to a linear combination of $\delta_1, \ldots, \delta_7$ if and only if the temporal walk

$$P' := \left(\left(v_{i-1}, v_{e_i}, t_i\right), \left(v_{e_i}, v_i, t_i + \lambda_i + \alpha(v_i)\right)\right)\Big)_{i=1}^k = (e'_i)_{i=1}^{2k} = (v'_{i-1}, v'_i, t'_i)_{i=1}^{2k}$$

from s to z in $\mathcal{G}'$ is optimal with respect to the new formula for a linear combination of our optimality criteria defined as follows:

$$\begin{aligned}
lin^T(P') = \ & \delta_1 \cdot t'_{2k} && \textit{Foremost}\\
& + \delta_2 \cdot (-t'_1) && \textit{Reverse-Foremost}\\
& + \delta_3 \cdot (t'_{2k} - t'_1) && \textit{Fastest}\\
& + \delta_4 \cdot \sum_{i=1}^{2k} c_\lambda(e'_i) && \textit{Shortest}\\
& + \delta_5 \cdot \sum_{i=1}^{2k} c(e'_i) && \textit{Cheapest}\\
& + (\delta_6/2) \cdot 2k && \textit{Minimum Hop-Count}\\
& + \delta_7 \cdot \sum_{i=1}^{2k-1} ((t'_{i+1} - (t'_i - A(v'_i))) \cdot ind(v'_i)) && \textit{Minimum Waiting Time.}
\end{aligned}$$

Transformation 1 runs in $O(|V| + |E|)$ time.

Proof We will show that any temporal walk P in G corresponds to a temporal walk P' in G' such that $\text{lin}(P) = \text{lin}^{\text{T}}(P')$ and vice versa. To this end, observe that each vertex $v \in V^E$ has an in-going and an out-going arc and the vertex set $V = V' \setminus V^E$ is an independent set in G'. Hence, each temporal walk in G' from s to z alternately uses vertices in V and V^E. Since each vertex in V^E represents an arc in G, each temporal walk in G' has a unique representation in G and vice versa. It remains to show that $\text{lin}(P) = \text{lin}^{\text{T}}(P') - C$ for any temporal walk P in G where C is a constant only depending on the last vertex in P. Observe that by construction each vertex v_i in P is the same vertex as v'_{2i} in P' and the arc $(v_{i-1}, v_i, t_i, \lambda_i)$ is represented by the vertex v'_{2i} and the arcs e'_{2i-1} and e'_{2i}. It holds that $\lambda_i = c_\lambda(e'_{2i}) = c_\lambda(e'_{2i-1}) + c_\lambda(e'_{2i})$ and $c(e_i) = c'(e'_{2i}) = c'(e'_{2i-1}) + c'(e'_{2i})$. Finally, $t_{i+1} = t'_{2i+1}$ and $t_i + \lambda_i = t'_{2i} - A(v'_{2i})$ (recall $A(v) = \alpha(v)$ if $v \in V$ and $A(v) = 0$ otherwise) and hence $t_{i+1} - (t_i + \lambda_i) = t'_{2i+1} - (t'_{2i} - A(v'_{2i}))$. Thus,

$$
\begin{aligned}
&\text{lin}(P) = \\
&\quad \delta_1 \cdot (t_k + \lambda_k) + \delta_2 \cdot (-t_1) + \delta_3 \cdot (t_k + \lambda_k - t_1) + \delta_4 \cdot \sum_{i=1}^{k} \lambda_i \\
&\quad + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - (t_i + \lambda_i)) \\
&= \delta_1 \cdot (t_k + \lambda_k + \alpha(t_k) - \alpha(t_k)) + \delta_2 \cdot (-t_1) \\
&\quad + \delta_3 \cdot (t_k + \lambda_k + \alpha(t_k) - \alpha(t_k) - t_1) + \delta_4 \cdot \sum_{i=1}^{k} \lambda_i + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) \\
&\quad + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - (t_i + \lambda_i + \alpha(t_i) - \alpha(t_i))) \\
&= \delta_1 \cdot (t'_{2k-1} - A(v'_{2k})) + \delta_2 \cdot (-t'_1) + \delta_3 \cdot (t'_{2k-1} - A(v'_{2k}) - t'_1) \\
&\quad + \delta_4 \cdot \sum_{i=1}^{2k} c_\lambda(e'_i) + \delta_5 \cdot \sum_{i=1}^{2k} c'(e'_i) + (\delta_6/2) \cdot 2k \\
&\quad + \delta_7 \cdot \left(\sum_{i=1}^{k} ((t'_{2i} - (t'_{2i-1} - A(v'_{2i-1}))) \cdot 0) + \sum_{i=1}^{k-1} (t'_{2i+1} - (t'_{2i} - A(v'_{2i}))) \right) \\
&= \delta_1 \cdot (t'_{2k-1} - A(v'_{2k})) + \delta_2 \cdot (-t'_1) \\
&\quad + \delta_3 \cdot (t'_{2k-1} - A(v'_{2k}) - t'_1) + \delta_4 \cdot \sum_{i=1}^{2k} c_\lambda(e'_i) + \delta_5 \cdot \sum_{i=1}^{2k} c'(e'_i) \\
&\quad + (\delta_6/2) \cdot 2k + \delta_7 \cdot \sum_{i=1}^{2k-1} ((t'_{i+1} - (t'_i - A(v'_i))) \cdot \text{ind}(v'_i)) \\
&= \text{lin}^{\text{T}}(P') - (\delta_1 + \delta_3) \cdot A(v'_{2k}).
\end{aligned}
$$

Observe that $(\delta_1 + \delta_3) \cdot A(v'_{2k})$ is independent of the temporal walk P' and hence any optimal walk P in G corresponds to an optimal temporal walk P' in G'. Lastly, notice that Transformation 1 runs in $O(|V| + |E|)$ time. □

The algorithm for computing optimal temporal walks that we will introduce in the forthcoming section will find temporal walks optimizing the formula $\text{lin}^{\text{T}}(\cdot)$ introduced

in Proposition 1. For instantaneous temporal graphs, where we do not have to use Transformation 1, optimizing according to $\mathrm{lin}^{\mathrm{T}}(\cdot)$ is not a drawback as stated in the following (recall that the definitions of lin and $\mathrm{lin}^{\mathrm{T}}$ depend on the parameters $\delta_1, \ldots, \delta_7$):

Observation 2 *Let $\mathcal{G} = (V, E, T, \beta)$ be an instantaneous temporal graph and let $c\colon E \to \mathbb{N}$ be a cost function. Let further $P := ((v_{i-1}, v_i, t_i, \lambda_i))_{i=1}^{k} = (e_i)_{i=1}^{k}$ be a temporal walk in $\mathcal{G}$. For $\delta_1, \ldots, \delta_7 \in \mathbb{Q}_0^+$, it holds that*

$$lin(P) = lin^T(P) + (\delta_6/2) \cdot k$$

for $c' = c$, $c_\lambda \equiv 0$, $ind \equiv 1$, and $A \equiv 0$.

Algorithm

In this section, we present a single-source optimal-walks algorithm with respect to any linear combination of our optimality criteria. That is, given a temporal graph $\mathcal{G} = (V, E, T, \beta)$, a cost function $c\colon E \to \mathbb{N}$, and a source vertex $s \in V$, we compute optimal temporal walks with respect to any linear combination with $\delta_1, \ldots, \delta_7 \in \mathbb{Q}$ from s to all vertices in the temporal graph (if they exist). To this end, we first apply Transformation 1 to $\mathcal{G}$ to obtain an instantaneous temporal graph. Algorithm 1 then performs for each $t \in [T]$ three main steps:

GraphGeneration. Generate graph G which only contains the arcs present at time step t and add arcs from s to each vertex v in G that has been reached within the last $\beta(v)$ time steps.

ModDijkstra. Run a modified version of Dijkstra's algorithm to compute for each v in G the optimal walk from s to v that arrives at time step t (if it exists).

Update. Update a list storing information on optimal walks from s to each $v \in V$. More precisely, for each $v \in V$ a list of tuples of arrival times and corresponding optimal values is updated.

Efficiently storing and accessing the value of an optimal walk from s to v that arrives at a certain time step t is the heart of the algorithm. We can maintain this information in $O(|E|)$ time during a run of Algorithm 1 such that this information can be accessed in constant time, leading to the following theorem:

Theorem 1 *With respect to any linear combination of the optimality criteria, optimal temporal walks from a source vertex s to each vertex in a temporal graph $\mathcal{G} = (V, E, T, \beta)$ can be computed in $O(|V| + |E| \log |E|)$ time.*

Algorithm Details. Let $\mathcal{G}'$ be a temporal graph with a cost function $c'\colon E \to \mathbb{N}$ and let $s \in V$ be the source. We can apply Transformation 1 to $(\mathcal{G}', c')$ to obtain $(\mathcal{G} = (V, E, T, \beta), c, c_\lambda, \mathrm{ind}, A)$ where $\mathcal{G}$ is an instantaneous temporal graph. If $\mathcal{G}'$ is already an instantaneous temporal graph, then we set $c = c'$, $c_\lambda \equiv 0$, $\mathrm{ind} \equiv 1$, and $A \equiv 0$ as justified by Observation 2.

Algorithm 1: Computes optimal walks.

Input: An instantaneous temporal graph $\mathcal{G} = (V, E, T, \beta)$, two cost functions c, c_λ, two vertex functions ind, A, and a source vertex $s \in V$.

Output: For each $v \in V$ the specific length of an optimal s-v walk.

Variables :

$\text{opt}(v)$	stores the value of an optimal walk from s to v within time interval $[0, t]$;
$L(v)$	is a sorted list $[(\text{opt}_{a_1}, a_1), \ldots, (\text{opt}_{a_k}, a_k)]$ where opt_{a_i} is an optimal value of a walk from s to v that arrives at time a_i with $t + \beta(v) \leq a_i \leq t$.
$\delta_1, ..., \delta_7$	linear combination of the optimality criteria *foremost*, *reverse-foremost*, *fastest*, *shortest*, *cheapest*, *minimum hop-count*, and *minimum waiting time*, respectively.

Initialize $\text{opt}(v) = \infty$ and $L(v)$ as empty list for all $v \in V \setminus \{s\}$
for $t = 1, \ldots, T$ *with* $E_t \neq \emptyset$ **do**
  $G, d_t, d_r \leftarrow$ `generateGraph(`G_t`)`
  $V', \text{opt}_t \leftarrow$ `modDijkstra(`G, d_t, d_r`)`
  for $v \in V'$ **do**
    `/* Update step */`
    $\text{opt}(v) \leftarrow \min\{\text{opt}(v), \delta_1 \cdot t - \delta_2 \cdot T + \delta_3 \cdot (t - T) + \text{opt}_t(v)\}$
    $L(v) \leftarrow$ append $(\text{opt}_t(v), t)$ and delete *redundant* tuples (see Lemma 1)
return opt

`/* GraphGeneration step: Generate graph` G `which only contains the arcs present at time step` t `and add arcs from` s `to each vertex` v `in` G `that has been reached within the last` $\beta(v)$ `time steps. */`

function `generateGraph(`G_t`)`**:**
  Initialize $E_r \leftarrow \emptyset$; $d_r(v, w) \leftarrow \infty$ and $d_t(v, w) \leftarrow \infty$ for all $v, w \in V_t \cup \{s\}$
  for $(v, w) \in E_t$ **do**
    $$d_t(v, w) \leftarrow \begin{cases} (\delta_2 + \delta_3) \cdot (T - t) + \delta_4 \cdot c_\lambda(v, w, t) + \delta_5 \cdot c(v, w, t) + \delta_6 & \text{if } v = s \\ \delta_4 \cdot c_\lambda(v, w, t) + \delta_5 \cdot c(v, w, t) + \delta_6 & \text{else} \end{cases}$$
  for $v \in V_t \setminus \{s\}$ **do**
    delete tuples (opt_a, a) in $L(v)$ with $a + \beta(v) < t$
    if $L(v)$ is not empty **then**
      $E_r \leftarrow E_r \cup \{(s, v)\}$
      $\text{opt} \leftarrow \min\{\text{opt}_a \mid (\text{opt}_a, a) \in L(v)\}$
      $d_r(s, v) \leftarrow \text{opt} + \delta_7 \cdot \text{ind}(v) \cdot (t - a + A(v))$
  return $((V_t \cup \{s\}, E_t \cup E_r), d_t, d_r)$

`/* ModDijkstra step: Returns all vertices` V' `with an arrival time exactly at time step` t `and their optimization value` $d\colon V' \to \mathbb{N}$. `*/`

function `modDijkstra(`$(V, E_t \cup E_r), d_t, d_r$`)`**:**
  initialize $\text{opt}_t(v) \leftarrow \infty$, $r(v) \leftarrow \infty$ for all $v \in V_t$, and $r(s) = 0$
  initialize $Q \leftarrow V$ and $V' \leftarrow \emptyset$
  while $Q \neq \emptyset$ **do**
    $v \leftarrow$ vertex in Q with minimum $r(v)$
    remove v from Q
    for $(v, w) \in E_t \cup E_r$ **do**
      $r(w) \leftarrow \min\{r(w), r(v) + \min\{d_t(v, w), d_r(v, w)\}\}$
      if $(v, w) \in E_t$ **then**
        $\text{opt}_t(w) \leftarrow \min\{\text{opt}_t(w), r(v) + d_t(v, w)\}$
        $V' \leftarrow V' \cup \{w\}$
  return V', opt_t

For given $\delta_1, \ldots, \delta_7 \in \mathbb{Q}$, Algorithm 1 for all $v \in V$ now computes an optimal walk $P = ((v_{i-1}, v_i, t_i))_{i=1}^{k} = (e_i)_{i=1}^{k}$ from s to v with respect to

$$\begin{aligned}\mathrm{lin}^{\mathrm{T}}(P) =& \delta_1 \cdot (t_k) + \delta_2 \cdot (-t_1) + \delta_3 \cdot (t_k - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\ &+ \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_{i+1})) \cdot \mathrm{ind}(v_i).\end{aligned}$$

We have shown in Proposition 1 that an optimal walk with respect to $\mathrm{lin}^{\mathrm{T}}(P)$ in $\mathcal{G}$ directly corresponds to an optimal walk with respect to $\mathrm{lin}(P)$ in the original temporal graph $\mathcal{G}'$.

For each vertex $v \in V \setminus \{s\}$, Algorithm 1 stores in $\mathrm{opt}(v)$ the value of an optimal walk from s to v and in $L(v)$ a list of all relevant arrival times from s to v with their optimal values. In the beginning, $\mathrm{opt}(v) = \infty$ and $L(v)$ is an empty list (Line 1 in Algorithm 1). Then, for each time step t and each $v \in V$, Algorithm 1 computes the optimal walk from the source s to v that arrives in time step t (if it exists). Thus, Algorithm 1 performs for each $t \in \{1, \ldots, T\}$ the following steps:

GraphGeneration. Generate a static graph G with `GenerateGraph` (Line 3 and Lines 9 to 19). This graph consists of the static graph $G_t = (V_t, E_t)$, that is, the static graph induced by all time-arcs with time stamp t, and the source vertex s.

The weight of an arc $(v, w) \in E_t$ is set to $\delta_4 \cdot c_\lambda(v, w) + \delta_5 \cdot c(v, w) + \delta_6$. If further $v = s$, then we have to add $(\delta_2 + \delta_3) \cdot (T - t)$ to take the departure time in s into account for the criteria *reverse-foremost* and *fastest*, see Line 12.

Additionally, non-existing arcs from s to each vertex $v \in V_t$ are added if there exists a temporal walk from s to v that arrived not later than $\beta(v)$ time steps ago. Let opt_a be the optimal value among all walks that arrive within the time interval $[t - \beta(v), t]$ and let a be the corresponding arrival time in v. Additionally, the minimum waiting time $A(v)$ plus the additional waiting time $(t - a)$ in v has to be taken into account if $\mathrm{ind}(v) = 1$. Hence, the weight of arc (s, v) is set to $\mathrm{opt}_a + \delta_7 \cdot \mathrm{ind}(v) \cdot (t - a + A(v))$, see Line 18. Let E_r be the set of these additional arcs. Then, $G = (V_t \cup \{s\}, E_t \cup E_r, d_t, d_r)$.

ModDijkstra. Run a modified Dijkstra Algorithm on G with `modDijkstra` (Line 4 and Lines 20 to 31). Instead of computing a shortest walk (using the original Dijkstra Algorithm) in G, compute a shortest walk among all walks that end in an arc of E_t. This represents a temporal walk that arrives in time step t with optimal value. The function `modDijkstra` returns the set V' of vertices that can be reached within G via an arc in E_t and the function $\mathrm{opt}_t \colon V' \to \mathbb{N}$ that maps each vertex $v \in V'$ to its optimal value of a walk from s to v arriving exactly at time t.

Update. For each $v \in V'$, set the optimum $\mathrm{opt}(v)$ to the minimum of its current value and the optimal value of a newly computed walk, that is, $\mathrm{opt}(v) := \min\{\mathrm{opt}(v), \delta_1 \cdot t + \delta_3 \cdot (t - T) + \mathrm{opt}_t(v)\}$ (Line 6). Herein, we have to add the arrival time t that has not been taken into account in the calculation of the optimal value because it is the same for all walks found at time step t. Add the tuple $(\mathrm{opt}_t(v), t)$ to list $L(v)$ (Line 7).

After the **Update** step for time step t, the list $L(v)$ contains all tuples $(\mathrm{opt}_{\mathrm{ar}}, \mathrm{ar})$ such that there exists a walk from s to v that arrives at time $\mathrm{ar} \in [t - \beta(v), t]$ with its optimal cost value $\mathrm{opt}_{\mathrm{ar}}$. We want to have constant-time access to the optimal value of a walk

that arrives in v within time interval $[t-\beta(v), t]$. This can be achieved by deleting tuples from list $L(v)$ that are *redundant*, that is these tuples are not meaningful for the correct computation of optimal walks. Let

$$L(v) = [(\text{opt}_{a_1}, a_1), \ldots, (\text{opt}_{a_k}, a_k)]$$

be such a list for a time step t with $t-\beta(v) \leq a_1 < \ldots < a_k \leq t$. A tuple $(\text{opt}_{\text{ar}}, \text{ar})$ is redundant if there exists a tuple with an arrival time greater than ar such that its optimal value is smaller than opt_{ar} plus the additional waiting time. This is shown with the following lemma:

Lemma 1 *For a time step $t \in \{1, \ldots, T\}$ and a vertex $v \in V$, if there are two tuples $(opt_{a_i}, a_i), (opt_{a_j}, a_j) \in L(v)$ with $a_i < a_j$ and*

$$opt_{a_j} \leq opt_{a_i} + \delta_7 \cdot ind(v) \cdot (a_j - a_i),$$

then (opt_{a_i}, a_i) is redundant and can be removed from $L(v)$.

Proof After all time-arcs with time stamp t have been processed, Algorithm 1 only considers time-arcs with time stamp $t' > t$. In the generated graph G (Line 3), the algorithm adds an arc from s to $v \in V_{t'}$ if a walk from s arrives in v within $[t'-\beta(v), t']$. If $a_i \in [t'-\beta(v), t']$, then $a_j \in [t'-\beta(v), t']$ because $a_i < a_j < t'$. Furthermore, let (opt, a) be the optimal value and the arrival time of a walk from s to v that arrives within time interval $[t'-\beta(v), t']$ that minimizes

$$\text{opt} + \delta_7 \cdot \text{ind}(v)(t' - a + A(v)).$$

Then, the weight of the arc (s, v) is set to this value. Since $a_i < a_j \in [t'-\beta(v), t']$ and $\text{opt}_{a_i} + \delta_7 \cdot \text{ind}(v) \cdot (a_j - a_i) \geq \text{opt}_{a_j}$, we know that

$$\text{opt}_{a_j} + \delta_7 \cdot \text{ind}(v)(t' - a_j + A(v)) \leq \text{opt}_{a_i} + \delta_7 \cdot \text{ind}(v)(t' - a_i + A(v)).$$

Hence, the tuple (opt_{a_i}, a_i) is not needed in the list $L(v)$ at time step t and can be removed. □

If $L(v)$ does not contain any redundant tuples, then it also holds that

$$\text{opt}_{a_1} + \delta_7 \cdot \text{ind}(v) \cdot (a_2 - a_1) < \cdots < \text{opt}_{a_k}.$$

Hence, (a_1, opt_{a_1}) contains the optimal value and arrival time of a walk that arrives within time interval $[t-\beta(v), t]$ and minimizes $\text{opt} + \delta_7 \cdot \text{ind}(v)(t' - a + A(v))$. It follows that finding $\text{opt} = \min\{\text{opt}_a \mid (\text{opt}_a, a) \in L(v)\}$ in Line 18 takes constant time. The deletion of redundant tuples takes $O(|E|)$ time during the whole run of Algorithm 1. With these considerations at hand, we can derive the following lemma.

Lemma 2 *Algorithm 1 runs in $O(|V| + |E| \log |E|)$ time.*

Proof The initialization in Algorithm 1 can be done in $O(|V|)$ time. Furthermore, the time-arcs have to be sorted by time stamps which takes $O(|E| \log |E|)$ time. Then, for each time step $t \in [T]$, Algorithm 1 generates in $O(|E_t| + |V_t|)$ time a static directed graph $G = (V_t \cup \{s\}, E_t \cup E_r)$ with $O(|V_t|)$ vertices and $O(|E_t| + |V_t|)$ arcs.

For each generated graph G, `modDijkstra` is executed in $O(|E_t| \log |E_t|)$ time. The updates of opt and L afterwards run in $O(|V_t|)$ time. Note that $|V_t|$ is the number of

vertices that have at least one in-going or out-going time-arc at time step t. Consequently, it holds that $|V_t| \leq 2|E_t|$.

Due to the sorting of $L(v)$, maintaining these lists in Lines 15 and 7 takes $O(|E|)$ time during the whole run of the algorithm as shown Lemma 1. In the list $L(v)$, we delete at most as many elements as there are time-arcs in the temporal graph. Recall that if $(\text{opt}_a, a) \in L(v)$, then there exists a time-arc $(w, v, a) \in E$.

We can analyze the running time by

$$\begin{aligned} & O\big(|V| + |E| + \sum_{t=1}^{T}(|E_t|) + (|V_t| \log |V_t|)\big) \\ = \; & O\big(|V| + |E| + \sum_{t=1}^{T}(|E_t|) + (|E_t| \log |E_t|)\big) \\ = \; & O\big(|V| + |E| + \sum_{t=1}^{T}|E_t| \log |E_t|\big) \\ \subseteq \; & O\big(|V| + |E| \log |E|\big). \end{aligned}$$

Hence, Algorithm 1 runs in $O\big(|V| + |E| \log |E|\big)$ time. □

Next, we show the correctness of Algorithm 1. We prove that for every time step t and for every vertex v, Algorithm 1 computes an optimal walk from s to v that arrives at time step t (if it exists).

Lemma 3 *For a time step $t \in [T]$ and a vertex $v \in V$, Algorithm 1 computes the optimal value of a temporal walk from s to v that arrives exactly in time step t.*

Proof The proof is by induction on the time step $t \in \{1, \ldots, T\}$.

In the beginning, $L(v)$ is empty. For $t = 1$, the algorithm generates a graph $G = (V_1 \cup \{s\}, E_1)$. For all arcs $(s, w) \in E_1$, the weights are set to

$$d_1(s, w) := (\delta_2 + \delta_3) \cdot (T - 1) + \delta_4 \cdot c_\lambda((v, w, 1)) + \delta_5 \cdot c((v, w, 1)) + \delta_6;$$

for the other arcs $(v, w) \in E_1$ the weights are set to

$$d_1(v, w) = \delta_4 \cdot c_\lambda(v, w) + \delta_5 \cdot c((v, w, 1)) + \delta_6.$$

Note that if there is an optimal temporal walk arriving in time step 1, then there also exists an optimal temporal walk arriving at time step 1. Now if there is an optimal walk $P = ((v_{i-1}, v_i, 1))_{i=1}^{k} = (e_i)_{i=1}^{k}$ from s to a vertex $v \in V$ that arrives exactly in time step 1, then there exists a path $P' = ((v_{i-1}, v_i))_{i=1}^{k} = (a_i)_{i=1}^{k}$ from s to v in G with value

$$\begin{aligned} \sum_{i=1}^{k} d_t(a_i) &= (\delta_2 + \delta_3) \cdot (T - 1) + \sum_{i=1}^{k} \delta_4 \cdot c_\lambda(v_{i-1}, v_i) + \delta_5 \cdot c(v_{i-1}, v_i) + \delta_6 \\ &= \text{opt}_1(v). \end{aligned}$$

Algorithm 1 finds in `modDijkstra` the path P', adds $(\mathrm{opt}_1(v), 1)$ to $L(v)$ and sets

$$\begin{aligned}
\mathrm{opt}(v) :=\ & \delta_1 \cdot 1 - \delta_2 \cdot T + \delta_3 \cdot (1 - T) + \mathrm{opt}_1(v) \\
=\ & \delta_1 \cdot 1 - \delta_2 \cdot T + \delta_3 \cdot (1 - T) + (\delta_2 + \delta_3) \cdot (T - 1) + \sum_{i=1}^{k} \delta_4 \cdot c_\lambda(v_{i-1}, v_i) \\
& + \delta_5 \cdot c(v_{i-1}, v_i) + \delta_6 \cdot k \\
=\ & \delta_1 \cdot 1 - \delta_2 \cdot 1 + \delta_3 \cdot 0 + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
=\ & \mathrm{lin}^{\mathrm{T}}(P).
\end{aligned}$$

Note that $A(v_i) = 0$ for $i \in [k-1]$ by Transformation 1 in time step 1. If there exists an optimal path P^* in G, then this directly translates to the existence of a temporal path P' that arrives also in time step 1 with a smaller optimal value than P, contradicting the assumption that P is optimal.

Now, let us assume that for all time steps $t' \in \{1, \ldots, t\}$ Algorithm 1 computed the optimal value opt of a walk from s to $v \in V$ that arrives exactly in time step t' and added $(\mathrm{opt} - \delta_1 \cdot t' + \delta_2 \cdot T - \delta_3 \cdot (t' - T), t')$ to $L(v)$. If for time step $t+1$ a vertex $v \in V$ has no in-going time-arc with time step $t+1$, then there cannot exist a temporal walk from s to v that arrives exactly in time step $t+1$. Thus, only vertices in V_{t+1} are candidates for a temporal walk that arrives exactly in time step $t+1$.

Let $v \in V_{t+1}$ be a vertex such that there is a temporal walk from s to v that arrives exactly in time step $t+1$. Let $P = ((v_{i-1}, v_i, 1))_{i=1}^{k} = (e_i)_{i=1}^{k}$ be an optimal walk from s to v that arrives exactly in time step $t+1$ with the optimal value

$$\begin{aligned}
\mathrm{lin}^{\mathrm{T}}(P) = \ & \delta_1 \cdot t_1 - \delta_2 \cdot t_k + \delta_3 \cdot (t_k - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i).
\end{aligned}$$

Assume towards a contradiction that Algorithm 1 does not find a walk from s to v with optimal value $\mathrm{lin}^{\mathrm{T}}(P)$.

First consider the case that $t_i = t+1$ for all $i \in [k]$, that is, all time-arcs of the temporal walk P have time stamp $t+1$. Then, we can assume that P is a temporal path. Hence there exists a path $P' = ((v_{i-1}, v_i))_{i=1}^{k} = (a_i)_{i=1}^{k}$ from s to v in G_{t+1} and therefore in G with optimal value

$$\begin{aligned}
\sum_{i=1}^{k} d_t(a_i) &= (\delta_2 + \delta_3) \cdot (T - (t+1)) + \sum_{i=1}^{k} \delta_4 \cdot c_\lambda(v_{i-1}, v_i) + \delta_5 \cdot c(v_{i-1}, v_i) + \delta_6 \\
&= \mathrm{opt}_{t+1}(v).
\end{aligned}$$

Algorithm 1 finds in `modDijkstra` the path P', adds $(\mathrm{opt}_{t+1}(v), t+1)$ to $L(v)$, and updates $\mathrm{opt}(v)$ to the minimum of $\mathrm{opt}(v)$ and

$$\begin{aligned}
&\delta_1 \cdot (t+1) - \delta_2 \cdot T + \delta_3 \cdot (t+1-T) + \mathrm{opt}_{t+1}(v) \\
=\ &\delta_1 \cdot (t+1) - \delta_2 \cdot T + \delta_3 \cdot (t+1-T) + (\delta_2+\delta_3) \cdot (T-(t+1)) \\
&+ \sum_{i=1}^{k} \delta_4 \cdot c_\lambda(v_{i-1}, v_i) + \delta_5 \cdot c(v_{i-1}, v_i) + \delta_6 \cdot k \\
=\ &\delta_1 \cdot t_1 - \delta_2 \cdot t_k + \delta_3 \cdot (t_k - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
&+ \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
=\ &\mathrm{lin}^{\mathrm{T}}(P).
\end{aligned}$$

Note again that $A(v_i) = 0$ for $i \in [k-1]$ if $t_{i-1} = t_i$ by Transformation 1. Hence, we find a walk from s to v at time step $t+1$ with optimal value $\mathrm{lin}^{\mathrm{T}}(P)$, contradicting our assumption.

Now assume for P that there exists an $\ell \in \{1, \dots, k-1\}$ such that for $j \in [\ell]$ it holds that $t_j < t+1$ and for $j' \in \{i+1, \dots, k\}$ it holds that $t_{j'} = t+1$. The temporal walk $P_\ell = ((v_{i-1}, v_i, t_i))_{i=1}^{\ell} = (e_i)_{i=1}^{k}$ is an optimal subwalk from s to v_ℓ that arrives exactly in time step t_ℓ, otherwise P is not optimal because it could be improved by replacing P_ℓ. The subwalk P_ℓ has an optimal value

$$\begin{aligned}
\mathrm{lin}^{\mathrm{T}}(P_\ell) = \ &\delta_1 \cdot t_1 - \delta_2 \cdot t_\ell + \delta_3 \cdot (t_\ell - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
&+ \delta_5 \cdot \sum_{i=1}^{\ell} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{\ell-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i).
\end{aligned}$$

Then

$$\begin{aligned}
\mathrm{opt}_{t_\ell}(v_\ell) = \ &\mathrm{opt}_{P_\ell} - \delta_1 \cdot t_\ell + \delta_2 \cdot T - \delta_3 \cdot (t_\ell - T) \\
= \ &(\delta_2+\delta_3) \cdot (T - t_\ell) + \delta_4 \cdot \sum_{i=1}^{\ell} c_\lambda(e_i) \\
&+ \delta_5 \cdot \sum_{i=1}^{\ell} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{\ell-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i).
\end{aligned}$$

By our induction hypothesis, the tuple $(\mathrm{opt}_{t_\ell}(v_\ell), t_\ell)$ was added to $L(v_\ell)$. If the tuple $(\mathrm{opt}_{t_\ell}(v_\ell), t_\ell)$ is not in $L(v_\ell)$ in time step $t+1$, then due to Lemma 1 there must be another tuple $(\mathrm{opt}_{\hat{t}}(v_\ell), \hat{t})$ in $L(v_\ell)$ with $t_\ell < \hat{t} < t+1 < t_\ell + \beta(v_\ell) \leq \hat{t} + \beta(v_\ell)$ and

$$\mathrm{opt}_{t_\ell}(v_\ell) + \delta_7 \cdot \mathrm{ind}(v_\ell)(t+1-t_\ell + A(v_\ell)) = \mathrm{opt}_{\hat{t}}(v_\ell) + \delta_7 \cdot \mathrm{ind}(v_\ell)(t+1-\hat{t} + A(v_\ell)).$$

Otherwise P is not optimal because it could be improved by replacing P_ℓ by the temporal walk represented by $(\mathrm{opt}_{\hat{t}}(v_\ell), \hat{t})$.

Now consider the generated graph $G = (V_{t+1} \cup \{s\}, E_{t+1} \cup E_r)$. The arc sequence $P_{t+1} = ((v_{i-1}, v_i))_{i=\ell+1}^{k} = (a_i)_{i=\ell+1}^{k}$ is a path in $G_{t+1} = (V_{t+1}, E_{t+1})$ and, thus, contained in G. The arc $a_\ell = (s, v_\ell)$ is contained in E_r with weight

$$d_r(s, v_\ell) = \mathrm{opt}_{t_\ell}(v_\ell) + (t+1-t_\ell + A(v_\ell)) \cdot \mathrm{ind}(v_\ell).$$

Thus, there is a walk from s to v in G and `modDijkstra` on G returns the vertex v because $a_k \in E_{t+1}$ with the optimal value

$$
\begin{aligned}
& d_r(a_\ell) + \sum_{i=\ell+1}^{k} d_t(a_i) \\
= \; & \mathrm{opt}_{t_\ell}(v_i) + \mathrm{ind}(v_\ell) \cdot (t + 1 - t_\ell + A(v_\ell)) \\
& + \sum_{i=\ell+1}^{k} \delta_4 \cdot c_\lambda(v_{i-1}, v_i, t+1) + \delta_5 \cdot c(v_{i-1}, v_i, t+1) + \delta_6 \\
= \; & (\delta_2 + \delta_3) \cdot (T - t_\ell) + \sum_{i=1}^{\ell} \left(\delta_4 \cdot c_\lambda(e_i) + \delta_5 \cdot c(e_i) + \delta_6\right) \\
& + \delta_7 \cdot \sum_{i=1}^{\ell-1} \mathrm{ind}(v_i) \cdot (t_{i+1} - t_i + A(v_i)) \\
& + \mathrm{ind}(v_\ell) \cdot (t + 1 - t_\ell + A(v_\ell)) \\
& + \sum_{i=\ell+1}^{k} \left(\delta_4 \cdot c_\lambda(v_{i-1}, v_i) + \delta_5 \cdot c(v_{i-1}, v_i) + \delta_6\right) \\
= \; & (\delta_2 + \delta_3) \cdot (T - t_\ell) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{\ell} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
= \; & (\delta_2 + \delta_3) \cdot (T - t_\ell) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
= \; & \mathrm{opt}_{t+1}(v).
\end{aligned}
$$

Note that by Transformation 1 $A(v_i) = 0$ for $i \in [\ell + 1, k-1]$ if $t_{i-1} = t_i$. Consequently, the tuple $(\mathrm{opt}_{t+1}(v), t+1)$ is added to $L(v)$ and $\mathrm{opt}(v)$ is set to the minimum of its current value and

$$
\begin{aligned}
& \delta_1 \cdot (t+1) - \delta_2 \cdot T + \delta_3(t + 1 - T) + \mathrm{opt}_{t+1}(v) \\
= \; & \delta_1 \cdot (t+1) - \delta_2 \cdot T + \delta_3(t + 1 - T) \\
& + (\delta_2 + \delta_3) \cdot (T - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
= \; & \delta \cdot (t+1) + \delta_2 \cdot (t_k) + \delta_3(t + 1 - t_1) + \delta_4 \cdot \sum_{i=1}^{k} c_\lambda(e_i) \\
& + \delta_5 \cdot \sum_{i=1}^{k} c(e_i) + \delta_6 \cdot k + \delta_7 \cdot \sum_{i=1}^{k-1} (t_{i+1} - t_i + A(v_i)) \cdot \mathrm{ind}(v_i) \\
= \; & \mathrm{lin}^{\mathrm{T}}(P).
\end{aligned}
$$

This is a contradiction to our assumption.

Lastly, observe that the algorithm only computes temporal walks that are contained in G as it only uses arcs from G_t which all correspond to temporal walks in G (single arcs or longer walks starting in s). Thus, for $t \in \{1, \dots, T\}$, Algorithm 1 computes an optimal temporal walk from s to $v \in V$ that arrives exactly in time step t. □

Based on this statement, we can finally prove the correctness of Algorithm 1. This concludes the proof of Theorem 1.

Lemma 4 *Algorithm 1 computes optimal walks from a given source vertex s to all vertices.*

Proof Let $P = ((v_{i-1}, v_i, 1))_{i=1}^{k} = (e_i)_{i=1}^{k}$ be a walk with minimum $\mathrm{lin}^{\mathrm{T}}(P)$ among all temporal walks from s to a vertex v. The walk P is also an optimal walk from s to v that arrives exactly in time step t_k. This is computed by Algorithm 1 in time step t_k as shown in Lemma 3. □

Experimental results

We implemented Algorithm 1 and performed experimental studies including comparisons to existing state-of-the-art algorithms by Wu et al. (2016). Note that the algorithms by Wu et al. (and our algorithm as well) are Dijkstra-like algorithms adjusted to the specific temporal setting. A notable difference is that our algorithm stops the graph exploration if no more vertex can be visited, whereas the algorithms by Wu et al. (2016) always iterate over all time-arcs. We show that our algorithm—while being able to solve a more general problem—can compete with these algorithms on real-world instances for the special case of computing temporal walks with no maximum-waiting-time constraints. We further examine the influence of different maximum-waiting-time values on the existence and structure (e.g., the number of cycles) of optimal temporal walks and on the running time of Algorithm 1.

Setup and statics

We implemented Algorithm 1 in C++ (v11) and performed our experiments on an Intel Xeon E5-1620 c omputer with 64 GB of RAM and four cores clocked at 3.6 GHz each. The operating system was Debian GNU/Linux 7.0 where we compiled the program with GCC v7.3.0 on optimization level -O3. We compare Algorithm 1 to the algorithms of Wu et al. (2016) using their C++ code and testing it on the same hardware and with the same compiler. We tested our algorithm on the same freely available data sets as Wu et al. (2016) from the well-established KONECT library (KONECT 2017). The graphs are listed in Table 2 with some relevant statistics.

For each optimization criterion, each $\beta \equiv c, c \in \{1, 2, 4, 8, \dots, 2^{\lceil \log T \rceil}\}$, and each data set, Algorithm 1 ran for 100 fixed source vertices of the data set chosen independently and uniformly at random. Our open source code is freely available at https://fpt.akt.tu-berlin.de/temporalwalks.

Findings

In the following, we first compare Algorithm 1 to the algorithm by Wu et al. (2016) in terms of running times in our experiments. In the second part, we analyze the effects of different maximum-waiting-time values β on Algorithm 1.

Table 2 Statistics for the real-world data sets used in our experiments (same freely available data sets as used by Wu et al. (2016), taken from the KONECT library (KONECT 2017)). The column "T" shows the lifetime of the temporal graph (after factoring out the precision in which time-arcs are measured) and the column "τ" shows the number of time steps of the graph for which there exists at least one time-arc

File	$\lvert V\rvert$	$\lvert E\rvert$	T	τ
elec	7,118	$1 \cdot 10^5$	$1.19 \cdot 10^8$	$1 \cdot 10^5$
facebook-wosn-links	63,731	$8.2 \cdot 10^5$	$1.23 \cdot 10^9$	$7.4 \cdot 10^5$
epinions	$1.3 \cdot 10^5$	$8.4 \cdot 10^5$	$8.16 \cdot 10^7$	939
enron	87,273	$1.1 \cdot 10^6$	$1.4 \cdot 10^9$	$2.1 \cdot 10^5$
digg-friends	$2.8 \cdot 10^5$	$1.7 \cdot 10^6$	$1.25 \cdot 10^9$	82,641
ca-cit-HepPh	28,093	$4.6 \cdot 10^6$	$3.15 \cdot 10^8$	2,337
youtube-u-growth	$3.2 \cdot 10^6$	$9.4 \cdot 10^6$	$1.94 \cdot 10^7$	203
dblp-coauthor	$1.3 \cdot 10^6$	$1.8 \cdot 10^7$	$2.4 \cdot 10^9$	70
flickr-growth	$2.3 \cdot 10^6$	$3.3 \cdot 10^7$	$1.7 \cdot 10^7$	134
wikipedia-growth	$1.9 \cdot 10^6$	$4 \cdot 10^7$	$1.93 \cdot 10^8$	2,198

Comparison

When comparing with the algorithms by Wu et al. (2016), we only use the runs with no maximum-waiting-time constraints ($\beta \equiv T$) and we tested all algorithms on the same set of randomly chosen starting vertices. In the experiments, we could only measure a very small effect of the optimization criteria on the running time. This even holds for linear combinations. The only exception was the computation of *foremost* which was a bit faster in comparison to the computation of the other criteria. For this reason we only include two examples here. We chose *foremost* and *shortest* as these are the two criteria where Algorithm 1 performed the best and the worst compared to the algorithms by Wu et al. (2016), respectively. The respective findings are illustrated in the box plots in Fig. 3.

As one can observe in Fig. 3, Algorithm 1 has a larger variance and, therefore, is more dependent on the choice of starting vertices. The reason is that even for $\beta \equiv T$, that is, no maximum waiting time constraints, not all vertices can reach all other vertices by temporal walks in the considered graphs: Algorithm 1 only considers arcs that start in vertices that were already visited and, hence, if many verices are not reached then the algorithm stops. In contrast, the algorithm by Wu et al. (2016) always considers the whole sorted time-arc list and, therefore, has almost no variance in the running time.

The high variance is also the reason why our algorithm has a higher average running time but a comparable median running time: On average, the algorithm by Wu et al. (2016) is 16.4 times faster for computing the *foremost* walk and 7.6 times slower for *shortest* walks. Considering the median running times, however, our algorithm is 5.8 times faster for the *foremost* walk but the algorithm by Wu et al. (2016) is 1.2 times faster for *shortest* walks. The higher average running time is a clear weakness of our algorithm. We believe, however, that the algorithm is still a valuable contribution as it solves more general problems: it can easily combine multiple optimization criteria and it can cope with maximum waiting times and instantaneous arcs, that is, arcs with $\lambda = 0$.

When looking at the time to read the data we can observe that our algorithm takes roughly twice to thrice the time for preprocessing compared to the algorithms by Wu et al. This is due to the fact that for each arc in the input graph Transformation 1 constructs a new vertex and a new arc and so the resulting graph is almost thrice the size. The time to read in the data is much larger than the time of the actual algorithm and so Algorithm 1

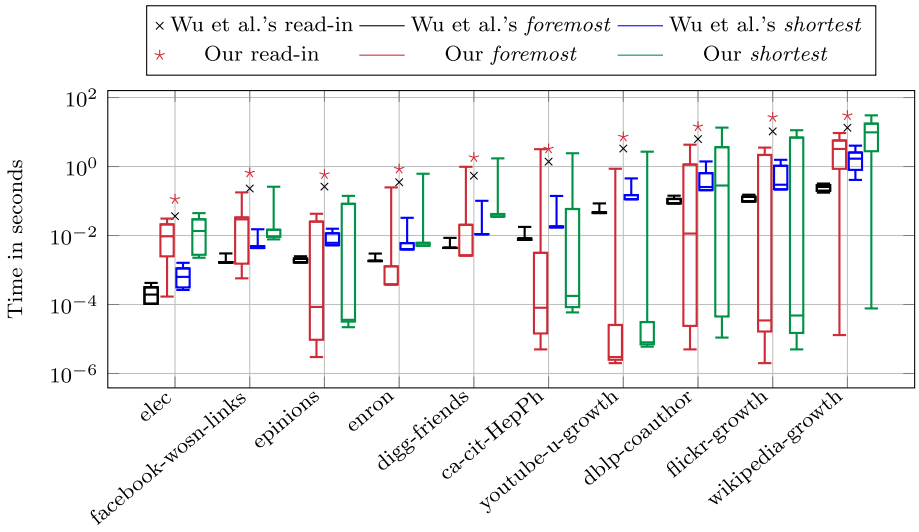


Fig. 3 Running time comparison for computing *foremost* and *shortest* walks. For each graph there are four box plots. From left to right these correspond to the following algorithms: *foremost* of Wu et al. (black), our *foremost* (red), *shortest* of Wu et al. (blue), our *shortest* (green). The boxes represent the 25% to 75% percentile of running times over the 100 runs for different sources on the respective temporal graph and the line within the boxes illustrates the 50% percentile (the median). The whiskers on the top and the bottom represent the best and worst running times, respectively. We only depict the running times of the algorithms after the data has been read in and was preprocessed by Transformation 1. The two plots with the crosses show the running times of reading in the input and preprocessing it

takes roughly thrice the time of the algorithms by Wu et al. (2016) if preprocessing is taken into account.

Finally, we compared the running time of Algorithm 1 with a single optimization criterion against the same algorithm with a linear combination of all criteria considered. Figure 4 displays the average and median running time for $\beta \equiv T$ on all considered data sets.

As expected, the linear combination of optimization criteria does not change the running time compared to a single criterion.

Effect of different maximum waiting-time values β

We next analyze the impact that the maximum-waiting-time constraint β has on Algorithm 1. Decreasing β can have two different effects: First, it can make temporal walks invalid as the maximum allowed waiting time in a vertex is exceeded. Thus, with small β-values certain vertices can only reach few vertices by temporal walks. The second effect is that a temporal walk is invalidated but can be fixed by a detour that starts and ends in the vertex in which the maximal waiting time was exceeded.

We first investigate the second effect. To this end, we partition the optimization criteria into two categories: The first category contains all optimization criteria for which a detour has no negative effect on the solution. These are *foremost*, *reverse-foremost*, *fastest*, and *minimum waiting time*. Since the solution for, e. g., *fastest* is only depending on the first and last time-arc of the temporal walk, adding a cycle somewhere in between does not change the solution. *Minimum waiting time* plays a special role here as this value can decrease when adding intermediate cycles. The second category contains all other optimization criteria, that is, those for which a detour has a negative effect on the solution.

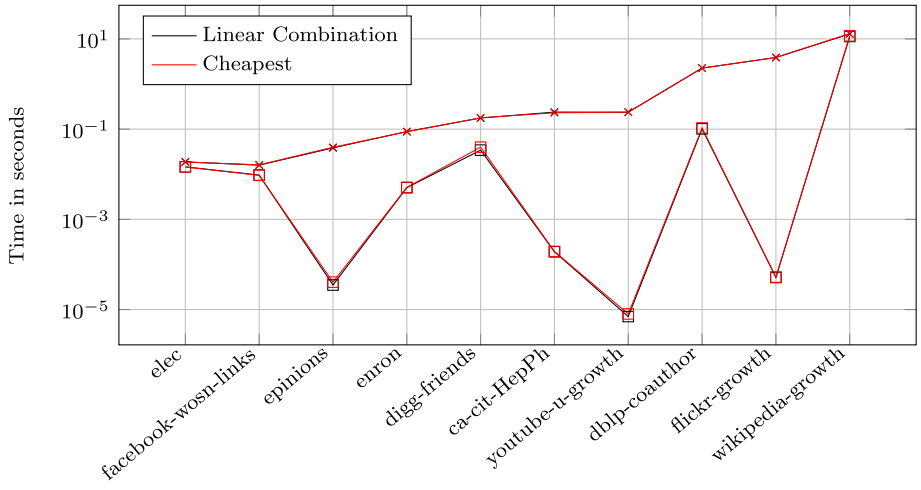


Fig. 4 Running time comparison for computing optimal walks with respect to a linear combination ($\delta_1 = \delta_2 = \ldots = \delta_7 = 1$) of different optimization criteria (black) and with respect to cheapest walks (red) for $\beta \equiv T$. Note that the red and black line are on top of each other and hence almost indistinguishable. The (upper) lines with crosses illustrate the average running time and the (lower) lines with boxes show the median running time

These are *minimum hop count*, *cheapest*, and *shortest*. Since we could not measure significant differences for the different optimization criteria within a category, we only display one figure for each category in Fig. 5.[2]

We remark that in the first category we implemented the algorithm such that irrelevant cycles (cycles which do not change the optimality value) are kept in the solution. Hence Fig. 5 (top left plot) displays values close to the upper bound on the number of cycles in an optimal solution.

Figure 5 (two bottom plots) show that the different categories behave very similarly when it comes to the running-time dependence on the value of β. It seems to be more likely that the first effect we described in the beginning of this subsection (that decreasing β-values can make temporal walks invalid as the maximum allowed waiting time in a vertex is exceeded) is more important for explaining the running times. With very small β-values, a vertex can only reach few other vertices and hence only few time-arcs are considered by Algorithm 1. With increasing β-values, there seems to be a critical value (around $0.1\% - 10\%$ of the lifetime of the temporal graph) where suddenly much more connections appear and hence the running time increases drastically. This observation is affirmed by Fig. 6, which shows that (almost) independently of the input graph, the running time is linearly depending on the number of vertices that are visited.

We believe that the difference for small β-values comes from the initialization which is again more depending on the input graph compared to the algorithms by Wu et al. This would explain, why our algorithms have a higher running time variance.

Conclusion

Building on and widening previous work of Wu et al. (2016), we provided a theoretical and experimental study of computing optimal temporal walks under waiting-time constraints.

[2] We omitted the data sets facebook-wosn-links, flickr-growth, ca-cit-HepPh, and youtube-u-growth in Fig. 5 to keep the figure clear. There is no information gain in displaying these data sets.

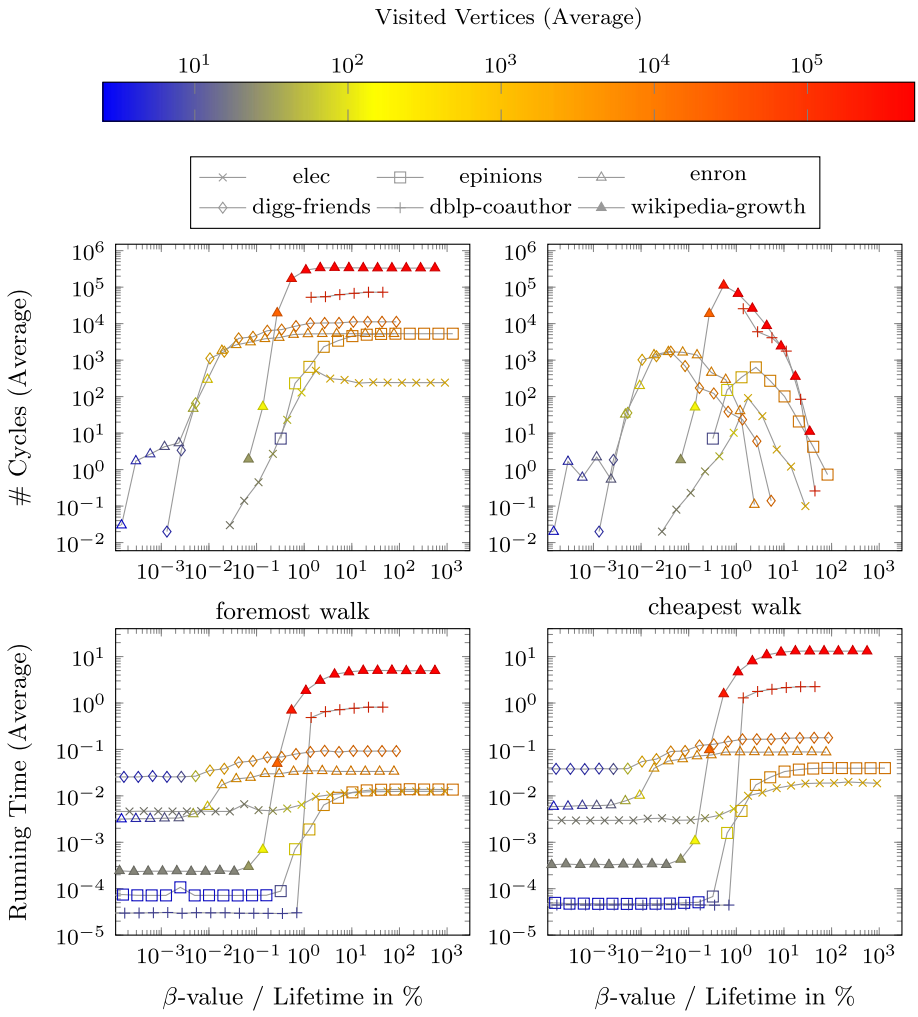


Fig. 5 Impact of different β-values on the number of cycles, the running time, and the number of vertices that can be reached by temporal walks from the chosen starting vertices. Each data point represents 100 runs for a specific β-value for randomly selected starting vertices in the same graph (the same vertices where used for all β-values). The symbol of the data point represents the input graph, the color represents the average number of visited vertices, and the x-coordinate represents the chosen β-value (for the sake of comparability in terms of β-value divided by the graphs lifetime). The y-coordinate represents in the two upper plots the average number of cycles in optimal walks to all reached vertices and in the bottom two plots the average running time for each source vertex. A line connects all data points that use the same graph. The two plots on the left side show results for *foremost* walks and the two plots on the right side show results for *cheapest* walks

The performed experiments indicate the practical relevance of our approach. As to future challenges, recall that moving from walks to paths would yield NP-hard optimization problems (Casteigts et al. 2015). Hence, for the path scenario the study of approximation, fixed-parameter, or heuristic algorithms is a natural next step. For the scenario considered in this work, note that we did not study the natural extension to Pareto-optimal walks (under several optimization criteria). Moreover, for (temporal) network centrality measures based on shortest paths and walks, counting or even listing *all* optimal temporal walks or paths would be of interest. First results in this direction indicate that for several

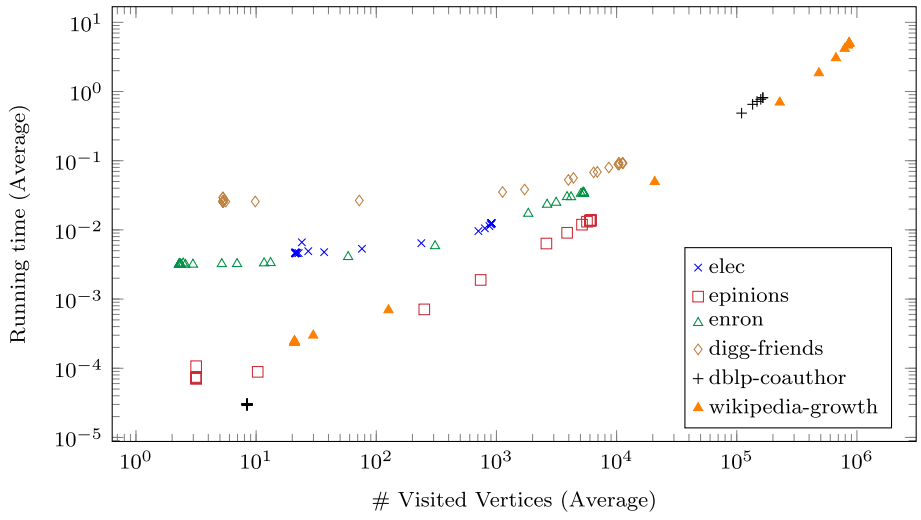


Fig. 6 Average number of visited vertices and its influence on the average running time for *foremost*. Each data point represents 100 runs of our algorithm for *foremost* for one of the considered graphs for a specific β-value. The x-coordinate represents the average number of visited vertices and the y-coordinate indicates the average running time

optimization criteria counting temporal paths is computationally hard even without waiting time constraints; in contrast, counting temporal walks seems easier (Rad et al. 2017; Buß et al. 2020).

Acknowledgements
We thank Lilian Jacobs (TU Berlin) for her programming work helping to enable our experimental studies and the anonymous reviewers of *COMPLEX NETWORKS 2019* and *Applied Network Science* for their constructive feedback.

Authors' contributions
Parts of this work originated from the master thesis of ASH. Overall, all authors have contributed equally to the paper. All authors read and approved the final manuscript.

Funding
ASH was supported by the DFG, project FPTinP (NI 369/16). Open access funding provided by Projekt DEAL.

Availability of data and materials
The data sets used and analyzed during the current study are available in the KONECT library, http://konect.cc/networks/ (KONECT 2017). The open source code of the algorithm is freely available at https://fpt.akt.tu-berlin.de/temporalwalks.

Competing interests
The authors declare that they have no competing interests.

Received: 10 March 2020 Accepted: 24 August 2020
Published online: 06 October 2020

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Upper Saddle River

Axiotis K, Fotakis D (2016) On the size and the approximability of minimum temporally connected subgraphs. In: Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP '16). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Saarbrücken. pp 149–114914

Barabási A-L (2016) Network Science. Cambridge University Press, Cambridge

Bast H, Delling D, Goldberg A, Müller-Hannemann M, Pajor T, Sanders P, Wagner D, Werneck RF (2016) Route planning in transportation networks. In: Algorithm Engineering - Selected Results and Surveys. Lecture Notes in Computer Science. Springer Vol. 9220. pp 19–80

Buß S, Molter H, Niedermeier R, Rymar M (2020) Algorithmic aspects of temporal betweenness. In: Proceedings of the 26th SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20). ACM. pp 2084–2092

Casteigts A, Flocchini P, Godard E, Santoro N, Yamashita M (2015) On the expressivity of time-varying graphs. Theor Comput Sci 590:27–37

Casteigts A, Flocchini P, Quattrociocchi W, Santoro N (2012) Time-varying graphs and dynamic networks. Int J Parallel Emergent Distrib Syst 27(5):387–408
Casteigts A, Himmel A-S, Molter H, Zschoche P (2019) The computational complexity of finding temporal paths under waiting time constraints. arXiv preprint arXiv:1909.06437. To appear at ISAAC '20
Dean BC (2004) Algorithms for minimum-cost paths in time-dependent networks with waiting policies. Networks 44:41–46
Fluschnik T, Molter H, Niedermeier R, Renken M, Zschoche P (2020) Temporal graph classes: A view through temporal separators. Theor Comput Sci 806:197–218
Fluschnik T, Niedermeier R, Schubert C, Zschoche P (2020) Multistage *s-t* path: Confronting similarity with dissimilarity. arXiv preprint arXiv:2002.07569. To appear at ISAAC '20
Himmel A, Bentert M, Nichterlein A, Niedermeier R (2019) Efficient computation of optimal temporal walks under waiting-time constraints. In: Proceedings of the 8th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS '19). Studies in Computational Intelligence. Springer, New York. Vol. 882. pp 494–506
Holme P (2015) Modern temporal network theory: a colloquium. Eur Phys J B 88(9):234
Holme P (2016) Temporal network structures controlling disease spreading. Phys Rev E 94(2):022305
Holme P, Saramäki J (2012) Temporal networks. Physics Reports 519(3):97–125
Holme P, Saramäki J (2013) Temporal networks as a modeling framework. In: Holme P, Saramäki J (eds). Temporal Networks. Springer, New York. pp 1–14
Holme P, Saramäki J (2019) Temporal Network Theory. Springer, New York
Kempe D, Kleinberg J, Kumar A (2002) Connectivity and inference problems for temporal networks. J Comput Syst Sci 64(4):820–842
Kim H, Anderson R (2012) Temporal node centrality in complex networks. Phys Rev E 85(2):026107
Kivelä M, Cambe J, Saramäki J, Karsai M (2018) Mapping temporal-network percolation to weighted, static event graphs. Sci Rep 8(1):12357
KONECT (2017) DNC emails network dataset. Inst Web Sci Technol. http://konect.uni-koblenz.de/networks/dnc-temporalGraph. Accessed 2017
Lightenberg W, Pei Y, Fletcher G, Pechenizkiy M (2018) Tink: A temporal graph analytics library for Apache Flink. In: Proc. of WWW '18. ACM, New York. pp 71–72
Masuda N, Holme P (2013) Predicting and controlling infectious disease epidemics using temporal networks. F1000prime Rep 5
Mertzios GB, Michail O, Spirakis PG (2019) Temporal network optimization subject to connectivity constraints. Algorithmica 81(4):1416–1449
Modiri AB, Karsai M, Kivelä M (2019) Efficient limited time reachability estimation in temporal networks. arXiv preprint arXiv:1908.11831
Newman MEJ (2018) Networks. Oxford University Press, Oxford
Nicosia V, Tang J, Mascolo C, Musolesi M, Russo G, Latora V (2013) Graph metrics for temporal networks. In: Temporal Networks. Springer, New York. pp 15–40
Nicosia V, Tang J, Musolesi M, Russo G, Mascolo C, Latora V (2012) Components in time-varying graphs. Chaos Interdisc J Nonlinear Sci 22(2):023101
Pan RK, Saramäki J (2011) Path lengths, correlations, and centrality in temporal networks. Phys Rev E 84(1):016105
Rad AA, Flocchini P, Gaudet J (2017) Computation and analysis of temporal betweenness in a knowledge mobilization network. Comput Soc Netw 4(1):5
Salathé M, Kazandjieva M, Lee JW, Levis P, Feldman MW, Jones JH (2010) A high-resolution human contact network for infectious disease transmission. Proc Natl Acad Sci 107(51):22020–22025
Santoro N, Quattrociocchi W, Flocchini P, Casteigts A, Amblard F (2011) Time-varying graphs and social network analysis: Temporal indicators and metrics
Wu H, Cheng J, Ke Y, Huang S, Huang Y, Wu H (2016) Efficient algorithms for temporal path computation. IEEE Trans Knowl Data Eng 28(11):2927–2942
Xuan BB, Ferreira A, Jarry A (2003) Computing shortest, fastest, and foremost journeys in dynamic networks. Int J Found Comput Sci 14(02):267–285
Zschoche P, Fluschnik T, Molter H, Niedermeier R (2020) The complexity of finding small separators in temporal graphs. J Comput Syst Sci 107:72–92

Publisher's Note