Technische Universität Berlin

Algorithmic Investigations into Temporal Paths

Masterarbeit

vorgelegt von

Anne-Sophie Himmel

zur Erlangung des Grades „Master of Science“(M. Sc.)
im Studiengang Computer Science

Betreuer:
Matthias Bentert
Dr. André Nichterlein
Prof. Dr. Rolf Niedermeier

Erstgutachter: Prof. Dr. Rolf Niedermeier
Zweitgutachter: Prof. Dr. Markus Brill

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenständige Anfertigung versichert an Eides statt:

Ort, Datum	Unterschrift

Abstract

Shortest paths are a fundamental concept in classical graph theory. Within the realm of temporal graphs—graphs that change over time—the optimization criterion of a *shortest* path is no longer unique as a result of the addition of a time component. We survey various optimization criteria, some well studied in the literature (e.g. *earliest arrival time*), others comparatively neglected thus far (e.g. *minimum waiting time*). We present properties of optimal temporal walks and discuss their algorithmic implications. We study the efficiency of algorithms computing single-source, single-sink, and all-pairs *shortest* path for different variants of temporal graphs and optimization criteria. We also consider parameterized problem variants. In addition to lower bounds, the parameterization by vertex cover number and treewidth of the underlying (classical) graph are examined.

Kurzzusammenfassung

Kürzeste Pfade bilden ein fundamentales Konzept der klassischen Graphentheorie. Im Bereich temporaler Graphen—Graphen die sich über die Zeit verändern—ist das Optimierungskriterium *kürzester* Pfad aufgrund der zusätzlichen Zeitkomponente nicht länger einzigartig. Sowohl in der Literatur fundierte Optimierungskriterien (z.B. früheste Ankunftszeit) als auch bisher vergleichsweise vernachlässigte Kriterien (z.B. minimale Wartezeiten) finden in dieser Arbeit Betrachtung. Eigenschaften von optimalen temporalen Pfaden und ihre algorithmischen Implikationen werden diskutiert. Die Effizienz von Algorithmen für verschiedene Problemvarianten von kürzesten Pfaden (*Single-Source*, *Single-Sink* und *All-Pairs Shortest Paths*) werden untersucht. Dabei werden auch verschiedene temporale Graphenmodelle und Optimierungskriterien betrachtet. Darüber hinaus werden parametrisierte Problemvarianten untersucht. Neben unteren Schranken sind Parametrisierungen anhand von Knotenüberdeckungszahl und Baumweite des unterliegenden klassichen Graphen weiterer Untersuchungsgegenstand.

Contents

1 Introduction

Pandemic spread of an infectious disease is a great threat to global health potentially associated with high mortality rates as well as economic fallout [Sal+10]. Understanding the dynamics of infectious disease spread within human proximity networks could facilitate the development of mitigation strategies. Understanding propagation patterns of diseases also has numerous applications beyond human health. Livestock trade networks offer a fitting example as zoonotic infectious diseases are one of the greatest economical threats in livestock trade [Baj+11; ML+16].

A large part of the legwork required to understand the dynamics of infectious diseases is the analysis of transmission routes through proximity networks [Sal+10]. Classical graph theory can be used to model the main structure of a network: Each person in the network is represented by a vertex and an edge between two vertices indicates at least one proximity contact between these persons. The time component of proximity contacts plays a crucial role in the analysis of transmission routes of a potential disease as shown in the following example:

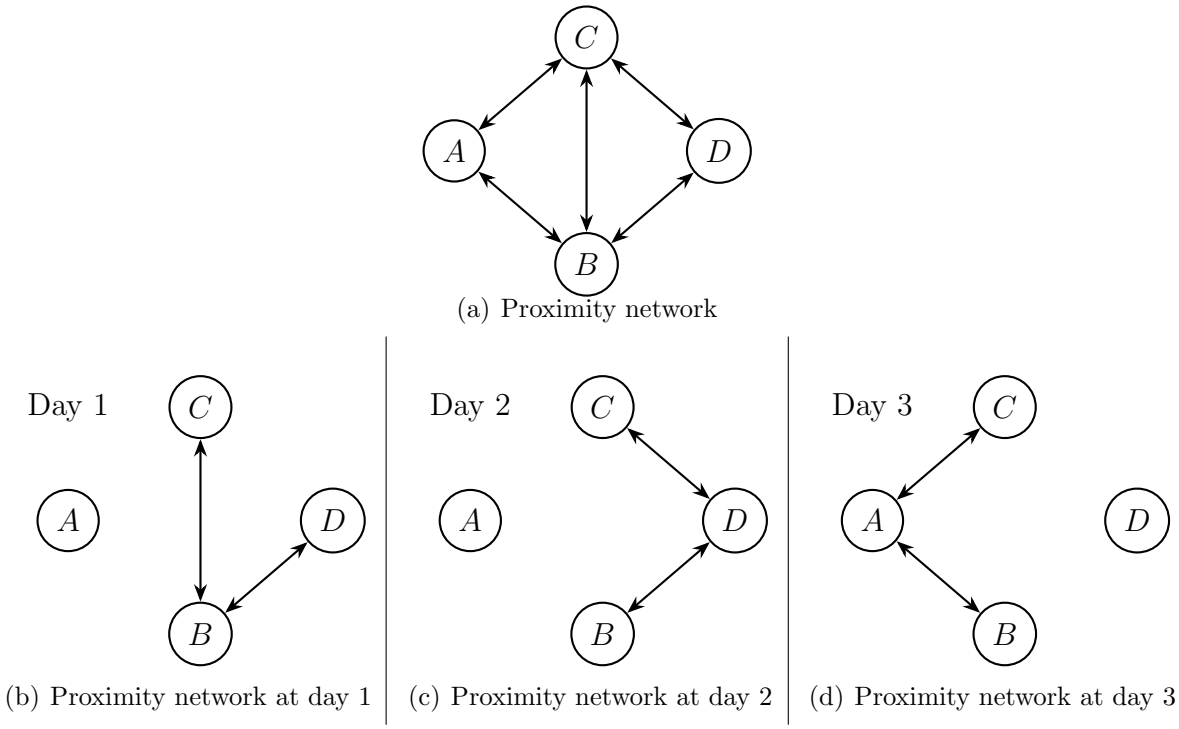
Example 1. If we restrict ourselves to a classical graph model of a proximity network shown in Fig. 1.1(a), then there are several transmission routes from A to D, e.g. $A \to B \to D$ and $A \to C \to D$, by which a disease could have spread. If we extend our model by the moments of proximity contacts in Fig. 1.1(b) to Fig. 1.1(d), then we reach the conclusion that a disease could not have spread from A to D. The proximity contacts $A \to B$ and $A \to C$ occurred on day three whereas the contacts $B \to D$ and $C \to D$ occurred on days one and two. Thus, A could have only infected B and C after proximity contact with D.

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 and the infectious period of the disease starts after one day and ends after the fourth day, then person B could not have infected a person C she met on day ten. Hence, person C could not have been infected by the disease via the transmission route $A \to B \to C$.

One model that is capable of representing both of these properties are temporal graphs.

Temporal Graphs. Temporal graphs are already a frequently used model in the prediction and control of infectious diseases [Hol15; Hol16; MH13]. Temporal graphs—also referred to as temporal networks [HS12; Nic+13], link streams [VLM16], evolving



(a) Proximity network

(b) Proximity network at day 1

(c) Proximity network at day 2

(d) Proximity network at day 3

Figure 1.1: A proximity network modelled as a static graph (Fig. 1.1(a)) and a closer look at the days in which the proximity contacts appear (Figs. 1.1(b) to 1.1(d)).

graphs [XFJ03], or time-varying graphs [Cas+12; San+11]—are graphs that change over time and are therefore capable of capturing the dynamics within a proximity network.

In this thesis, we will consider a fairly general temporal graph model: A temporal graph consists of a *lifetime*, a *set of vertices* and a *set of time-arcs*. A time-arc is a directed edge 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, these vertices exhibit *minimum dwell time* α and *maximum dwell time* β that can reflect the infectious period in our previous example.

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

(1) a time interval $[T]$, where $[T] = \{1, \dots, T\} \subseteq \mathbb{N}$ and T is the lifetime of $\mathcal{G}$,

(2) a vertex set V,

(3) a time-arc set $E \subseteq V \times V \times [T] \times \{0, \dots, T\}$,

(4) a minimum dwell function $\alpha \colon V \to \{0, \dots, T\}$, and

(5) a maximum dwell function $\beta \colon V \to \{0, \dots, T\}$.

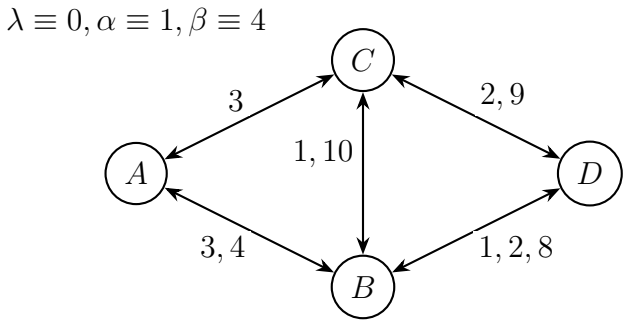


Figure 1.2: A temporal graph with transmission time zero on every time-arc ($\lambda \equiv 0$), a minimum dwell time of one ($\alpha \equiv 1$) and a maximum dwell time of four ($\beta \equiv 4$) in each vertex.

The areas of application of temporal graphs are numerous: in addition to human and animal proximity networks, they are used in communication networks, traffic networks, and distributed computing among others [Hol15; HS12].

Example 3. Fig. 1.2 displays a temporal graph within our proximity contact network on days one to three (Fig. 1.1(b) to Fig. 1.1(d)) plus additional days four to ten. The lifetime T of the temporal graph is ten, the number of days the network was observed. The numbers on the arcs indicate the days on which the contacts occur. For example, the number 3 on the edge between A and B signifies that A and B had proximity contact on day three. Normally, we would write $(3, 0)$ to show that the transmission time is zero. For the sake of simplicity, we do not write the transmission time, if it is constant on every time-arc. The constant transmission time is signaled by $\lambda \equiv 0$. We assume that a person becomes contagious one day after infection and is cured after four days of infection, that is, a minimum dwell time of one and a maximum dwell time of four in each vertex. The constant dwell times are signaled by $\alpha \equiv 1$ and $\beta \equiv 4$ in Fig. 1.2. Note that a period of contagion starting one day after infection could also be represented by a transmission time of one on the time-arcs.

Temporal Walks & Optimal Temporal Walks. Within this model, temporal walks—also referred to as temporal journeys [Mer+13; Nic+13; XFJ03]—are the fundamental concept that implements transmission routes.

A temporal walk is a sequence of time-arcs which connect a sequence of vertices and are non-decreasing in time. In our model, a temporal walk additionally has to fulfill the minimum and maximum dwell time in intermediate vertices.

Definition (Temporal Walk)**.** Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$ and two vertices $v, w \in V$, a *temporal walk* from v to w is a sequence of time-arcs $(e_1, e_2, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ for all $i \in [k-1]$ such that

(1) $v = v_1$ and $w = w_k$,

(2) $w_j = v_{j+1}$, and

(3) $t_j + \lambda_j + \alpha(w_j) \leq t_{j+1} \leq t_j + \lambda_j + \beta(w_j)$ for all $j \in [k-1]$.

The third property signifies that after arriving in a a vertex v in time $t + \lambda$, it is only possible to move on from this vertex earliest $\alpha(v)$ time steps after arrival and at latest $\beta(v)$ time steps after arrival.

Example 4. If we consider the temporal graph in Fig. 1.2, then there is only one valid temporal walk (transmission route) from A to D: $A \overset{4}{\to} B \overset{8}{\to} D$. 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 the route $A \overset{4}{\to} B \overset{10}{\to} C$ as discussed in Example 2. Hence, person A could not have infected C via person B because B was not contagious anymore when she had contact with C.

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 properties can be optimized as a consequence of the introduction of a time variable. Possible properties (with the names we choose) include: arrival time (*Foremost*), departure time (*Reverse-Foremost*), duration (*Fastest*), transmission time (*Shortest*), number of time-arcs (*Minimum Hop-Count*), cost (*Cheapest*), probability (*Most-likely*), and waiting time (*Minimum Waiting Time*). The optimal walks *Foremost*, *Fastest*, and *Minimum Hop-Count* are used quite frequently in the literature [HS12; Nic+13; San+11; XFJ03]. *Shortest* was introduced by Wu et al. [Wu+16]. The remaining properties did not receive much attention in the literature so far. We provide a brief motivation for them all. Not all of them warrant consideration in proximity networks. We will provide examples of the use of all properties that are discarded in the context of proximity networks from their respective fields of application.

Foremost. A foremost walk is a temporal walk from one vertex to another vertex that has the earliest arrival time possible. Computing a foremost walk from a source vertex to all vertices in the proximity network signifies the speed with which an infectious disease could spread. In Fig. 1.2, person A can infect person B and C on day 3, however person D can only be infected on day 8. Consequently, the infectious disease could have permeated the entire system by day 8.

Reverse-Foremost. A reverse-foremost walk is a temporal walk from one vertex to another vertex that exhibits the latest departure time possible. 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. In Fig. 1.2, person A must still be contagious on day 4 to have infected the entire system by the end of the observation period.

Fastest. A fastest walk is a temporal walk from one vertex to another vertex which exhibits the minimum duration, that is, the minimum difference between departure and arrival times. For a proper motivation, we must leave proximity networks and delve into the field of flight networks. Airports represent our vertices, time-arcs flights from one airport to another. The time stamp indicates the departure time of a flight, the transmission time indicates the duration. The minimum dwell time in the vertices signifies the minimum time required in an airport to catch a connecting flight. The duration is the variable passengers aim to minimize in order to streamline their journey.

Shortest. A shortest walk is a temporal walk from one vertex to another vertex 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 from one vertex to another vertex 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.

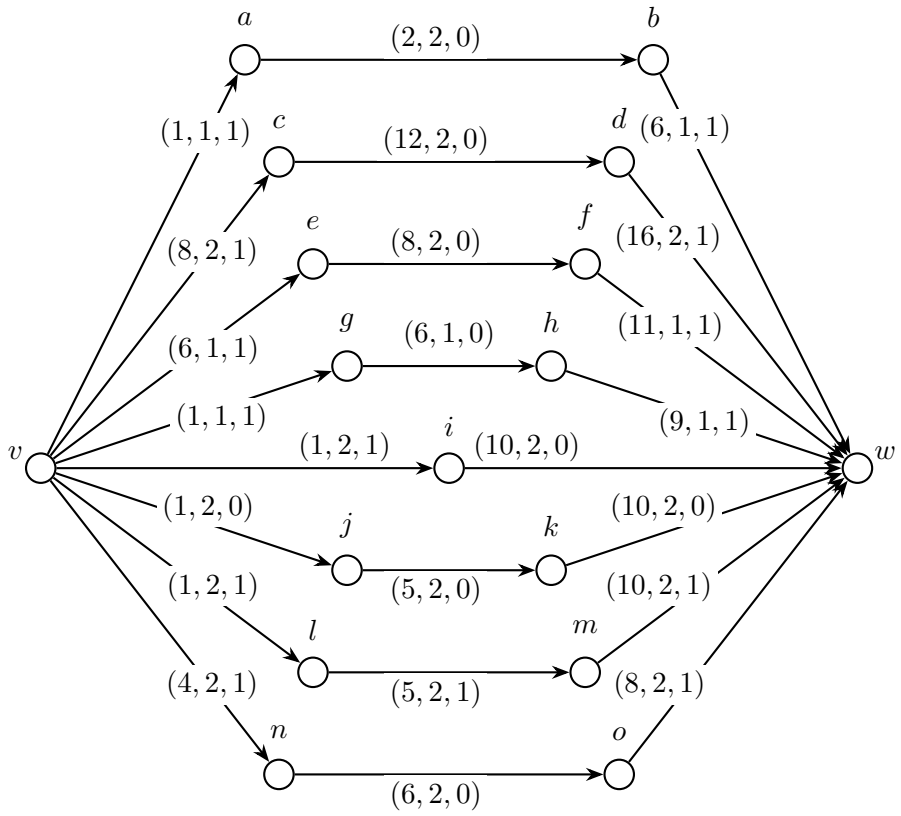
Cheapest. For a given cost function on the time-arcs, a cheapest walk is a temporal walk from one vertex to another vertex with the minimum sum of costs over all time-arcs. The benefits of the minimization of this property within flight networks are obvious: Weighing long travel times and multiple connections 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 from one vertex to another with the highest probability. One application lies in the field of proximity networks: 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. The respective probabilities of the time-arcs within the walk are multiplied.

Minimum Waiting Time. The minimum waiting time walk is a temporal walk from one vertex to another that has minimum sum of waiting times over all intermediate vertices. Routing packets through a router network prioritizes minimum waiting times of packages in the routers to improve the overall performance of the network.

Fig. 1.3 gives an overview of the optimal walks. A formal definition will be given in the Section 2.2.

The focus of this thesis lies in finding such optimal walks within temporal graphs. We elaborate the main structural properties of temporal walks and optimal temporal walks. Based on these properties, we introduce algorithms for finding single-source, single-sink and all-pairs optimal walks. In the end, we discuss the potentials and limitations of



name	optimization property	Example
foremost	earliest arrival time	(v, a, b, w)
reverse-foremost	latest departure time	(v, c, d, w)
fastest	minimum duration	(v, e, f, w)
shortest	minimum travel time	(v, g, h, w)
minimum hop-count	minimum number of time-arcs	(v, i, w)
cheapest	minimum cost	(v, j, k, w)
most-likely	highest probability	(v, l, m, w)
minimum waiting time	waiting time in intermediate vertices	(v, n, o, w)

Figure 1.3: An overview of the optimal walk definitions considered in the thesis. The example-column provides the vertex sequence of an optimal walk in the above displayed temporal graph with $\alpha \equiv 0$ and $\beta \equiv 18$. The three-tuples on the time-arcs represent the time stamp, the transmission time, and potential cost (given by an additional cost function on the time-arcs) respectively.

parameterized problem variants for more efficient algorithms for finding optimal walks in temporal graphs.

Related Work. Temporal walks are a fundamental concept in temporal graph theory. Temporal connectivity is a main concept which is based on the definition of temporal walks [AF16; Ber96; KKK00; Mer+13]. In this context, connected components [Nic+12], flows [KLS02; Sku09] and s-t-separation [KKK00; Zsc+17] are studied. In the most-commonly used model of temporal graphs [HS12; Mer+13], two variants of temporal walks are studied: strict and non-strict temporal walks. Non-strict temporal walks demand non-decreasing time steps on the time-arcs whereas strict temporal walks demand increasing time steps on the time-arcs. The second model can be simulated in our model by adding a transmission time of one on each time-arc within the temporal graph.

A quite common approach in the context of temporal walks is the transformation of a temporal graph into a static graph while maintaining all walks in the temporal graph—often referred to as *line graphs* or *static expansions* [Dea04; KKK00; Mer+13; Wu+16].

Optimal walks are the basis for concepts such as eccentricity, diameter, betweeness and closeness centrality that are widely adapted for temporal graphs [KA12; PS11; San+11; Tan+13]. Nevertheless, the efficiency of computing optimal walks in temporal graphs has received little attention.

Xuan, Ferreira, and Jarry [XFJ03] compute single-source and all-pair optimal walks for *Foremost*, *Fastest* and *Minimum Hop-Count* on time evolving graphs—a restricted class of our temporal graph model in which the lifetime of the graph is bounded in the input size and the transmission times are zero. Their single-source algorithms run in quasi-linear and quadratic time respectively. All-pairs fastest walk is computed in cubic time. One problem is that the running times are depending on the lifetime of the graph which can be exponentially large in comparison to the input size of a temporal graph.

Wu et al. [Wu+16] introduced algorithms for computing single-source optimal walks for *Foremost*, *Reverse-Foremost*, *Fastest*, and *Shortest* on temporal graphs without α- and β-restrictions. The algorithms run in linear and quasi-linear time with respect to the number of time-arcs, provided that transmission times are greater than zero on every time-arc and a sorted time-arcs list is given. A problem is that their algorithms seems to be not adaptable to graphs with arbitrary transmission times without an increase in the running time.

The minimum and maximum dwell time in the vertices have not received much attention in the context of temporal walks even though they are considered as reasonable extensions to the temporal walk model [Hol15; HS12; PS11]. Zschoche et al. [Zsc+17] have studied these α- and β-restrictions for s-t-separation in temporal graphs. Dean [Dea04] studied waiting time policies for finding optimal walks on a restricted temporal graph model, the so-called time-dependent networks, in which the lifetime of the graph is also bounded in the input size.

Structure and Contributions of the Thesis. In the beginning, we want to highlight three key messages of the thesis:

1. Different temporal graph parameters—the transmission times λ and the minimum dwell time α—can be dissolved in linear time while maintaining temporal walks and optimal temporal walks alike.

2. The maximum dwell time has no influence on the running time of finding optimal walks despite the fact that nearly no structural properties of optimal walks hold in temporal graphs with bounded maximum dwell time.

3. There is no deterministic algorithm for finding optimal temporal walks that has a running time that is bounded by a function in the number of vertices in our general temporal graph model.

In more detail, the thesis has the following contributions:

In Chapter 2 we formally define the basic concepts of temporal graphs, temporal walks, and optimal temporal walks. We provide all further terms frequently used throughout the thesis.

In Chapter 3, we elaborate some basic observation concerning the temporal graph parameters λ, α, and β and their impact on temporal walks. We introduce two transformations to dissolve the transmission times and the α-restriction while maintaining all temporal walks in the temporal graph. We present properties of optimal temporal walks in our graph model, followed by observations concerning the structure of optimal walks in temporal graphs without β-restriction.

Chapter 4 is devoted to finding optimal walks in temporal graphs. In Section 4.1, algorithms solving Single Source Optimal Walk are presented. Two algorithms computing *Foremost* and *Fastest* in $O(M)$ and $O(M \log M)$ time respectively are developed where M is the number of time-arcs. The second algorithm is exemplary for the remaining optimal walk definitions.

We further introduce Transformation 3 to transform a temporal graph to a static graph while maintaining all temporal walks in the graph. Optimality values of the walks are preserved by cost functions on the arcs.

Section 4.2 considers the problem of solving Single Sink Optimal Walk. We introduce a transformation that 'reverses' a temporal graph such that solving single-source optimal walk gives a solution for single-sink optimal walk.

In Section 4.3, we introduce an adaptation of the Floyd-Warshall algorithm for solving All-Pairs Optimal Walk on temporal graphs without β-restrictions. This can be solved in $O(n^2M)$ time for *Foremost*, *Reverse-Foremost*, and *Fastest*, and in $O(M^3)$ time for the remaining optimal walk variants (except for *Minimum Waiting Time*) where n is the number of vertices and M is the number of time-arcs in the temporal graph. Not only do the algorithms compute an optimal walk between all pairs of vertices, but also within any possible time interval of the lifetime of a temporal graph. The main results of Chapter 4 are summarized and referenced in Table 1.1.

Table 1.1: Main results of the thesis concerning the problem variants single-source (SSOP), single-sink (SSIOP), and all-pairs (APOP) optimal walk. The variable n denotes the number of vertices and the variable M denotes the number of time-arcs.

Problem		Running Time	Reference
SSOP	(*Foremost*)	$O(M)$	Algorithm 1
		$O(M \log M)$	Algorithm 2
SSIOP	(*Reverse-Foremost*)	$O(M)$	Transformation 4
		$O(M \log M)$	
APOP	(*Foremost, Reverse-Foremost, Fastest)*	$O(n^2 M)$	Algorithm 3
		$O(M^3)$	Algorithm 4

In Chapter 5 we discuss the potentials and limitations of fixed-parameter algorithms for finding optimal walks in temporal graphs and introduce two parameterized algorithms for all-pair optimal walk. First, we show in Section 5.1 that there cannot exist a deterministic algorithm that has a running time that is upper bounded only in the number of vertices in the graph for finding optimal walks in temporal graphs. This observation is followed by two algorithms solving all-pairs optimal walk parameterized by the vertex cover number in Section 5.2.1 and treewidth in Section 5.2.2 of the underlying, undirected, static graph respectively.

Chapter 6 concludes the thesis with a summary of our main results and finishes with an outlook on potential research directions.

2 Preliminaries

In this chapter, we formally introduce the main concepts of the thesis: temporal graphs, temporal walks, and optimal temporal walks. We introduce terms and definitions frequently used throughout the thesis.

2.1 Temporal Graphs

A temporal graph is a graph that changes over time. We briefly repeat the formal definition of a temporal graph introduced in Chapter 1:

Definition 2.1.1 (Temporal Graph)**.** A *temporal graph* $\mathcal{G} = (V, E, [T], \alpha, \beta)$ is defined as a five-tuple consisting of

(1) a time interval $[T]$, where $[T] = \{1, \dots, T\} \subseteq \mathbb{N}$ and T is the *lifetime* of $\mathcal{G}$,

(2) a *vertex set* V,

(3) a *time-arc set* $E \subseteq V \times V \times [T] \times \{0, \dots, T\}$,

(4) a *minimum dwell function* $\alpha \colon V \to \{0, \dots, T\}$, and

(5) a *maximum dwell function* $\beta \colon V \to \{0, \dots, T\}$.

A time-arc $e = (v, w, t, \lambda) \in E$ is a directed connection between v and 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 *arrival time* in vertex w sums up to $t + \lambda$. Furthermore, we call v the *startpoint* and w the *endpoint* of the time-arc.

Example. Given a time-arc $e = (v, w, 4, 5)$, the transmission starts at time step 4 from startpoint v, crosses the arc for 5 time steps, and arrives at time $4 + 5 = 9$ at the endpoint w. We assume that for all time-arcs $e = (v, w, t, \lambda) \in E$ we have $t + \lambda \in T$ so that the arrival time in an endpoint lies within the lifetime of the temporal graph.

To extract the single elements of a time-arc $e = (v, w, t, \lambda)$, we write $\mathrm{start}(e)$ for the startpoint v, $\mathrm{end}(e)$ for the endpoint w, $\mathrm{t}(e)$ for the time step t, and $\lambda(e)$ for the transmission time λ of time-arc e.

Furthermore, there are the α- and β-restrictions on the waiting time in the vertices: The two dwell functions $\alpha \colon V \to \mathbb{N}$ and $\beta \colon V \to \mathbb{N}$ assign each vertex a minimum and maximum dwell time respectively. The minimum dwell time α is the minimum time an agent has to stay in the vertex before she can move further in the temporal graph. The

maximum dwell time β is the maximum time an agent can stay in the vertex before she is no longer allowed to move further in the graph.

Example. Given a vertex v with $\alpha(v) = 2$ and $\beta(v) = 6$, let us assume an agent arrives in the vertex v at time step 10. She cannot move to another vertex before time step $10+2 = 12$. Moreover, she cannot move to another vertex after time step $10 + 6 = 16$ anymore.

If there are no α- and β-restrictions specifically mentioned, then we assume that for every vertex $v \in V$ it holds that $\alpha(v) = 0$ and $\beta(v) = T$. This means that an agent can remain an arbitrary amount of time in a vertex before she can be move to another vertex in the graph. We write $\alpha \equiv 0$ and $\beta \equiv T$ to imply the constant 0-function and the constant T-function respectively. For simplicity, we write $\mathcal{G} = (V, E, [T], \beta)$ if $\alpha \equiv 0$, and we write $\mathcal{G} = (V, E, [T])$ if $\alpha \equiv 0$ and $\beta \equiv T$. If there is a constant $c \in \mathbb{N}$ such that for all $e \in E$ it holds that $\lambda(e) = c$, then we write $\lambda \equiv c$.

In some scenarios, we need to consider time-arc-weighted temporal graphs. A respective cost function $c\colon E \to \mathbb{N}$ assigns each time-arc a value, that is, how much it costs to transmit through the time-arc. Depending on the situation, the costs can also be interpreted as probabilities. If there is no cost function given, then it can be assumed that all time-arcs have a unit value of one.

Induced Temporal Graphs & Underlying Graphs. One concept we will often use through out the thesis is the concept of induced temporal graphs. A temporal graph can be induced by a set of vertices $V' \subseteq V$ or by a time interval $T' \subseteq T$:

Definition 2.1.2 (Induced Temporal Graph)**.** Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph and let $V' \subseteq V$, $T' \subseteq T$:

- The temporal graph $\mathcal{G}[V'] = (V', E', T, \alpha', \beta')$ with
  (1) $E' = \{(v, w, t, \lambda) \in E \mid v, w \in V'\}$,
  (2) $\alpha' : V' \to \mathbb{N}$ with $\alpha'(v) = \alpha(v)$ for all $v \in V'$, and
  (3) $\beta' : V' \to \mathbb{N}$ with $\beta'(v) = \beta(v)$ for all $v \in V'$.
  is the *temporal graph induced by the vertex set* V'.
- The temporal graph $\mathcal{G}[T'] = (V, E', T', \alpha, \beta)$ with
  $$E' = \{(v, w, t, \lambda) \in E \mid t \in T' \wedge t + \lambda \in T'\}.$$
  is the *temporal graph induced by the time interval* T'.

The underlying graph is the static graph that results by ignoring the time component of the time-arcs. We distinguish between directed and undirected underlying graphs:

Definition 2.1.3 (Underlying Graph)**.** Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph.

- The *underlying, directed graph* of $\mathcal{G}$ is defined as $G_d[\mathcal{G}] = (V, E')$ with
  $$E' = \{(v, w) \mid (v, w, t, \lambda) \in E\}.$$
- The *underlying, undirected graph* of $\mathcal{G}$ is defined as $G_u[\mathcal{G}] = (V, E')$ with
  $$E' = \{\{v, w\} \mid (v, w, t, \lambda) \in E \vee (w, v, t, \lambda) \in E\}.$$

Temporal Graph Input. In our algorithmic investigations, we will assume that the time-arcs are sorted by time stamp. This is a legitimate assumption because most temporal graph datasets come with this kind of ordering—see the datasets presented in Barrat and Fournet [BF14], Gemmetto, Barrat, and Cattuto [GBC14], Goerke [Goe11], Isella et al. [Ise+11], Stehlé et al. [Ste+11], and Vanhems et al. [Van+13].

Definition 2.1.4 (Sorted Time-Arc List)**.** Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph, and $[e_1, e_2, \ldots, e_{|M|}]$ be a list of all time-arcs in E. We call $[e_1, e_2, \ldots, e_{|M|}]$ a *sorted time-arc list* if $t(e_1) \leq t(e_2) \leq \cdots \leq t(e_{|M|})$.

If we do not assume that such a sorted time-arc list is given, then we can observe the following: There exist temporal graph instances such that a temporal path consists of all time-arcs sorted by time stamp. Thus, the running-time lower bound of sorting the time-arcs by time stamps—$\Omega(M \log M)$—is a lower bound on finding a temporal walk within the graph.

Reference Guide. As an overview of the major concept of temporal graphs, we introduce and repeat a number of notations and definitions that we use throughout the thesis. This table should not been seen as a subject of study but more as a reference guide. In the context of a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, we denote by

V	the vertex set of $\mathcal{G}$;
E	the time-arc set of $\mathcal{G}$;
$[T]$	the time interval of $\mathcal{G}$ with $[T] = \{1, \ldots, T\}$;
$\mathrm{start}(e)$	the startpoint v of the time-arc $e := (v, w, t, \lambda) \in E$;
$\mathrm{end}(e)$	the endpoint w of the time-arc $e := (v, w, t, \lambda) \in E$;
$\mathrm{t}(e)$	the time step t of the time-arc $e := (v, w, t, \lambda) \in E$;
$\lambda(e)$	the transmission time λ of the time-arc $e := (v, w, t, \lambda) \in E$;
$\lambda_{\max}$	the maximum transmission time in $\mathcal{G}$, that is, $\lambda_{\max} := \max_{(v,w,t,\lambda)\in E} \lambda$;
α	the minimum dwell with $\alpha : V \to \mathbb{N}$;
β	the maximum dwell function with $\beta : V \to \mathbb{N}$;
n	the number $\lvert V \rvert$ of vertices in $\mathcal{G}$;
M	the number $\lvert E \rvert$ of time-arcs in $\mathcal{G}$;
M_t	the number of time-arcs in $\mathcal{G}$ with time stamp smaller or equal to t, that is, $M_t := \lvert\{(v, w, t', \lambda) \in E \mid t' \leq t\}\rvert$;

V_t the vertex subset $V_t \subseteq V$ at time t, that is, $V_t := \{v \mid (v, w, t, \lambda) \in E \vee (w, v, t, \lambda) \in E\}$;

E_t the arc subset at time t, that is, $E_t := \{(v, w) \mid (v, w, t, \lambda) \in E\}$;

G_t the directed, static graph $G_t := (V_t, E_t)$;

n_t the number $|V_t|$ of vertices in G_t;

m_t the number $|E_t|$ of arcs in G_t;

$\mathcal{G}[V']$ the temporal graph of $\mathcal{G}$ induced by the vertex set V' where $V' \subseteq V$;

$\mathcal{G}[T']$ the temporal graph of $\mathcal{G}$ induced by the time interval T' where $T' \subseteq T$;

$G_u[\mathcal{G}]$ the underlying, undirected graph of $\mathcal{G}$;

$G_d[\mathcal{G}]$ the underlying, directed graph of $\mathcal{G}$;

τ the set of all time-steps in which there exists at least one time-arc, that is, $\tau := \{t \mid (v, w, t, \lambda) \in E\}$;

τ_v^+ the set of all time steps in which the vertex v has an out-going arc, formally, $\tau_v^+ := \{t \mid (v, w, t, \lambda) \in E\}$;

$\tau_{\max}^+$ the maximum number of time steps in which a vertex has an out-going arc among all vertices in V, formally, $\tau_{\max}^+ := \max_{v \in V} |\tau_v^+|$;

τ_v^- the set of all time steps in which the vertex v has an in-going arc, formally, $\tau_v^- := \{t + \lambda \mid (w, v, t, \lambda) \in E\}$;

$\tau_{\max}^-$ the maximum number of time steps in which a vertex has an in-going arc among all vertices in V, formally, $\tau_{\max}^- := \max_{v \in V} |\tau_v^-|$;

d_v^+ the set of all time-arcs which start in vertex v, formally, $d_v^+ := \{(v, w, t, \lambda) \in E\}$;

d_v^- the set of all time-arcs which end in vertex v, formally, $d_v^- := \{(w, v, t, \lambda) \in E\}$;

$d_{v,w}$ the set of all time-arcs which start in vertex v and end in vertex w, formally, $d_{v,w} := \{(v, w, t, \lambda) \in E\}$.

2.2 Temporal Walks & Optimal Walks

A temporal walk is a walk in a temporal graph that respects time, that is, the visited time-arcs of a temporal walk have to be increasing in time. Additionally, the transmission time and the α- and β-restrictions have to be respected.

Definition 2.2.1 (Temporal Walk)**.** Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$ and two vertices $v, w \in V$, a *temporal walk* from v to w is a sequence of time-arcs $(e_1, e_2, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ for all $i \in [k]$ such that

(1) $v = v_1$ and $w = w_k$,

(2) $w_j = v_{j+1}$, and

(3) $t_j + \lambda_j + \alpha(w_j) \leq t_{j+1} \leq t_j + \lambda_j + \beta(w_j)$ for all $j \in [k-1]$.

The *length* $|\cdot|$ of a temporal walk $(e_1, e_2, \ldots, e_k)$ is $|(e_1, e_2, \ldots, e_k)| := k$, the number of time-arcs in the sequence. If there is a temporal walk from v to w, then v *reaches* w.

In the literature, temporal walks are often referred to as temporal journey [Mer+13; Nic+13; XFJ03]. Also notice that the vertices do not have to be distinct in a temporal walk. Therefore, we introduce the term *temporal path*. A temporal path is a temporal walk where all vertices are pairwise distinct.

Definition 2.2.2 (Temporal Path)**.** Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$ and two vertices $v, w \in V$, a *temporal path* from v to w is a temporal walk $(e_1, e_2, \ldots, e_k)$ where $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ for all $i \in [k]$ such that

(1) $v_i \neq v_j$ for all $j, i \in [k]$ with $i \neq j$, and

(2) $v_1 \neq w_k$.

We will often talk about subsequences of temporal walks:

Definition 2.2.3 (Subwalk & Subpath)**.** Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph and $P = (e_1, e_2, \ldots, e_k)$ a temporal walk in $\mathcal{G}$. Every temporal walk $P_{i,j} = (e_i, e_{i+1}, \ldots, e_j)$ with $i, j \in [k]$ is called a *subwalk* of P. If $i = 1$, then we call $P_{i,j}$ a *prefix-walk*. If $j = k$, then we call $P_{i,j}$ a *postfix-walk*. If $P_{i,j}$ is a temporal path, then we call $P_{i,j}$ a *subpath* of P.

Note that every subwalk of a temporal path is also a path.

Optimal Temporal Walk. Due to the additional time component, the definition of an optimal walk in temporal graphs can be done in several ways. Different definitions of optimal temporal walks were first provided by Xuan, Ferreira, and Jarry [XFJ03] and extended by Santoro et al. [San+11] and Wu et al. [Wu+16]. In this section, we list the definitions that we have found in the literature and introduce some new definitions that we consider as useful. We already introduced our optimal walk definitions informally in Chapter 1. Now, we provide a formal definition of optimal walks:

Definition 2.2.4 (Optimal Temporal Walk)**.** Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph, $c\colon E \to \mathbb{N}$ be a cost function, and $v, w \in V$ be two vertices. A temporal walk $P = (e_1, e_2, \ldots, e_k)$ from v to w with $e_i = (v_i, w_i, t_i, \lambda_i)$ for all $i \in [k]$ is called an *optimal temporal walk* if it optimizes the following condition (among all temporal walks from v to w):

name	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	$\sum_{i=1}^{k} c(e_i)$
most-likely	max	$\prod_{i=1}^{k} c(e_i)$
minimum waiting time	min	$\sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i)$

An *optimal temporal path* is an optimal temporal walk that fulfills the properties of a temporal path, that is, all visited vertices are pairwise distinct.

For the sake of simplification, we often refer to optimal temporal walks as optimal walks. Note that the definition of most-likely walk can easily be transformed into cheapest walk. First it holds that

$$\prod_{i=1}^{k} c(e_i) \sim \sum_{i=1}^{k} \log c(e_i)$$

In most-likely walks we only want to look at probabilities, implying $c(e) \in [0, 1]$ for all $e \in E$. Hence, we can transform a most-likely walk into a cheapest walk by considering the costs $\log\big(1 - c(e)\big)$ on the time-arcs. We will therefore not consider most-likely walks separately.

Also note that if we are looking into temporal graphs with transmission times zero, then some of the optimal walk variants are equivalent and some are useless:

Shortest. Any temporal from a vertex v to a vertex w is a shortest walk because the sum of transmission times is always zero. Thus, this definition does not make sense in temporal graphs where all transmission times are zero.

Fastest & Minimum Waiting Time. The optimization value of Fastest can be rewritten as the sum of all transmission times plus the sum of all waiting times

$$\sum_{i=1}^{k} \lambda_i + \sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i).$$

If the transmission times are zero, then the first term is set to zero in every temporal walk.

2.3 Problem Definitions

We will give formal definitions of the problem variants of finding optimal walks in temporal graphs that are used throughout the thesis. All these problems can be used for all of our optimal walk definitions.

The first problem is finding an optimal walk from a source vertex to each vertex in the temporal graph.

Single Source Optimal Walk
Input: Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, a cost function $c\colon E \to \mathbb{N}$, and a vertex $v \in V$.
Task: Find an optimal walk from v to every $w \in V$.

The next problem is finding an optimal walk form each vertex in the temporal graph to a distinct sink vertex:

Single Sink Optimal Walk
Input: Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, a cost function $c\colon E \to \mathbb{N}$, and a vertex $w \in V$.
Task: Find an optimal walk from every $v \in V$ to w.

The last problem is finding an optimal walk between all pairs of vertices in the temporal graph.

All-Pairs Optimal Walk
Input: Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$ and a cost function $c\colon E \to \mathbb{N}$.
Task: Find an optimal walk from every $v \in V$ to every $w \in V$.

We will continue in the next chapter by elaborating different temporal graph variants and their impact on temporal walks. We also look into various properties of optimal temporal walks before we start studying efficient algorithms solving all three of the above mentioned problem variants of finding optimal temporal walks.

3 Structure of Temporal Walks

This chapter is devoted to the structure of temporal walks. First, we will investigate some parameters of temporal graphs and their impact on the structure of temporal walks. We consider three of these parameter: the minimum dwell time α, the transmission time λ, and the maximum dwell time β. Furthermore, we will show how we can transform a temporal graph so that there are no transmission times and no minimum dwell times while maintaining all temporal walks in the graph. These transformations justify our restricted temporal graph model in the algorithmic investigations of finding optimal walks. Next, we will study various properties of optimal walks. We will refer to these properties frequently in the course of this thesis due to their great influence on optimal walk algorithms.

3.1 Temporal Graph Variants

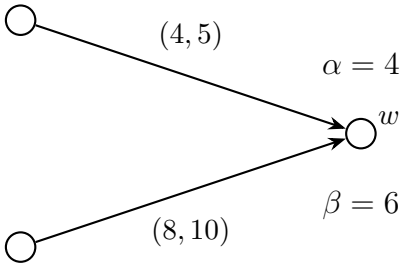
Parameters—the minimum dwell time α, the transmission time λ, and the maximum dwell time β—have great impact on the structure of temporal walks. This includes the existence of cycles or the sequence of time-arcs in a temporal walk. We will elaborate this impact in this section.

In the beginning, we will consider the parameter minimum dwell time. We will introduce a transformation that allows us to transform any temporal graph in linear time into a temporal graph with no minimum dwell time in the vertices. The resulting graph of course maintains all temporal walks. This allows us to restrict the algorithmic investigations to the following temporal graph model— temporal graphs $\mathcal{G} = (V, E, [T], \alpha, \beta)$ with

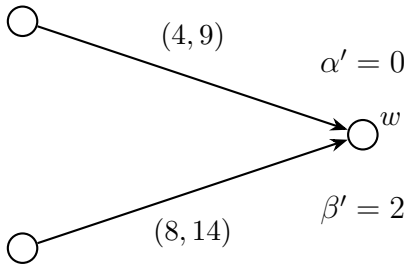
1. $\alpha(v) = 0$ for all $v \in V$,
2. $\lambda(e) \geq 0$ for all $e \in E$, and
3. $\beta(v) \leq T$ for all $v \in V$.

Remark. If it holds that $\lambda(e) = 0$ for all $e \in E$, then we call the temporal graph *instantaneous*. If it holds that $\lambda(e) > 0$ for all $e \in E$, then we call the temporal graph *non-instantaneous*. If it holds that $\beta(v) = T$ for all $v \in V$, then we call the temporal graph *incurable*.

We will continue with showing the impact of the transmission times to temporal walks. We will further introduce a transformation to reduce the transmission time of all time-arcs to zero in linear time while maintaining all temporal walks in the temporal graph. In the end, we will consider the last parameter: the maximum dwell time β. The



(a) Vertex w with $\alpha(w) = 4$



(b) Vertex w with α encoded in the new transmission times λ' and the maximum dwell function β'

Figure 3.1: Removing minimum dwell time α (Transformation 1)

maximum dwell time has a great influence on the structure of temporal walks due to the possible existence of cycles in any temporal walk between two vertices. Consequently, the maximum dwell time plays an important role for the algorithmic efficiency of finding optimal walks.

3.1.1 Minimum Dwell Time α.

Every temporal graph can be transformed into a temporal graph without minimum dwell times in the vertices and at the same time maintaining all temporal walks. The transformation shifts the minimum dwell time to the transmission time of the in-going time-arcs of the vertices: The idea is to stay longer on the time-arc that leads to a vertex instead of being forced to remain in the vertex for the minimum dwell time. The shift and, thus, the removal of the minimum dwell time is illustrated in Figure 3.1.

Transformation 1 (Remove minimum dwell time α). Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph, transform $\mathcal{G}$ into a temporal graph $\mathcal{G}' = (V, E', T, \beta')$ with

(1) $E' = \{(v, w, t, \lambda + \alpha(w)) \mid (v, w, t, \lambda) \in E\}$ and

(2) $\beta' \colon V \to \mathbb{N}$ with $\beta'(v) = \beta(v) - \alpha(v)$ for every $v \in V$.

This transformation of the minimum dwell time runs in $O(n+M)$ time because we only have to look at every vertex and its in-going time-arcs once. The transformed graph $\mathcal{G}'$ has neither an increase in the number of vertices nor in the number of time-arcs and, thus, $\mathcal{G}'$ has n vertices and M time-arcs. The transformation also has no impact on the temporal walks as we show in the following lemma.

Lemma 3.1.1. *Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph and let $\mathcal{G}' = (V, E', [T], \beta')$ be the temporal graph resulting from applying Transformation 1 to $\mathcal{G}$. The time-arc sequence $P = (e_1, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ is a temporal walk in $\mathcal{G}$ if and only if $P' = (e'_1, \ldots, e'_k)$ with $e'_i = (v_i, w_i, t_i, \lambda_i + \alpha(w_i)) \in E'$ is a temporal walk in $\mathcal{G}'$.*

Proof. Let us consider a temporal walk $P = (e_1, \dots, e_k)$ in $\mathcal{G}$ where $e_i = (v_i, w_i, t_i, \lambda_i)$. We know by construction that $e'_i \in E'$ is equal to $(v_i, w_i, t_i, \lambda_i + \alpha(w_i))$ and, consequently, e_i and e'_i have the same startpoint v_i and the same endpoint w_i. It remains to show that the chronological sequence in $P' = (e'_1, \dots, e'_k)$ fulfills the definition of a temporal walk. We know for every $e_i, e_{i+1} \in P$ that

$$\begin{aligned} & t_i + \lambda_i + \alpha(w_i) && \leq\ t_{i+1} \ \leq\ t_i + \lambda_i + \beta(w_i) \\ \Leftrightarrow\ & t_i + (\lambda_i + \alpha(w_i)) && \leq\ t_{i+1} \ \leq\ t_i + (\lambda_i + \alpha(w_i)) + \beta(w_i) - \alpha(w_i) \\ \Leftrightarrow\ & t_i + (\lambda_i + \alpha(w_i)) + 0 && \leq\ t_{i+1} \ \leq\ t_i + (\lambda_i + \alpha(w_i)) + \beta'(w_i). \end{aligned}$$

We can see that the chronological sequence of every $e'_i, e'_{i+1} \in P'$ is therefore also valid. Thus, P' is a temporal walk in $\mathcal{G}'$. The other direction of the equivalence can be shown the same way. □

Remark. Note that all temporal walks from v to w in $\mathcal{G}'$ arrive $\alpha(w)$ time steps later than the original temporal walks in $\mathcal{G}$. Also one has to pay attention when computing a shortest walk or a minimum waiting time walk due to the shift of the minimum waiting time α to the transmission times. For a shortest walk, an easy workaround is to store the actual transmission time of the time-arcs in a cost function on the time-arcs. For a minimum waiting time walk, the minimum dwell time must always be added for any intermediate vertex in a temporal walk. Furthermore, assume we are given a temporal graph with a sorted time-arc list—see Definition 2.1.4. Transformation 1 does not change the time stamps of the arcs. Therefore, the sorted list of the time-arcs can be maintained.

Due to Transformation 1, we can always shift the minimum dwell time of a vertex to the transmission times of the in-going time-arcs in linear time without loosing a potential sorted list of the time-arcs. Thus, it is sufficient to only look at temporal graphs with minimum dwell times zero when investigating the algorithmic aspects of optimal walks. We will do so throughout the thesis.

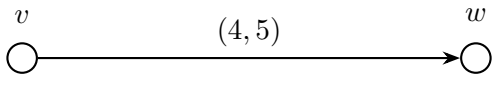
3.1.2 Transmission Time λ.

Looking at the structure of temporal walks, there is a crucial difference between non-instantaneous temporal graphs and those with arbitrary transmission times. In non-instantaneous temporal graphs we know that the time-arcs of a temporal walk have strictly increasing time stamps. Thus, the order in which the time-arcs can appear in a temporal walk is already known.

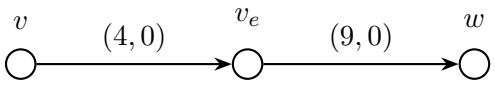
Observation 3.1.2. *Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a non-instantaneous temporal graph. For every temporal walk $(e_1, \dots, e_k)$ with $i_1, \dots, i_k \in [M]$ it holds that*

$$t(e_1) < t(e_2) < \cdots < t(e_k).$$

In contrast to Observation 3.1.2, if we look at general temporal graphs, then a temporal walk within these graphs can consist of a subsequence of time-arcs with the same time stamp.



(a) Time-arc $e = (v, w, 4, 5)$



(b) Splitting time arc e via v_e into $e^s = (v, v_e, 4, 0)$ and $e^a = (v_e, w, 4 + 5, 0)$

Figure 3.2: Removing transmission times λ (Transformation 2)

Removing Transmission Time λ. We can transform every temporal graph with arbitrary transmission times into an equivalent instantaneous temporal graph in terms of temporal walks. We will show that this transformation can be executed in linear time.

The basic idea of the transformation is to encode the transmission time of a time-arc $e = (v, w, t, \lambda)$ by splitting e with a vertex v_e into two time-arcs: the first one $e_1 = (v, v_e, t, 0)$ appearing at time step t and the second one $e_2 = (v_e, w, t + \lambda, 0)$ appearing at time step $t + \lambda$. In this way, we are forced to stay in vertex v_e for the whole transmission time before reaching vertex w. The idea is illustrated in Fig. 3.2. The transmission time of the two newly added time-arcs is set to zero. Formally, the transformation can be described as follows:

Transformation 2 (Remove transmission time λ). Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph, transform $\mathcal{G}$ into a temporal graph $\mathcal{G}' = (V', E', [T], \alpha', \beta')$ with

(1) $V' = V \cup \{v_e \mid e \in E\}$,

(2) $E' = \{(v, v_e, t, 0), (v_e, w, t + \lambda(e), 0) \mid (v, w, t, \lambda) \in E\}$,

(3) $\alpha' \colon V' \to \mathbb{N}$ with

$$\alpha'(v) = \begin{cases} \alpha(v) & \text{for all } v \in V \\ 0 & \text{for all } v \in \{v_e \mid e \in E\}, \text{ and} \end{cases}$$

(4) $\beta' \colon V' \to \mathbb{N}$ with

$$\beta'(v) = \begin{cases} \beta(v) & \text{for all } v \in V \\ T & \text{for all } v \in \{v_e \mid e \in E\}. \end{cases}$$

In Transformation 2, we take all vertices of the original graph and for every time-arc, we add one additional vertex to our vertex set and two predefined time-arcs in our time-arc set. The functions α' and β' must be defined for every vertex in the newly constructed vertex set which contains $n + M$ vertices. Hence, the resulting temporal graph consists of $n + M$ vertices and $2M$ time-arcs and can be constructed in $O(n + M)$ time. Furthermore, the transformation has no influence on the existing temporal walks in the graph:

Lemma 3.1.3. *Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph and $\mathcal{G}' = (V', E', [T], \alpha', \beta')$ be the temporal graph resulting from applying Transformation 2 to $\mathcal{G}$. Given two vertices $v, w \in V$, the time-arc sequence $P = (e_1, \dots, e_k)$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ is a temporal walk from v to w in $\mathcal{G}$ if and only if $P' = (e_1^s, e_1^a, \dots, e_k^s, e_k^a)$ with $e_i^s = (v_i, v_{e_i}, t_i, 0) \in E'$, $e_i^a = (v_{e_i}, w_i, t_i + \lambda_i, 0) \in E'$ is a temporal walk from v to w in $\mathcal{G}'$.*

Proof. Let $P = (e_1, \dots, e_k)$ be a temporal walk from v to w with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$. By Transformation 2, it holds that $e_i^s = (v_i, v_{e_i}, t_i, 0) \in E'$ and $e_i^a = (v_{e_i}, w_i, t_i + \lambda_i, 0) \in E'$. It is easy to verify that the time-arc sequence (e_i^s, e_i^a) is a valid temporal walk due to $\alpha(v_{e_i}) = 0$ and $\beta(v_{e_i}) = T$. Furthermore, we know for every $e_i, e_{i+1} \in P$ that

$$\begin{aligned}
& t(e_i) + \lambda(e_i) + \alpha(\mathrm{end}(e_i)) \leq t(e_{i+1}) \leq t(e_i) + \lambda(e_i) + \beta(\mathrm{end}(e_i)) \\
\Leftrightarrow\ & t_i + \lambda_i + \alpha(w_i) \leq t_{i+1} \leq t_i + \lambda_i + \beta(w_i) \\
\Leftrightarrow\ & t(e_i^a) + \alpha(\mathrm{end}(e_i^a))) \leq t(e_{i+1}^s) \leq t(e_i^a) + \beta(\mathrm{end}(e_i^a))
\end{aligned}$$

It further holds that $\mathrm{end}(e_i^a) = \mathrm{start}(e_{i+1}^s)$ if and only if $\mathrm{end}(e_i) = \mathrm{start}(e_{i+1})$. Thus, the sequence (e_i^a, e_{i+1}^s) is a temporal walk in $\mathcal{G}'$ if and only the sequence (e_i, e_{i+1}) is a temporal walk in $\mathcal{G}$.

We can conclude that every time-arc sequence $P = (e_1, \dots, e_k)$ in $\mathcal{G}$ is a temporal walk if and only if $P' = (e_1^s, e_1^a, \dots, e_k^s, e_k^a)$ is a temporal walk in $\mathcal{G}'$. □

Remark. Note that the transmission time of an original time-arc e is now encoded in the unique dwell time in the vertex v_e within a temporal walk. Also only the waiting times in the original vertices shall count for a minimum waiting time walk.

There is a useful observation that can be made if Transformation 2 is applied to a non-instantaneous temporal graph:

Observation 3.1.4. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a non-instantaneous temporal graph. After applying Transformation 2 to $\mathcal{G}$, for the resulting graph $\mathcal{G}'$ the following holds: The static graph G_t is acyclic for all $t \in [T]$.*

Recall that G_t is the static graph induced by all time-arcs with time step t. The observation follows directly from Observation 3.1.2. Observation 3.1.4 will be useful because there are certain algorithms—like Dijkstra's algorithm computing shortest path—that run faster on acyclic graphs.

Furthermore, if a sorted time-arc list (see Definition 2.1.4) shall be maintained, then applying Transformation 2 requires additional time:

Lemma 3.1.5. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph. After applying Transformation 2 to $\mathcal{G}$, it takes $O\big(M \log(\min\{\lambda_{\max}, M\})\big)$ time to maintain a sorted time-arc list for the resulting graph $\mathcal{G}'$.*

Proof. Let $[e_1, \dots, e_{|M|}]$ be a sorted time-arc list of $\mathcal{G}$. Let L be a new sorted time-arc list of $\mathcal{G}'$. In the beginning, L is empty. Additionally, we store a sorted list L_t of time steps t' together with a list of time-arcs with time stamp t'. The list L_t is also initially empty. For every time-arc e_i with $i \in [M]$—starting with $e_i := e_1$ and increasing i in each iteration—do the following:

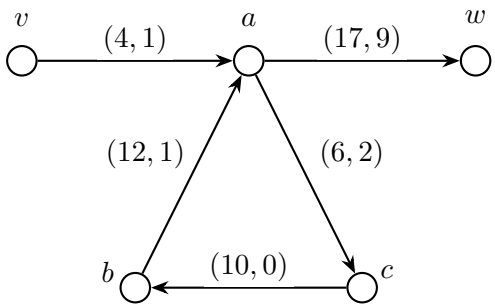


Figure 3.3: A temporal graph with $\beta \equiv 4$ in which the only temporal walk from v to w visits a twice.

1. Delete all time steps in L_t that are smaller than $t(e_{i+1})$ in ascending order and append the corresponding time-arcs to L.
2. Append e_i^s to L.
3. Place $t(e_i^a)$ into L_t, if it does not already exist, and add e_i^a to the time-arc list of time step $t(e_i^a)$.

The list L_t contains at most $\min\{\lambda_{\max}, M\}$ time steps at the same time. Such a sorted list L_t can be implemented with AVL-trees [AVL62]. Thus, all operations—search, insertion, deletion—run in $O(\log \min\{\lambda_{\max}, M\})$ time. Therefore, to get a sorted time-arc list after having applied Transformation 2 takes $O\big(M \log \min\{\lambda_{\max}, M\}\big)$ time. □

In summary, Observation 3.1.2 suggests that temporal walks are easier to find in non-instantaneous temporal graphs. Furthermore, for a temporal graph we introduced Transformation 2 which results in a instantaneous temporal graph. Thus, we mostly consider either instantaneous or non-instantaneous temporal graphs, that is,

1. transmission times always zero ($\lambda(e) = 0$ for all $e \in E$) and
2. transmission times always greater than zero ($\lambda(e) > 0$ for all $e \in E$) respectively.

3.1.3 Maximum Dwell Time β.

A bounded maximum dwell time in the vertices has significant impact on temporal walks. Given a temporal graph $\mathcal{G} = (V, E, [T], \beta)$ with $\beta(v) < T$ for a vertex $v \in V$, we can be forced to make a detour because we exceeded the maximum dwell time. As a consequence, there can be vertices $v, w \in V$ such that any temporal walk from v to w is not a path. An example is given in in Figure 3.3: The only walk from vertex v to w visits vertex a twice.

Observation 3.1.6. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph. There can exist two vertices $v, w \in V$ such that each temporal walk from v to w visits some vertices at least twice.*

In contrast, in incurable temporal graphs it holds that if there is a temporal walk between two vertices, then there is always a temporal path between these two vertices.

Lemma 3.1.7. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph. If there are two vertices $v, w \in V$ such that there is a temporal walk from v to w, then there always is a temporal path from v to w.*

Proof. Let $P = (e_1, \dots, e_k)$ be a temporal walk from v to w, and let $P_{i,j} = (e_i, \dots, e_j)$ be the longest temporal subwalk of P that starts and ends in the same vertex, that is, $i, j = \operatorname{argmax}_{i,j \in [k]}\{j - i \mid \operatorname{start}(e_i) = \operatorname{end}(e_j)\}$. If such a subwalk does not exist, then P is already a temporal path. Otherwise, $\hat{P} = (e_1, \dots, e_{i-1}, e_{j+1}, \dots, e_k)$ is a temporal walk from v to w because $\operatorname{end}(e_{i-1}) = \operatorname{start}(e_{j+1})$ and by definitions

$$t(e_{i-1}) + \lambda(e_{i-1}) \;\leq\; t(e_i) \;\leq\; t(e_{j+1}).$$

Now, we can apply this procedure recursively to $\hat{P}$ until no more cycles exist. □

Remark. Note that if there are maximum dwell times, then in the proof of Lemma 3.1.7 the maximum dwell time in $\operatorname{start}(e_{j+1})$ might be exceeded within $\hat{P}$. Thus, $\hat{P}$ would not be a valid temporal path.

Because of Lemma 3.1.7, we will sometimes only consider the special case of incurable temporal graphs, temporal graphs with $\beta(v) = T$ for all $v \in V$.

After elaborating the impact of the graph parameters—transmission time λ, minimum dwell time α, and maximum dwell time β—on temporal walks, we will summarize the main take-away for the algorithmic investigations into temporal walks: We showed how to transform in linear time any temporal graph into a temporal graph without minimum dwell time. Thus, throughout this thesis we will only consider temporal graphs without minimum dwell time. Furthermore, we will mostly consider instantaneous temporal graphs with transmission times zero or non-instantaneous temporal graphs with transmission times greater than zero. We introduced Transformation 2 to transform any temporal graph in linear time into an instantaneous temporal graph. We studied graphs with a maximum dwell time which led us to the distinction between temporal graphs in general and incurable temporal graphs ($\beta(v) = T$ for all $v \in V$) in terms of the maximum dwell time.

An overview of the different temporal graph models considered throughout the thesis is provided in Fig. 3.4.

3.2 Optimal Temporal Walks

This section is devoted to optimal temporal walks. We will examine structural properties of optimal walks and briefly discuss their algorithmic impact on finding optimal walks in temporal graphs.

We distinguish between temporal graphs in general and incurable temporal graphs. Note that all structural properties of optimal temporal walks that we consider for temporal graphs in general also hold for incurable temporal graphs. We start elaborating structural properties of optimal walks in all temporal graphs.

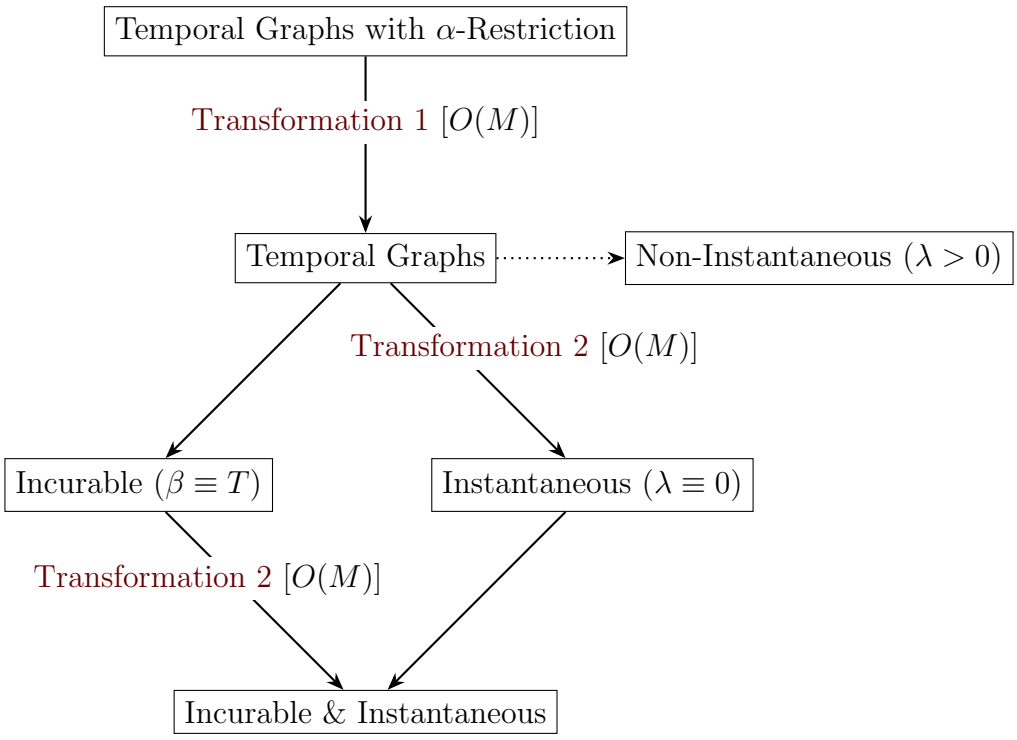


Figure 3.4: An overview of the temporal graph models where M is the number of time-arcs. In $[\cdot]$ the running time of the respective transformation is given.

3.2.1 Temporal Graphs

We will introduce three properties of optimal walks that we consider important for finding optimal walks in temporal graphs: a property for Foremost, one for Reverse-Foremost and one for Fastest.

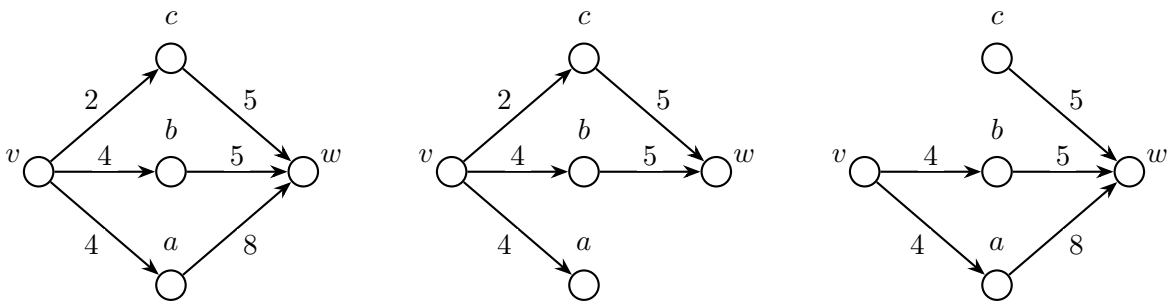
Foremost & Reverse-Foremost. The following two properties concern the set of possible optimal walks: Looking at a foremost walk $P = (e_1, \dots, e_k)$, there is always a foremost walk that is a reverse-foremost walk within time interval $[1, t(e_k) + \lambda(e_k)]$.

Observation 3.2.1. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph, $v, w \in V$ be two vertices, and $P = (e_1, \dots, e_k)$ be a foremost walk from v to w. Then there is a foremost walk that is a reverse-foremost walk from v to w in $\mathcal{G}[[1, t(e_k) + \lambda(e_k)]]$.*

Thus, computing a reverse-foremost walk from v to w for every time interval $[1, t]$ with $t \in [T]$ also returns a foremost walk from v to w.

The reverse statement holds for reverse-foremost walks: Looking at a reverse-foremost walk $P = (e_1, \dots, e_k)$, there is always a reverse-foremost walk that is a foremost walk within time interval $[t(e_1), T]$.

Observation 3.2.2. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph, $v, w \in V$ be two vertices, and $P = (e_1, \dots, e_k)$ be a reverse-foremost walk from v to w. Then there is a reverse-foremost walk that is a foremost walk from v to w in $\mathcal{G}[[t(e_1), T]]$.*



(a) Temporal graph $\mathcal{G}$ (b) $\mathcal{G}[[1,5]]$ (c) $\mathcal{G}[[4,8]]$

Figure 3.5: A temporal graph $\mathcal{G}$ with $\lambda \equiv 0$ and $\beta \equiv 3$ and its induced subgraphs $\mathcal{G}[[1,5]]$ and $\mathcal{G}[[4,8]]$.

Fastest. If $P = (e_1, \ldots, e_k)$ is a fastest walk, then it is a foremost walk within time interval $[t(e_1), T]$. And it is a reverse-foremost walk within time interval $[1, t(e_k)+\lambda(e_k)]$. This has already been shown by Wu et al. [Wu+16, Lemma 8].

Lemma 3.2.3. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph, $v, w \in V$ be two vertices, and $P = (e_1, \ldots, e_k)$ be a fastest walk from v to w. Then the temporal walk P is a foremost walk from v to w in $\mathcal{G}[[t(e_1), T]]$ and a reverse-foremost walk from v to w in $\mathcal{G}[[1, t(e_k) + \lambda(e_k)]]$.*

Proof. Let $P^* = (e_1, \ldots, e_k)$ be a fastest walk from v to w in $\mathcal{G}$. Assume towards a contradiction that P^* is not a foremost walk from v to w in the induced graph $\mathcal{G}[[t(e_1), T]]$. Let P' be a foremost walk from v to w in $\mathcal{G}[[t(e_1), T]$. This means that walk P' starts not before time step $t(e_1)$ and arrives before time step $t(e_k) + \lambda(e_k)$. But then P' would be a faster walk from v to w in $\mathcal{G}$ than the walk P^*—a contradiction.

Assume towards a contradiction that P^* is not a reverse-foremost walk from v to w in $\mathcal{G}[[1, t(e_k) + \lambda(e_k)]]$. Let P' be a reverse-foremost walk from v to w in $\mathcal{G}[[1, t(e_k) + \lambda(e_k)]]$. This means that walk P' starts later than $t(e_1)$ and arrives not later than time step $t(e_k) + \lambda(e_k)$. Hence, P' is a faster walk from v to w in $\mathcal{G}$ than the walk P^*—a contradiction. □

Let us look at a vertex $v \in V$ of a temporal graph with out-going time-arcs at time-steps τ_v^+. Lemma 3.2.3 implies the following: If we compute for all $t \in \tau_v^+$ a foremost walk to a vertex $w \in V$ starting at time step t or later, then one of these foremost walks is a fastest walk from v to w. The same procedure works for reverse-foremost walks.

Example. Fig. 3.5 illustrates Observation 3.2.1, Observation 3.2.2, and Lemma 3.2.3. Let us look at the temporal graph $\mathcal{G}$ in Fig. 3.5(a):

- There are two foremost walks from v to w in the temporal graph $\mathcal{G}$: The temporal walks $P_\mathrm{f} = \big(v, c, 2, 0), (c, w, 5, 0)\big)$ and $P'_\mathrm{f} = \big((v, b, 4, 0), (b, w, 5, 0)\big)$. If we look at $\mathcal{G}[[1,5]]$ in Fig. 3.5(b), then P'_f is a reverse-foremost walk.

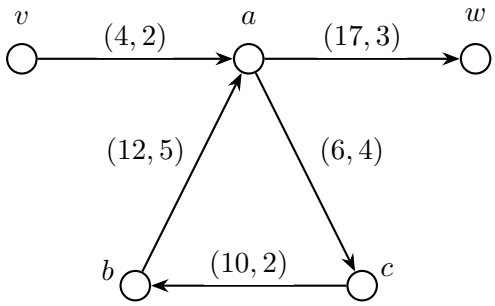


Figure 3.6: An incurable temporal graph in which the waiting time in intermediate vertices from v to w decreases by walking a cycle.

- There are two reverse-foremost walks from v to w in $\mathcal{G}$: The temporal walks $P_{\mathrm{r}} = \big(v, b, 4, 0), (b, w, 5, 0)\big)$ and $P'_{\mathrm{r}} = \big((v, a, 4, 0), (a, w, 8, 0)\big)$. If we look at $\mathcal{G}[[4, 8]]$ in Fig. 3.5(c), then P_{r} is a foremost walk.

- There is one fastest walk from v to w in the temporal graph $\mathcal{G}$: The temporal walk $P_{\mathrm{fa}} = \big(v, b, 4, 0), (b, w, 5, 0)\big)$. If we look at $\mathcal{G}[[1, 5]]$ in Fig. 3.5(b), then P_{fa} is a reverse-foremost walk. If we look at $\mathcal{G}[[4, 8]]$ in Fig. 3.5(c), then P_{fa} is a foremost walk.

The restricted dwell time in the vertices prohibit any statement about the general structure of optimal walks. This changes in incurable temporal graphs. Next, we will consider optimal walks in incurable temporal graphs.

3.2.2 Temporal Graphs with Unbounded Dwell Time

In incurable temporal graphs, we can find additional properties concerning the structure of optimal temporal walks that can be exploited when finding optimal walks.

The most important property is the following: For any optimal temporal walk, we can always find an optimal temporal path. Minimum Waiting Time is an exception since the path constructed in Lemma 3.1.7 does not preserve this optimality criterion.

Observation 3.2.4. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph. There can exist two vertices $v, w \in V$ such that every minimum waiting time walk from v to w visits a vertex at least twice.*

In Fig. 3.6, we can see that running a cycle reduces the sum of waiting times on the walk from a to e. We start with proving the above statement about optimal walks in incurable temporal graphs.

Lemma 3.2.5. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph. If there are two vertices $v, w \in V$ such that there is an optimal temporal walk from v to w, then there always is an optimal temporal path from v to w (except for Minimum Waiting Time).*

Proof. Let us consider an optimal walk $P = (e_1, \dots, e_k)$ from v to w. By Lemma 3.1.7, we already know that we can construct a temporal path from a temporal walk by deleting all longest subwalks $P_{i,j} = (e_i, \dots, e_j)$ where $\text{start}(e_i) = \text{end}(e_j)$ for $i, j \in [k]$. If there is no such subwalk, then we are done. Otherwise, we show that the resulting temporal path P' is optimal. The temporal path P' is contained in the temporal walk P. Also the optimality value only depends on the following criterion:

- *Foremost*: the starting time of the first time-arc in the walk,
- *Reverse-Foremost*: the arrival time of the last time-arc in the walk,
- *Fastest*: the difference between the arrival time of the last time-arc and the starting time of the first time-arc in the walk,
- *Shortest*: the sum of transmission times of the time-arcs in the walk,
- *Minimum Hop-Count*: the sum of time-arcs in the walk, and
- *Cheapest*: the sum of costs of the time-arcs in the walk. Recall that we only consider positive time-arc costs.

The optimality of P cannot be lost by removing time-arcs from the walk P. Thus, the resulting walk P' has to be optimal. □

Lemma 3.2.5 is summarized in Table 3.1. It does not hold in temporal graphs in general as already shown in Observation 3.1.6. We conclude that if there is a walk from a vertex v to a vertex w, then there exists an optimal temporal path (except for Minimum Waiting Time) in incurable temporal graphs. Such a property also holds for shortest walks in static graphs with no negative cycles. Additionally, in static graphs, any subpath of a shortest path is a shortest path. This property makes it easy to solve Single-Source Shortest Path in static graphs. Unfortunately, such a statement does not hold for temporal graphs due to the additional time component [Wu+16]. We are, however, able to make a statement about the subwalks of an optimal temporal walk.

Foremost & Reversed-Foremost. Foremost walks are one of the easiest temporal walks to find, especially in incurable temporal graphs: for any foremost temporal walk, there exists a foremost path such that all prefix-paths are foremost paths as well. This has already been shown by Wu et al. [Wu+16, Lemma 6] and Xuan, Ferreira, and Jarry [XFJ03, Theorem 1].

Lemma 3.2.6. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph and $v, w \in V$ be two vertices. If there is a temporal walk from v to w, then there exists a foremost path $P = (e_1, \dots, e_k)$ from v to w such that for every $i \in [k-1]$ the subpath $P_i = (e_1, \dots, e_i)$ is a foremost path from v to* $\text{end}(e_i)$.

Table 3.1: Given an incurable temporal graph, this table shows for which of our optimal temporal walk definitions it holds that if there is a temporal walk between two vertices, then there always exists an optimal solution that is a path between them.

Optimality Criterion	Optimal walk is a path?
Foremost	✓
Reverse-Foremost	✓
Fastest	✓
Shortest	✓
Minimum Hop-Count	✓
Cheapest	✓
Minimum Waiting Time	×

Proof. Due to Lemma 3.2.5, we know that there exists a foremost temporal path from v to w. Let $P = (e_1, \dots, e_k)$ be a foremost path from v to w and let $i \in [k-1]$ be the largest index such that $P_i = (e_1, \dots, e_i)$ is not a foremost path from v to $\text{end}(e_i)$. If such an i does not exist, then we are done. Otherwise, there is a foremost path $\hat{P} = (\hat{e}_1, \dots, \hat{e}_{i'})$ from v to $\text{end}(e_i)$ that arrives at time-step $t(\hat{e}_{i'}) + \lambda(\hat{e}_{i'}) < t(e_i) + \lambda(e_i)$. But then the temporal path $P^* = (\hat{e}_1, \dots, \hat{e}_{i'}, e_{i+1}, \dots, e_k)$ is a foremost path from v to w as well. □

Example. For a small example, we look at temporal graph $\mathcal{G}$ in Fig. 3.7. Let us look at two different foremost paths from v to e: $P = \big((v, b, 2, 0), (b, d, 6, 0), (d, e, 8, 0)\big)$ and $P' = \big((v, a, 4, 0), (a, b, 5, 0), (b, d, 6, 0), (d, e, 8, 0)\big)$. It holds that the prefix-paths of P from v to b and from v to d are foremost paths. This does not hold for path P'. If we assume that we have bounded dwell time $\beta \equiv 2$, then only P' would be a valid path. This shows that Lemma 3.2.6 only holds in incurable temporal graphs.

Furthermore, we know that every postfix-walk of a foremost walk is a foremost walk within its time interval.

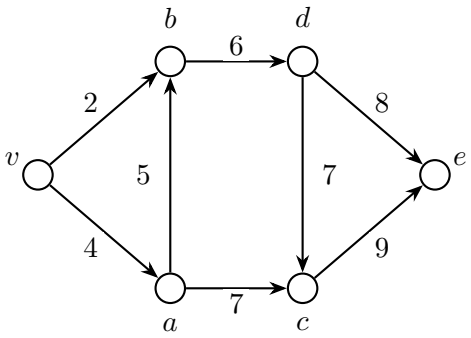


Figure 3.7: An incurable temporal graph $\mathcal{G}$ with $\lambda \equiv 0$.

Lemma 3.2.7. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph and $v, w \in V$ be two vertices. For every foremost walk $P = (e_1, \dots, e_k)$ from v to w it holds that for every $j \in \{2, \dots, k\}$ the subwalk $P_j = (e_j, \dots, e_k)$ is a foremost walk from* $\text{start}(e_j)$ *to w in $\mathcal{G}[[t(e_{j-1}) + \lambda(e_{j-1}), T]]$.*

Proof. Let $P = (e_1, \dots, e_k)$ be a foremost walk from v to w and let $j \in \{2, \dots, k\}$. Assume towards a contradiction that $P_j = (e_j, \dots, e_k)$ is not a foremost walk from v_i to w within time interval $T_j = [t(e_{j-1}) + \lambda(e_{i-1}), T]$. Let $\hat{P} = (\hat{e}_1, \dots, \hat{e}_{j'})$ be a foremost walk from $\text{start}(e_j)$ to w within T_j. It holds that $t(e_{j-1}) + \lambda(e_{j-1}) \leq t(\hat{e}_1)$ and $t(\hat{e}_{j'}) + \lambda(\hat{e}_{j'}) < t(e_k) + \lambda(e_k)$. But then the temporal walk $P^* = (e_1, \dots, e_{j-1}, \hat{e}_1, \dots, \hat{e}_{j'})$ is a temporal walk with an earlier arrival time than P—a contradiction to our assumption that P is a foremost walk. Thus, P_j is a foremost walk within T_j. □

Example. In Fig. 3.7, we can also see that for any foremost path—let us look at $P = \big((v, b, 2, 0), (b, d, 6, 0), (d, e, 8, 0)\big)$—the following holds: the postfix-paths of P from b to e and from d to e are foremost path in $\mathcal{G}[[2, 9]]$ and $\mathcal{G}[[6, 9]]$ respectively, as suggested by Lemma 3.2.7. It does not hold in temporal graphs in general. If we assume that the graph $\mathcal{G}$ in Fig. 3.7 has a bounded dwell time $\beta(d) = 1$, then for the a valid foremost path from v to e—$P' = \big((v, a, 4, 0), (a, b, 5, 0), (b, d, 6, 0), (d, c, 7, 0), (c, e, 9, 0)\big)$—the property does not hold.

One more observation can be made concerning foremost walks. If we have two time-steps t and t' with $t < t'$, then a foremost walk P from a vertex v to a vertex w starting earliest in time step t has an arrival time smaller or equal to a foremost walk P' from v to w starting earliest at time-step t'. If this is not the case, then P is not a foremost path because P' starts also earliest in time t and has an earlier arrival time than P.

Observation 3.2.8. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph, $t, t' \in [T]$ be two time-steps with $t < t'$, and $v, w \in V$ be two vertices. If there is a foremost walk P from v to w in $\mathcal{G}[[t, T]]$ and a foremost walk P' from v to w in $\mathcal{G}[[t', T]]$, then P has an arrival time not greater than P'.*

For reverse-foremost walks, we can make a strong statement about the postfix-walks: for any reverse-foremost walk, there exists a reverse-foremost path such that all postfix-paths are reverse-foremost paths as well.

Lemma 3.2.9. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph and $v, w \in V$ be two vertices. If there is a temporal walk from v to w, then there exists a reverse-foremost path $P = (e_1, \dots, e_k)$ from v to w such that for every $i \in \{2, \dots, k\}$ the temporal path $P_i = (e_i, \dots, e_k)$ is a reverse-foremost path from start(e_i) to w.*

Proof. Due to Lemma 3.2.5, we know that there exists a reverse-foremost temporal path from v to w. Let $P = (e_1, \dots, e_k)$ be this reverse-foremost path from v to w, and let $i \in \{2, \dots, k\}$ be the smallest index such that $P_i = (e_i, \dots, e_k)$ is not a reverse-foremost path from start(e_i) to w. If it does not exist, then we are done. Otherwise, we know that there is a reverse-foremost path $\hat{P} = (\hat{e}_{i'}, \dots, \hat{e}_k)$ from start(e_i) to w that departs at time-step $t(\hat{e}_{i'}) > t(e_i)$. But then the temporal path $P^* = (e_1, \dots, e_{i-1}, \hat{e}_{i'}, \dots, \hat{e}_k)$ is a reverse-foremost path form v to w as well. □

Example. For a small example, we consider again the temporal graph $\mathcal{G}$ in Fig. 3.7. Let us look at two reverse-foremost paths from v to e: $P = \big((v, a, 4, 0), (a, c, 7, 0), (c, e, 9, 0)\big)$ and $P' = \big((v, a, 4, 0), (a, b, 5, 0), (b, d, 6, 0), (d, c, 7, 0), (c, e, 9, 0)\big)$. It holds that the postfix-paths of P from a to e and from c to e are reverse-foremost paths. This does not hold for path P'. If we assume that we have a maximum dwell time $\beta \equiv 2$, then only P' would be a valid path. This shows that Lemma 3.2.9 only holds in incurable temporal graphs. In the walk P it also holds that the prefix-paths from v to a and from v to c are a reverse-foremost path in $\mathcal{G}[[1, 7]]$ and $\mathcal{G}[[1, 9]]$ respectively. This holds in incurable temporal graphs which will be shown in Lemma 3.2.10.

Furthermore, we know that every prefix-walk of a reverse-foremost walk is a reverse-foremost walk within its time interval.

Lemma 3.2.10. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph and let $v, w \in V$ be two vertices. For every reverse-foremost walk $P = (e_1, \dots, e_k)$ from v to w it holds that for every $j \in [k-1]$ the subpath $P_j = (e_1, \dots, e_j)$ is a reverse-foremost path from v to* end(e_j) *in $\mathcal{G}[[1, t(e_{j+1})]]$.*

Proof. Let $P = (e_1, \dots, e_k)$ be a reverse-foremost walk from v to w and let $j \in [k-1]$. Assume towards a contradiction that $P_j = (e_1, \dots, e_j)$ is not a reverse-foremost walk from v to end(e_j) within time interval $T_j = [1, t(e_{j+1})]$. Let $\hat{P} = (\hat{e}_1, \dots, \hat{e}_{j'})$ be a reverse-foremost walk from v to end(e_j) within T_j. It holds that $t(\hat{e}_{j'}) + \lambda(\hat{e}_{j'}) \leq t(e_{j+1})$ and $t(e_1) < t(\hat{e}_1)$. But then the temporal graph $P^* = (\hat{e}_1, \dots, \hat{e}_{j'}, \hat{e}_{j+1}, \dots, e_k)$ is a temporal walk with a departure time later than P—a contradiction to our assumption that P is a reverse-foremost walk. Thus, P_j is a reverse-foremost walk within T_j. □

Shortest, Minimum Hop-Count & Cheapest. For Shortest, Minimum Hop-Count, and Cheapest, we can make a statement about all subwalks of an optimal walk. The basis for their common property is that they all sum up the respective time-arc values of the walk. Shortest sums up the transmission times on the time-arcs, Minimum Hop-Count sums up the number of time-arcs, and Cheapest sums up the costs of the time-arcs in a temporal walk. Each optimal temporal walks $P = (e_1, \dots, e_k)$ from a vertex v to a vertex w holds that each temporal subwalk $P_{i,j} = (e_i, \dots, e_j)$ of P is an optimal walk within its time

boundaries, that is, the time-interval starting at time $t(e_{i-1}) + \lambda(e_{i-1})$ and ending in time $t(e_{j+1})$.

Lemma 3.2.11. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph, and $v, w \in V$ be two vertices. For each optimal walk with respect to Shortest, Minimum Hop-Count, and Cheapest $P = (e_1, \ldots, e_k)$ from v to w it holds that each subwalk $P_{i,j} = (e_i, \ldots, e_j)$ with $i, j \in [k]$ is an optimal temporal walk in $\mathcal{G}[T_{i,j}]$ where*

$$T_{i,j} := \begin{cases} [1, t(e_{j+1})] & \text{if } i = 1, \\ [t(e_{i-1}) + \lambda(e_{i-1}), T] & \text{if } j = k, \\ [t(e_{i-1}) + \lambda(e_{i-1}), t(e_{j+1})] & \text{otherwise.} \end{cases}$$

Proof. Let $P = (e_1, \ldots, e_k)$ be an optimal walk from v to w and let $P_{i,j} = (e_i, \ldots, e_j)$ be a subwalk of P with $i, j \in [k]$. Assume towards a contradiction that $P_{i,j} = (e_i, \ldots, e_j)$ is not an optimal walk within time interval $T_{i,j}$. Thus, there is a walk $\hat{P} = (\hat{e}_1, \ldots, \hat{e}_{k'})$ from start(e_i) to end(e_j) that is optimal within $\mathcal{G}[T_{i,j}]$. It holds:

$$\sum_{l=1}^{k'} d(\hat{e}_l) \;<\; \sum_{l=i}^{j} d(e_l)$$

with $d(e) := \lambda(e)$ for Shortest, $d(e) := 1$ for Minimum Hop-Count, and $d(e) := c(e)$ for Cheapest for all $e \in E$.

But then the temporal walk $P^* = (e_1, \ldots, e_{i-1}, \hat{e}_{1'}, \ldots, \hat{e}_{k'}, e_{j+1}, \ldots, e_k)$ is preferred over P since the replacement of $P_{i,j}$ by $\hat{P}$ reduces the overall value of the temporal walk:

$$\sum_{l=1}^{i-1} d(e_l) + \sum_{l=1}^{k'} d(\hat{e}_i) + \sum_{i=j+1}^{k} d(e_i) \;<\; \sum_{l=1}^{i-1} \lambda(e_l) + \sum_{l=i}^{j} d(e_l) + \sum_{l=j+1}^{k} d(e_l) \;=\; \sum_{l=1}^{k} d(e_l)$$

This is a contradiction to our assumption that P is optimal. Thus, any subwalk $P_{i,j}$ is an optimal walk within $T_{i,j}$. □

The plethora of properties for optimal walks will be used in the algorithmic investigations for finding optimal walks in Chapter 4 and Chapter 5. All properties will be referenced when used throughout the thesis.

4 Algorithms for Finding Optimal Walks

This chapter is devoted to finding optimal walks in temporal graphs. We will first introduce some algorithms for solving Single Source Optimal Walk and then we will give a reduction from temporal graphs to static graphs that maintains all optimal walks by using different cost functions. We continue with solving Single Sink Optimal Walk. We give a transformation of temporal graphs that 'reverses' the graph and, thus, allows us to use Single Source Optimal Walk algorithms. In the end, we will adapt the Floyd-Warshall Algorithm to solve All-Pairs Optimal Walk.

4.1 Single-Source Optimal Walk

In this section, we will examine Single Source Optimal Walk on temporal graphs. For the sake of simplicity, we only study instantaneous temporal graphs. A temporal graph can be transformed by Transformation 2 into an instantaneous temporal graph in linear time. To maintain a sorted time-arc list, there is $O(M \log \min\{\lambda_{\max}, M\})$ time needed as shown in Lemma 3.1.5 to sort the newly created time-arcs by time stamp. The time to newly sort the time-arc list can be the dominating factor in the overall running time.

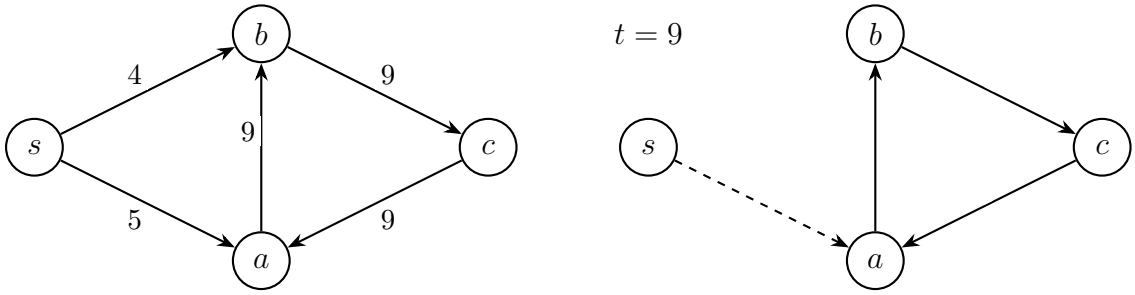
We introduce algorithms solving Single Source Optimal Walk for Foremost and Fastest that run in $O(M)$ and $O(M \log M)$ time respectively. The last algorithm is exemplary for all remaining optimal temporal walk definitions. In the end of the section, we introduce a transformation to static graphs in which the optimal walk definitions can be encoded via different cost functions on the arcs. This allows us to use all known results for finding shortest paths in static graphs.

4.1.1 Foremost

We start with an algorithm solving Single Source Foremost Walk on instantaneous temporal graphs. Afterwards, we will shortly discuss the adjustments that can be done to simplify the algorithm for instantaneous, incurable temporal graphs.

Given an instantaneous temporal graph $\mathcal{G} = (V, E, [T], \beta)$ and a source vertex $s \in V$, we construct an algorithm for finding the earliest arrival time from s to each vertex $v \in V$ in $\mathcal{G}$. The main difficulty an algorithm has to overcome is that the prefix-walks of any foremost walk does not have to be a foremost walk.

Example. Fig. 4.1(a) displays such a situation: The only walk from s to c is via the



(a) Temporal graph $\mathcal{G}$ with $\lambda \equiv 0$ and $\beta \equiv 4$

(b) Generated static graph for time step $t = 9$ in Algorithm 1

Figure 4.1: An instantaneous temporal graph and the static graph generated in Algorithm 1—Line 6 for time step $t = 9$.

vertex b. The earliest arrival time from s to b is in time step 4. But then, we would exceed the maximum dwell time of 4 before we can continue to c in time step 9. But with the arrival time 9 in b via a, we can arrive in vertex c.

We can conclude that we not only have to keep track of the earliest arrival time in a vertex, but also of the latest arrival time seen so far in any vertex.

This observation leads to the following idea for an algorithm: For each $t \in \{1, \dots, T\}$, the algorithm computes the latest arrival time from s to each vertex within time interval $[1, t]$. The algorithm can exploit the information of the latest arrival time within time interval $[1, t-1]$ for that purpose. Additionally, the algorithm has to store the smallest $t \in T$ for which it finds a latest arrival time from s to $v \in V$—the earliest arrival time in vertex v.

We will first explain the details of Algorithm 1 that implements the idea described above. We then prove the correctness of our approach and derive the running time of $O(M)$.

Given an instantaneous temporal graph $\mathcal{G} = (V, E, [T], \beta)$ and a source vertex $s \in V$, for each vertex $v \in V \setminus \{s\}$, we store the earliest arrival time from s to v in opt_v. The latest arrival time from s to v that we have seen so far in the run of our algorithm is stored in last_v. Initially, the variables are set to $\text{opt}_v = \infty$ and $\text{last}_v = -\infty$ (Lines 2 to 4). Now, for each $t \in \{1, \dots, T\}$ in which there exists at least one time-arc, Algorithm 1 generates a static graph G (Line 6 & Lines 16 to 19). This static graph consists of the static graph G_t, that is, the static graph induced by all time-arcs with time stamp t, and the source vertex s. Arcs are added from s to each vertex $v \in V_t$ for which we have previously found a latest arrival time last_v such that the maximum dwell time in v is not exceeded in time step t, that is, $E_r = \{(s, v) \mid v \in V_t \wedge t \leq \text{last}_v + \beta(v)\}$ (Lines 16 to 18). Such a generated graph is displayed in Fig. 4.1(b). The arcs in E_r are displayed with dashed lines; the arcs in E_t are displayed with solid lines.

Now, the algorithm conducts a modified breadth-first search on the graph G (Line 7) (for a short description of the algorithm see paragraph below). It returns a set V_{reached} of all vertices for which there exists a walk from s in G ending with an arc in E_t. For

Algorithm 1: Single Source Foremost Walk on temporal graphs with transmission time zero and bounded dwell time

```
/* Description of Variables:
   opt_v stores the earliest arrival time in v,
   last_v stores the latest arrival time in v not later than time step t,
   G is a temporal graph with a sorted time-arc list.
                                                                          */
1  function SingleSourceOptimalWalk(G, s ∈ V):
       /* Initialization                                                 */
2      for v ∈ V \ {s} do
3          opt_v = ∞
4          last_w = −∞
5      for t = 1, ..., T with E_t ≠ ∅ do
6          G ← GenerateDirectedGraph(G_t, s, opt, last)
           /* Modified breadth-first search: returns all vertices with
              an arrival time exactly at time step t                     */
7          V_reached ← modBFS(G, s)
8          for v ∈ V_reached do
9              opt_v = min{opt_v, t}
10             last_v = t
           /* Break condition: all earliest arrival times found          */
11         if ∀v ∈ V : opt_v < ∞ then
12             break
13     return opt
   /* Recall G_t = (V_t, E_t) is the static graph of time step t (see p.18
      for definition)                                                    */
14 function GenerateDirectedGraph(G_t, s, opt, last)):
15     E_r ← ∅
16     for v ∈ V_t \ {s} do
17         if t ≤ last_v + β(v) then
18             E_r ← E_r + (s, v)
19     return (V_t ∪ {s}, E_t ∪ E_r)
```

all $v \in V_{\text{reached}}$, it holds that there is an s-v walk in $\mathcal{G}$ arriving exactly at time step t. In Fig. 4.1(b), we can see that there is a walk with vertex sequence (s, a, b, c, a) in G from s to a with a solid arc. Thus, there is a temporal walk from s to a with arrival time exactly 9. For all vertices $v \in V_{\text{reached}}$, the algorithm sets $\text{opt}_v = \min\{t, \text{opt}_v\}$ and $\text{last}_v = t$ (Lines 9 and 10). Note that if opt_v is once set to something smaller than ∞, then it cannot improve anymore because the algorithm looks at the time steps in $[T]$ in ascending order. Consequently, if for all $v \in V$ it holds that $opt_v < \infty$ after a time step $t_{\max}$, then the algorithm can stop (Lines 11 and 12). At the end, the algorithm returns the earliest arrival times for all vertices (Line 13).

The *modified breadth-first search* is rooted in the source vertex s. For all vertices v, reached$[v]$ is set to false. In the first step all out-going arcs of s are added to a queue Q. For each arc (v, w) in Q where reached$[w]$ is set to false, all out-going arcs of w are added to Q and reached$[w]$ is set to true. If the arc (v, w) is in the set E_t, then the vertex w is added to V_{reached}. With this procedure, each vertex v for which there exists an $s - v$ walk ending with an arc in E_t is found. The algorithm runs in linear time because each arc is added at most once to Q. All other operations can be done in constant time.

Algorithm 1 finds the earliest arrival time for every vertex in $O(n + M_{t_{\max}})$ time where $M_{t_{\max}} := |\{(v, w, t, \lambda) \in E \mid t \leq t_{\max}\}|$, if a sorted time-arc list is given. In case we are also interested in an actual foremost walk, then we have to additionally store the time-arcs used to reach each vertex during the run of the algorithm.

We want to start by showing the running time of the Algorithm 1:

Lemma 4.1.1. *Algorithm 1 runs in $O(n + M)$ time.*

Proof. The initialization in Algorithm 1 is done in $O(n)$ time. Then, for each time step $t = 1, \ldots, T$, Algorithm 1 generates a static directed graph with $O(n_t)$ vertices and $O(m_t + n_t)$ arcs which takes $O(m_t + n_t)$ time. The static graph can be constructed by going through the sorted time-arc list and gathering all time-arcs with the same time step t. At most n_t additional arcs are added to the graph G_t. This requires $O(n_t)$ look ups. The entire procedure runs in linear time. On each of the static graphs, a modified breadth-first search is conducted in $O(n_t + m_t)$ time and the opt- and last-variables are updated in $O(n_t)$ time.

Note that n_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 $n_t \leq 2m_t$. Furthermore, we only have to check the time steps until we have found an arrival time for every vertex, the maximum of which is $t_{\max}$. We can sum up the running time by

$$\begin{aligned} & O\big(n + \sum_{t=1}^{t_{\max}} (m_t + n_t) + (m_t + n_t) + n_t\big) \\ = \; & O\big(n + \sum_{t=1}^{t_{\max}} m_t\big) \\ = \; & O\big(n + M_{t_{\max}}\big) \end{aligned}$$

Hence, Algorithm 1 runs in $O(n + M_{t_{\max}})$ time. □

To prove the correctness of Algorithm 1, we first show that it computes for each $t \in [T]$ and each $v \in V$ the latest arrival from s to v within the time interval $[1, t]$.

Lemma 4.1.2. *After time step $t \in [T]$, Algorithm 1 computed the latest arrival time from s to each vertex $v \in V$ within $\mathcal{G}[[1, t]]$.*

Proof. We prove this lemma by induction on the time step $t \in [T]$.

In the beginning of Algorithm 1, for all $v \in V$, $\text{last}_v = -\infty$. Now, the algorithm generates a graph $G = (V_1 \cup \{s\}, E_1)$. No further arcs are added because no vertex has been reached so far. If there is a walk from s to a vertex $v \in V \setminus \{s\}$ in G, then Algorithm 1 sets $\text{last}_v = 1$. This is the latest arrival time possible for $t = 1$.

Now, let us assume that after a time step $t \in [T]$, Algorithm 1 computed the latest arrival time from s to each vertex $v \in V$ within $\mathcal{G}[[1, t]]$. If for time step $t+1$ there is no time-arc with time stamp $t+1$, then there cannot be an arrival time $t+1$ in any vertex. Thus, the latest arrival times in $\mathcal{G}[[1, t+1]]$ are equal to the latest arrival times in $\mathcal{G}[[1, t]]$ which were computed correctly by induction hypothesis. Otherwise, only vertices with an in-going time-arc in time step $t + 1$ are candidates for a latest arrival time in $t + 1$.

Let us assume towards a contradiction that there is a vertex $v \in V_{t+1}$ which was not updated after time step $t + 1$ even though there exists a temporal walk from s to v with arrival time exactly $t + 1$. Let $P = (e_1, \ldots, e_k)$ be such a temporal walk with $e_i = (v_i, w_i, t_i, 0) \in E$.

Case 1. If $t_i = t+1$ for all $i \in [k]$, then the walk $((v_1, w_1), \ldots, (v_k, w_k))$ is a path from s to v in G_{t+1}. In the modified breadth-first search v can be reached from s with the arc $(v_k, w_k) \in E_t$ and, thus, last_v is set to $t + 1$. This is a contradiction to our assumption.

Case 2. Otherwise, there is an $i \in [k-1]$ such that for all $j \in [i]$ it holds that $t_j < t+1$ and for all $j' \in \{i + 1, \ldots, k\}$ it holds that $t_{j'} = t + 1$. By our induction hypothesis, we know that for w_i the algorithm computed that latest arrival time last_{w_i} within time interval $[1, t]$ correctly. It holds that $t_i \leq \text{last}_{w_i}$. We further know that

$$t + 1 \leq t_i + \beta(w_i) \leq \text{last}_{w_i} + \beta(w_i) \tag{4.1}$$

because P is a valid temporal walk.

Let us now consider the generated graph $G = (V_{t+1} \cup \{s\}, E_t \cup E_r)$ with

$$E_r = \{(s, v) \mid v \in V_{t+1} \wedge t + 1 \leq \text{last}_v + \beta(v)\}$$

in Line 6. The walk $W = ((v_{i+1}, w_{i+1}), \ldots, (v_k, w_k))$ is a walk from s to v in $G_{t+1} = (V_{t+1}, E_{t+1})$ because $t_{j'} = t + 1$ for $j' \in \{i + 1, \ldots, k\}$. Thus, W is contained in G. Also the arc (s, v_{i+1}) is contained in E_r due to Eq. (4.1) and $w_i = v_{i+1}$. Thus, there is a walk from s to v in G. Our modified breadth-first search on G returns the vertex v because it can be reached via the arc $(v_k, w_k) \in E_T$. Recall that $v = w_k$. Consequently, last_v is set to $t + 1$. This is a contradiction to our assumption.

Algorithm 1 uses in the generated graph G in Line 6 only existing arcs or arcs that represent a valid temporal walk—see Lines 16 to 19. Thus, a non-existing latest arrival time for any vertex can not be found. We can conclude that after a time step $t \in [T]$, Algorithm 1 computed the latest arrival time from s to each vertex $v \in V$ within $\mathcal{G}[[1,t]]$ correctly. □

With Lemma 4.1.2 at hand, we can prove the correctness of Algorithm 1:

Lemma 4.1.3. *Algorithm 1 solves* Single Source Foremost Walk.

Proof. Assume towards a contradiction that Algorithm 1 does not find the correct earliest arrival time from s to a vertex $v \in V$. Let t_v^* be the actual earliest arrival time in v. But then, the latest arrival time of v within $\mathcal{G}[[1,t_v^*]]$ is t_v^*. This was computed correctly by Algorithm 1 as shown in Lemma 4.1.2. For time step t_v^* the variable opt_v was set to t_v^* and never changed again, as discussed above. □

With Lemmas 4.1.1 and 4.1.3 we can finally state the following theorem:

Theorem 4.1.4. *Algorithm 1 solves* Single Source Foremost Walk *on an instantaneous temporal graph $\mathcal{G}$ in $O(n + M_{t_{\max}})$ time where $t_{\max}$ is the largest earliest arrival time to any vertex in $\mathcal{G}$.*

Remark. If there is a temporal graph with unbounded dwell time in the vertices, then it is not necessary to compute the latest arrival time within every time interval $[1,t]$ for $t \in T$. It is sufficient to only compute the earliest arrival time in each vertex. We make use of the fact that there always exists a foremost path such that all prefix paths are foremost paths shown in Lemma 3.2.6. However, this simplification has no influence on the running time of the algorithm.

4.1.2 Fastest

Now, we will have a closer look at fastest walks and the complexity Single Source Fastest Walk on instantaneous temporal graphs. We introduce an algorithm that computes a fastest walk from a source vertex s to every vertex in a temporal graph in $O(M \log M)$ time. We will then briefly explain how this algorithm can be simplified for instantaneous, incurable temporal graphs.

Given a temporal graph $\mathcal{G} = (V, E, [T], \beta)$ and a source vertex $s \in V$, the main idea of the algorithm is the following: For every $t \in [T]$, it computes the latest departure time for a walk from s to all vertices that arrives exactly in time step t. The fastest walk from s to a vertex w is a reverse-foremost walk—a walk with the latest departure time—within the time interval $[1,t]$ with $t \in [T]$ by Lemma 3.2.3. Thus, during the execution of the algorithm we find the fastest walk from s to any vertex $w \in V$.

Algorithm 2 has two additional benefits:

1. It computes the reverse-foremost walk—the latest departure time—from s to each vertex $w \in V$. This is the latest possible departure time for a walk from s to w that has been found during the run of the algorithm.

Algorithm 2: Single Source Fastest Walk on temporal graphs with transmission time zero and bounded dwell time

```
/* Description of Variables:
   opt_v stores the duration of a fastest path to w within [0,t],
   L_v is a sorted list [(d_1,a_1),...,(d_k,a_k)] where d_i is a the latest
       departure time of a walk from s to v that arrives in time a_i
       with a_i ≤ t ≤ a_i + β_w. We will sort the list such that:
       d_1 > d_2 > ··· > d_k and a_1 < a_2 < ··· < a_k, and
   G is a temporal graph with a sorted time-arc list.
                                                                    */
1  function SingleSourceFastestWalk(G, s ∈ V):
       /* Initialization                                           */
2      for v ∈ V \ {s} do
3          opt_v = ∞
4          L_v ← empty list
5      for t = 1,...,T with E_t ≠ ∅ do
6          G, c ← GenerateDirectedGraph(G_t, s, L)
           /* Modified Dijkstra Algorithm: returns all vertices with an
              arrival time exactly at time step t and their latest
              departure time                                        */
7          V_reached, c_dep ← modDijkstra(G, c, s)
8          for v ∈ V_reached do
9              opt_v = min{opt_v, t − (T − c_dep(v))}
10             L_v ← add (T − c_dep(v), t) & delete redundant tuples (Lemma 4.1.5)
11     return opt
   /* Recall G_t = (V_t, E_t) is the static graph of time step t (see p.22
      for definition)                                               */
12 function GenerateDirectedGraph(G_t, s, L)):
13     E_r ← ∅
14     for v ∈ V_t \ {s} do
15         if ∃(d, a) ∈ L_v : a ∈ [t − β(v), t] then
16             E_r ← E_r ∪ {(s, v)}
17             c((s, v_r)) = T − d with d := max{d | (d, a) ∈ L_v ∧ a ∈ [t − β(v), t]}
18     for e ∈ E_t do
19         c(e) = { T − t   , if e = (s, ∗)
                  { 0       , else
20     return ((V_t ∪ {s}, E_t ∪ E_r), c)
```

2. It computes the foremost walk—the earliest arrival time—from s to each vertex $w \in V$. This is the the earliest time step in which the algorithm finds a walk from s to w. Among all foremost walks from s to w it even finds the one with the latest departure time in s.

Next, we will explain the main computational steps of Algorithm 2. We will further describe how the information about the latest departure time for a walk from s to all vertices that arrives exactly in time step t for each $t \in [T]$ can be stored efficiently. The efficient access to this data is the crux of the algorithm. We are able to maintain the information in $O(M)$ time during the execution of the algorithm to allow constant time access to the data of interest. We will derive a total running time of $O(M \log M)$ time for Algorithm 2.

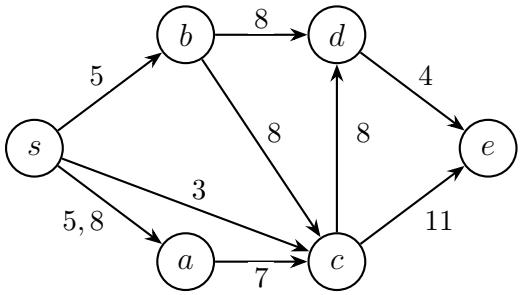
The details of Algorithm 2 are as follows: Let $\mathcal{G} = (V, E, [T], \beta)$ be an instantaneous temporal graph and vertex $s \in V$ be the source. For each vertex $v \in V \setminus \{s\}$, it stores the duration of a fastest walk from s to w in opt_v and all arrival times from s to v with their latest possible departure time in L_v. In the beginning, $\text{opt}_v = \infty$ and L_v is empty (Lines 2 to 4).

For each time step $t \in [T]$, the algorithm computes the latest departure time for walks from s to all vertices that arrive exactly in time step t. It generates a graph G (Line 6 and Lines 14 to 20). 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. Arcs are added from s to each vertex $v \in V_t$ for which there exists a temporal walk from s to v that arrives within the maximum dwell time, that is, within $[t - \beta(v), t]$. Let d be the latest departure time among all such walks. The weight of the arc is then set to $T - d$. This information can be found in the list L_v. Let E_r be the set of the additionally added arcs, that is, $E_r = \{(s, v) \mid v \in V_t \setminus \{s\} \wedge \exists (d, a) \in L_v \colon a \in [t - \beta(v), t]\}$ (Lines 14 to 17). The weight of a arcs $(v, w) \in E_t$ are set or overwritten to 0, if $v \neq s$, and to $T - t$, if $v = s$ (Lines 18 and 19).

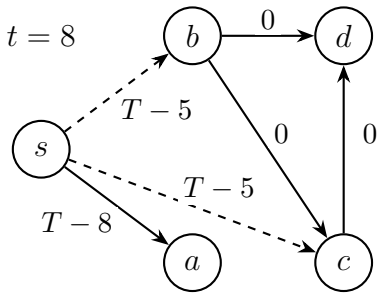
Now, Algorithm 2 runs a modified Dijkstra Algorithm on G (for a short description of the algorithm see paragraph below). It computes a shortest walk from s to each vertex ending with an arc in E_t. The shortest walk is a walk with the latest departure time. The modified Dijkstra Algorithm returns the set of vertices V_{reached} that can be reached within G via an arc in E_t and the function $c_{\text{dep}} \colon V_{\text{reached}} \to [1, t]$ that maps each $v \in V_{\text{reached}}$ to its latest departure time d (in form $T - d$) for a s-v walk that arrives exactly in t. For each $v \in V_{\text{reached}}$, opt_v is set to the minimum of its current value and the duration of a newly computed walk, that is, $\text{opt}_v = \min\{t - (T - c_{\text{dep}}(v)), \text{opt}_v\}$ (Line 9). The tuple $(T - c_{\text{dep}}(v), t)$ is added to list L_v (Line 10).

The *modified Dijkstra Algorithm* is rooted in the source vertex s. In the first step, the source vertex s with distance$[s]$ set to zero is added to a priority queue Q. All other vertices v are added to Q with distance$[v]$ set to infinity and reached$[v]$ set to infinity. Now, in each step the vertex v with the smallest distance to s is removed from Q. Then, for all out-going arcs (v, w) from v the algorithm checks whether

$$\text{distance}[w] > \text{distance}[v] + c((v, w)).$$



(a) Temporal graph $\mathcal{G}$ with $\lambda \equiv 0$ and $\beta \equiv 4$



(b) Generated static graph for time step $t = 8$ in Algorithm 2

Figure 4.2: An instantaneous temporal graph and the static graph generated in Algorithm 2—Line 6 for time step $t = 8$.

If it holds, then a shorter path is found. If $(v, w) \in E_t$, then the algorithm checks whether

$$\text{reached}[w] > \text{distance}[v] + c((v, w)).$$

If it holds, then a later departure time of a walk to w is found that arrives exactly in time step t. With this procedure, each vertex v for which there exists an s-v walk ending with an arc in E_t with the latest departure time is found. The algorithm runs in $O(m_t \log m_t)$.

Example. In Fig. 4.2, we briefly explain Algorithm 2 on the temporal graph $\mathcal{G}$ displayed in Fig. 4.2(a). After the algorithm considered all time steps up to $t = 7$, walks from s to the vertices a, b, and c is found. More precisely, $L_a = [(5, 5)]$, $L_b = [(5, 5)]$, and $L_c = [(3, 3), (5, 7)]$. In time step 8, Algorithm 2 first removes all elements for which the maximum dwell time is exceeded, that is, $(3, 3)$ in L_c due to the arrival time 3. Now the static graph in Line 6 is generated. This is displayed in Fig. 4.2(b): The solid black arcs belong to G_8—the static graph induced by all time-arcs with time stamp 8 in $\mathcal{G}$—and the dashed black arcs E_r represent walks from s to the vertices with arrival time within $[t - \beta, t - 1] = [4, 7]$. The weight is T minus the latest departure time to arrive within the time interval $[4, 7]$. Now, Algorithm 2 conducts the modified Dijkstra Algorithm on the graph in Fig. 4.2(b). A walk from s to d via c is found that contains at least one arc of E_t, that is, a black arc, with minimum cost $T - 5$, hence, with latest departure time 5 and arrival time 8. The tuple $(5, 8)$ is added to L_c and opt_d is updated to $8 - 5 = 3$. Furthermore, there is a walk from s to c via b that ends with a solid black arc with minimum cost $T - 5$. Thus, there is a walk from s to c with latest departure time 4 that arrives exactly in time step 8. Thus, $(5, 8)$ is added to L_c. The tuple $(8, 8)$ is added to L_a.

The list L_c now contains $[(5, 7), (5, 8)]$. But the element $(5, 7)$ is not needed anymore because both tuples have a departure time 5 whereas $(5, 8)$ has a greater arrival time. Hence, for time steps $9, \ldots, 12$ (time steps for which the maximum dwell time in c is not exceeded for the arrival time 8), the tuple $(5, 8)$ is sufficient to determine the latest

departure time. The general structure of these lists is discussed in the next paragraph.

To implement Algorithm 2 efficiently, we have to optimize the list L_v for each vertex $v \in V$. Recall that L_v is the list of all arrival times within $[1, t]$ from s to v with their corresponding latest departure time for which we have not exceeded the maximum dwell time in v for a time step t. Let

$$L_v = [(d_1, a_a), (d_2, a_2), \ldots, (d_k, a_k)]$$

with $t - \beta(v) \leq a_1 < a_2 < \cdots < a_k < t$ be such a list. It holds for each tuple (d_i, a_i) with $i \in [k]$ that there is a walk from s to v that arrives in time step a_i and d_i is the latest departure time among all walks from s to v that arrive exactly in time step a_i. For these tuples, we can show that some of them are redundant:

Lemma 4.1.5. *For a time step $t \in [T]$ and a vertex $v \in V$, if there are two tuples $(d, a), (d', a') \in L_v$ with $a < a'$ and $d \leq d'$, then (d, a) can be removed from L_v.*

Proof. The tuples $(d, a), (d', a') \in L_v$ imply that there is a temporal walk P from s to v with departure time d and arrival time a and another temporal walk P' from s to v with departure time d' and arrival time a'. After time step t, Algorithm 2 only considers time-arcs with time stamps $t' \in \{t+1, \ldots, T\}$. For any generated graph G (Line 6) for a time step t' with $v \in V_{t'}$, the algorithm adds an arc from s to v if there is an arrival time within $[t' - \beta(v), t']$. If $a \in [t' - \beta(v), t']$, then $a' \in [t' - \beta(v), t']$ because $a < a' < t'$. Furthermore, let d_l be the latest departure time of a walk from s to v such that its arrival time is within $[t' - \beta(v), t']$. The weight of the arc (s, v) is set to $T - d_l$. Thus, if $a, a' \in [t' - \beta(v), t']$, then the weight of (s, v) is set to $T - d'$ because $d \leq d'$. Thus, the tuple (d, a) is not needed in the list L_v at time step t anymore and can be removed. □

After removing all redundant tuples from L_v, it holds that $d_1 > d_2 > \cdots > d_k$ in L_v.

(1) Due to the sorting $a_1 < a_2 < \cdots < a_k$, it is not necessary to go through the whole list to through out elements that are not in the range of the maximum dwell time anymore.

(2) Due to the sorting $d_1 > d_2 > \cdots > d_k$, it is not necessary to go through the whole list to find the redundant tuples for a newly added element (d, a). It holds that $a_k < a$ and, thus, all tuples (d_i, a_i) with $d_i < d$ can be deleted.

After establishing the structure of the list L_v for $v \in V$, we can show the running time of Algorithm 2:

Lemma 4.1.6. *Algorithm 2 runs in $O(n + M \log M)$ time.*

Proof. The initialization in Algorithm 2 can be done in $O(n)$ time. Then, for each time step $t = 1, \ldots, T$, Algorithm 2 generates a static directed graph $G = (V_t \cup \{s\}, E_t \cup E_r)$ with $O(n_t)$ vertices and $O(m_t + n_t)$ arcs which takes $O(m_t + n_t)$ time. The correct weight for each arc $(s, v) \in E_r$ can be found in L_v. More precisely, Algorithm 2 first deletes all

elements $(d, a) \in L_v$ with $a < t - \beta(v)$. Owing to the sorting of L_v this takes only $O(M)$ time during the whole run of the algorithm because we only delete at most as many elements as there are time-arcs in the temporal graph. Recall that if $(d, a) \in L_v$, then there exists a time-arc $(*, v, a, 0) \in E$.

In each of the generated graphs G, the modified Dijkstra Algorithm is executed in $O(m_t \log m_t)$ time. The opt_v variables are updated, and an element is added to L_v for each $v \in V_{\text{reac}}$ in $O(n_t)$ time. finally, Algorithm 2 deletes all elements in L_v that are redundant. Owing to the sorting in L_v, it takes only $O(M)$ time during the whole run of the algorithm.

Note, that n_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 $n_t \leq 2m_t$. We can sum up the running time by

$$\begin{aligned}
&O\big(n + M \log M + M + \sum_{t=1}^{T} (m_t + n_t) + (m_t \log m_t) + n_t\big) \\
\subseteq\ &O\big(n + \sum_{t=1}^{T} m_t \log m_t\big) \\
=\ &O\big(n + M \log M + M \log M\big) \\
=\ &O\big(n + M \log M\big)
\end{aligned}$$

Hence, Algorithm 2 runs in $O\big(n + M \log M\big)$ time. Note that if for each $t \in [T]$ the generated graph G is an acyclic graph, then the modified Dijkstra Algorithm runs in linear time. Thus, the running time of Algorithm 2 would be $O(n + M)$. □

For the correctness of Algorithm 2, we first have to show that for every $t \in [T]$ and for every $v \in V$, Algorithm 2 computes a walk from s to v that arrives exactly in t with the latest departure time among all walks from s to v that arrive exactly in t.

Lemma 4.1.7. *In time step $t \in [T]$, Algorithm 2 computes the latest departure time for a temporal walk from s to $v \in V$ that arrives exactly in time step t.*

Proof. The proof is by induction on the time step $t \in [T]$.

In the beginning of Algorithm 2 , L_v is empty for all $v \in V$. Now, the algorithm generates a graph $G = (V_1 \cup \{s\}, E_1)$. No further arcs are added because no vertex has been reached so far. All arcs from s to a vertex are weighted with $T-1$, all other arcs are weighted with zero. If there is a walk from s to a vertex $v \in V$ in G, then Algorithm 1 finds a walk from s to v with value $T - 1$. Thus, it adds $(1, 1)$ to L_v. Time step 1 is the latest departure time possible to arrive in time step 1.

Now, let us assume that in all time steps $t' \in [t]$, Algorithm 2 computes the latest departure time for a temporal walk from s to $v \in V$ that arrives exactly in time step t'. If for time step $t+1$ a vertex $v \in V$ has no in-going time-arcs with time stamp $t+1$, 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 $P = (e_1, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i, 0) \in E$ be a walk with the latest departure time among all temporal walks from s to v that arrive exactly in time step $t+1$. The time step t_1 is the latest departure time. Let us assume towards a contradiction that Algorithm 2 does not find that walk and, thus, does not add the tuple $(t_1, t+1)$ to L_v.

Case 1. If $t_i = t+1$ for all $i \in [k]$, then the walk $((v_1, w_1), \ldots, (v_k, w_k))$ is a walk from s to v in G_{t+1}. Thus, the modified Dijkstra Algorithm finds a walk from s to v ending with an arc in E_t and value $T - (t+1)$ because the weight of (v_1, w_1) is set to $T - (t+1)$. The time step $t+1$ is the latest time possible to depart from s and arrive in v in time step $t+1$. Thus, $(t+1, t+1)$ is added to L_v. This is a contradiction to our assumption.

Case 2. Otherwise, there is an $i \in [k-1]$ such that for $j \in [i]$ it holds that $t_j < t+1$ and for $j' \in \{i+1, \ldots, k\}$ it holds that $t_{j'} = t+1$. We know that t_1 has to be the latest departure time for a walk from s to w_i to arrive exactly in time step t_i. Otherwise, P would not be a walk with the latest departure time. By our induction hypothesis, Algorithm 2 computed the latest departure time to w_i with arrival time exactly t_i correctly. Thus, the tuple (t_1, t_i) was added to L_{w_i}. If (t_1, t_i) is not in L_{w_i} in time step $t+1$, then there must be another tuple $(t_1, \hat{t}_i)$ in L_{w_i} with $t_i < \hat{t}_i < t+1$ due to Lemma 4.1.5. We further know that

$$t + 1 \leq t_i + \beta(w_i) \leq \hat{t}_i + \beta(w_i) \tag{4.2}$$

because P is a valid temporal walk.

Let us now consider the generated graph $G = (V_{t+1} \cup \{s\}, E_{t+1} \cup E_r)$. The arc sequence $((v_{i+1}, w_{i+1}), \ldots, (v_k, w_k))$ is a walk from v_{i+1} to $w_k = v$ in $G_{t+1} = (V_{t+1}, E_{t+1})$ because $t_{j'} = t+1$ for $j' \in \{i+1, \ldots, k\}$ and, thus, contained in G. Also the arc (s, v_{i+1}) is contained in E_r with weight $T - t_1$, due to $w_i = v_{i+1}$ and Eq. (4.2). Thus, there is a walk from s to v in G. The modified Dijkstra Algorithm on G returns the vertex v with value $T - t_1$ because there is a walk from s to v ending with an arc in E_{t+1}. Consequently, $(t_1, t+1)$ is added to L_v. This is a contradiction to our assumption.

Wrongly adding a tuple (d, a) to a list L_v is not possible because the algorithm uses in G only existing arcs of G_t and arcs that represent a valid temporal walk.

Thus, after a time step $t \in [T]$, Algorithm 2 computed the latest departure time for a temporal walk from s to $v \in V$ that arrives exactly in time step t. □

Last but not least, we prove the correctness of Algorithm 2:

Lemma 4.1.8. *Algorithm 2 solves* Single Source Fastest Walk.

Proof. Assume towards a contradiction that Algorithm 2 does not find the minimum duration from s to a vertex $v \in V$. Let $P = (e_1, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i, 0) \in E$ be

a walk with the minimum duration among all temporal walks from s to v. This walk P is a reverse-foremost walk within time interval $[1, t_k]$ by Lemma 3.2.3. That means, P has the latest departure time among all temporal walks from s to v that arrive exactly in time step t_k. This was computed correctly by Algorithm 2 in time step t_k as shown in Lemma 4.1.7. Thus, the tuple (t_1, t_k) was added to L_v and the variable opt_v was set to $t_k - t_1$—the minimum duration of a walk from s to v. That is a contradiction to our assumption that Algorithm 2 did not find the fastest walk. □

Due to Lemmas 4.1.6 and 4.1.8, we can conclude the following theorem:

Theorem 4.1.9. *Algorithm 2 solves* Single Source Fastest Walk *on an instantaneous temporal graph in* $O(n + M \log M)$ *time.*

Remark. Note that the running time our algorithm is dominated by the running time of the modified Dijkstra Algorithm that runs in $O(M \log M)$ time. This running time overshadows the running time of the sorting of the time-arcs by time stamp. Thus, we do not have to assume that a sorted time-arc list is given to reach the running time.

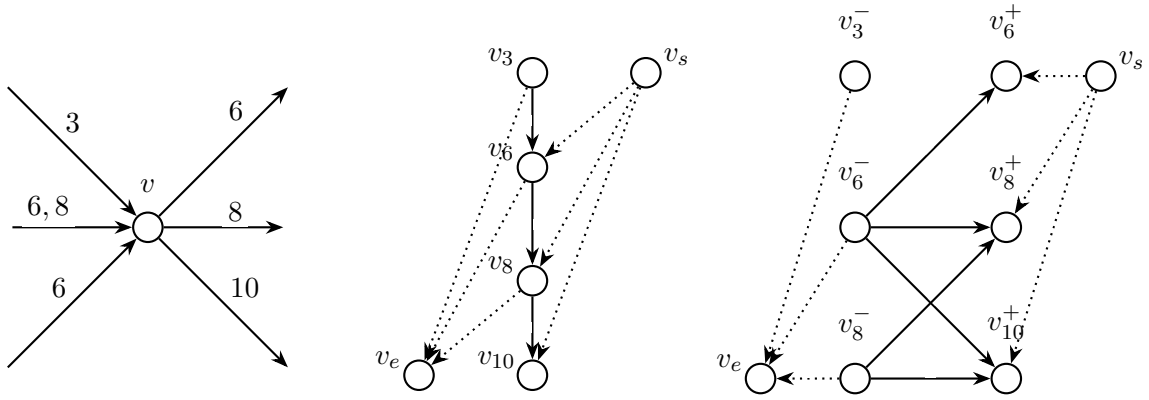
If an instantaneous, incurable temporal graph is given, then it is sufficient to only store the latest departure time seen so far instead of a list L.

We already established in the beginning of this subsection that Algorithm 2 also computes the Reverse-Foremost: the latest departure time from s to each vertex $v \in V$ during the run of the algorithm. For Cheapest and Minimum Hop-Count, the main ideas of Algorithm 2 can be adapted: for every $t \in [T]$, the algorithm computes the minimum cost and minimum hop-count respectively for a walk from the s to all vertices in V that arrives exactly in time step t (instead of computing the latest departure time). The definition Shortest does not make sense to consider in instantaneous temporal graphs, and the Minimum Waiting Time is equal to Fastest.

For non-instantaneous temporal graphs some observations have to be recalled: As we have established in Transformation 2, we can transform in linear time any non-instantaneous temporal graph into an instantaneous temporal graph. A sorted time-arc list can be maintained in $O(M \log \min\{\lambda_{\max}, M\})$ time as shown in Lemma 3.1.5. Recall that for each $t \in [T]$ the static graph G_t is acyclic as shown in Observation 3.1.4. The arcs from s to some vertices in V_t could lead to a cycle, but we can delete all arcs from any vertex to s when we conduct the modified Dijkstra Algorithm. Thus, the generated graph G remains acyclic. We can conclude that Single Source Fastest Walk on non-instantaneous temporal graphs can be solved in $O(n + M \log \min\{\lambda_{\max}, M\})$ time (see proof of Lemma 4.1.6).

4.1.3 Transformation to Static Graphs

We will now show how a temporal graph can be transformed to a static graph such that all temporal walks can be extracted from the static graph. Even more, the different optimality values of our optimal walks can be preserved by different cost function. Conducting this transformation allows us to use the plethora of results for finding shortest paths in path in static graphs for our purpose.



(a) Vertex v with in- and out-going time-arcs

(b) Gadget of v for $\beta \equiv \infty$

(c) Gadget of v for $\beta \equiv 4$

Figure 4.3: A vertex v with $\tau_v^- = \{3, 6, 8\}$, $\tau_v^+ = \{6, 8, 10\}$ and the gadgets of v within the transformations to static graphs.

Depending on the existence of bounded maximum dwell time in the vertices, the reduction slightly differs, but the main idea is to build a gadgets for every vertex in the temporal graph. The gadget contains a vertex for every time step in which the vertex has in-going or out-going time-arcs. These vertices shall be connected to each other such that the connections represent valid time sequences within a temporal walk. We have to distinguish between incurable temporal graphs and general temporal graph:

Incurable Temporal Graphs. For a given temporal graph $\mathcal{G} = (V, E, [T])$ and a vertex $v \in V$, the gadget contains a vertex v_t for each $t \in \tau_v = \tau_v^- \cup \tau_v^+$. These vertices represent the vertex v at the time steps t. A vertex v_t shall be connected to all vertices $v_{t'}$ if and only if $t \leq t'$ (non-decreasing in time).

Let $\tau_v = \{t_1, t_2, \ldots, t_{|\tau_v|}\}$ with $t_1 \leq t_2 \leq \cdots \leq t_{|\tau_v|}$. An arc is added from v_{t_i} to $v_{t_{i+1}}$ for all $i \in [|\tau_v| - 1]$. In this way, there is a a path from v_t to any $v_{t'}$ with $t \leq t'$ and no path from $v_{t'}$ to v_t within the gadget. An example is provided in Fig. 4.3(b) with the vertices and the solid black arcs.

Temporal Graphs. For a given temporal graph $\mathcal{G} = (V, E, [T], \beta)$ and a vertex $v \in V$, the gadget contains a vertex v_t^- for each $t \in \tau_v^-$ and a vertex v_t^+ for each $t \in \tau_v^+$. These vertices represent the vertex v at the certain time steps t of in-going or out-going time-arcs. Here, a vertex v_t^- shall be connected to all vertices $v_{t'}^+$ if and only if $t \leq t'$ (non-decreasing in time) and $t' \leq t + \beta(v)$ (non-exceeding maximum dwell time).

Let $\tau_v^+ = \{t_1^+, t_2^+, \ldots, t_{|\tau_v^+|}^+\}$ with $t_1^+ \leq t_2^+ \leq \cdots \leq t_{|\tau_v|}^+$. An arc is added from v_t^- to $v_{t_j}^+$ with $t_j \in \tau_v^+$, $t \leq t_j$ and $t_j \leq t + \beta(v)$. In this way, there is a direct arc from v_t^- to $v_{t'}^+$ if and only if $t \leq t' \leq t + \beta(v)$ and nowhere else. An example is provided in Fig. 4.3(c) with the vertices and the solid black arcs.

In both cases, a vertex v_s and v_e are added additionally to the gadget representing the vertex as a startpoint of a temporal walk and an endpoint of a temporal walk, respectively. An arc is added from v_s to every v_t/v_t^+ for every $t \in \tau_v^+$ and from v_t/v_t^- to v_e for every $t \in \tau_v^-$. These arcs are illustrated by the dashed arcs in Figs. 4.3(b) and 4.3(c). Also an arc from v_s to v_e is added.

By building a gadget for every vertex $v \in V$, the time component of the temporal graph is completely encoded in the time step representatives in the gadgets. The last step is to connect the gadgets according to the time-arcs. That is, for every time-arc $(v, w, t, \lambda) \in E$ an arc $(v_t, w_{t+\lambda})/(v_t^-, w_{t+\lambda}^+)$ is added.

A similar approach was introduced by Wu et al. [Wu+16] for incurable temporal graphs and even earlier by Dean [Dea04] in the research field of time-dependent networks with waiting policies. We extend the reduction by encoding all of our optimal walk definitions as cost functions on the arcs. We analyze the running time of the reduction and the size of the resulting static graph. Furthermore, we discuss briefly the structure of the resulting static graph and its impact on the running time of solving Single Source Optimal Walk. Due to the similarity of these two transformation, we only introduce the transformation of temporal graphs in general.

Transformation 3. Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph and let $c\colon E \to \mathbb{N}$ be a cost function on the time-arcs. We construct a directed static graph $G = (V_G, E_G)$ with

(1) $V_G = V_t^- \cup V_t^+ \cup V_s \cup V_e$ where

$$\begin{aligned} V_t^- &= \{v_t^- \mid v \in V \wedge t \in \tau_v^-\}, \\ V_t^+ &= \{v_t^+ \mid v \in V \wedge t \in \tau_v^+\}, \\ V_s &= \{v_s \mid v \in V\}, \text{ and} \\ V_e &= \{v_e \mid v \in V\}. \end{aligned}$$

(2) $E_G = E_s \cup E_e \cup E_{s,e} \cup E_w \cup E_t$ where

$$\begin{aligned} E_s &= \{(v_s, v_t^+) \mid v \in V \wedge t \in \tau_v^+\}, \\ E_e &= \{(v_t^-, v_e) \mid v \in V \wedge t \in \tau_v^-\}, \\ E_{s,e} &= \{(v_s, v_e) \mid v \in V\}, \\ E_w &= \{(v_t^-, v_{t'}^+) \mid v \in V \wedge t \in \tau_v^- \wedge t' \in \tau_v^+ \wedge t \leq t' \leq t + \beta(v)\}, \text{ and} \\ E_t &= \{(v_t^+, w_{t+\lambda}^-) \mid (v, w, t, \lambda) \in E\}. \end{aligned}$$

To preserve the optimization value of the temporal walks, we introduce several cost functions on the arcs E_G:

Foremost. Let $c_{\mathrm{f}}\colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{f}}(e) = \begin{cases} t & \text{, if } e = (v_t^-, v_e) \in E_e \\ 0 & \text{, else.} \end{cases}$$

Reverse-Foremost. Let $c_{\mathrm{r}} \colon E_G \to \mathbb{N}$ with

$$c_r(e) = \begin{cases} T - t & \text{, if } e = (v_s, v_t^+) \in E_s \\ 0 & \text{, else.} \end{cases}$$

Fastest. Let $c_{\mathrm{fa}} \colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{fa}}(e) = \begin{cases} t' - t & \text{, if } e = (v_t^+, w_{t'}^-) \in E_t \\ t' - t & \text{, if } e = (v_t^-, v_{t'}^+) \in E_w \\ 0 & \text{, else.} \end{cases}$$

Shortest. Let $c_{\mathrm{s}} \colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{s}}(e) = \begin{cases} t' - t & \text{, if } e = (v_t^+, w_{t'}^-) \in E_t \\ 0 & \text{, else} \end{cases}$$

Minimum Hop-Count. Let $c_{\mathrm{h}} \colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{h}}(e) = \begin{cases} 1 & \text{, if } e = (v_t^+, w_{t'}^-) \in E_t \\ 0 & \text{, else.} \end{cases}$$

Cheapest. Let $c_{\mathrm{c}} \colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{c}}(e) = \begin{cases} c((v, w, t, t' - t)) & \text{, if } e = (v_t^+, w_{t'}^-) \in E_t \\ 0 & \text{, else.} \end{cases}$$

Minimum Waiting time. Let $c_{\mathrm{w}} \colon E_G \to \mathbb{N}$ with

$$c_{\mathrm{w}}(e) = \begin{cases} t' - t & \text{, if } e = (v_t^-, v_{t'}^+) \in E_w \\ 0 & \text{, else.} \end{cases}$$

Any walk in the static graph G is equivalent to a temporal walk in $\mathcal{G}$ and vice versa. Even more, a shortest walk in G with respect to one of the above mentioned cost functions is an optimal walk in $\mathcal{G}$ and vice versa.

Lemma 4.1.10. *Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph, $G = (V_G, E_G)$ be the static graph by applying Transformation 3 to $\mathcal{G}$, and $v, w \in V$ be two vertices. There is an optimal temporal walk from v to w in $\mathcal{G}$ with optimality value c^* if and only if there exists a shortest path from v_s to v_e in G with minimum cost c^*.*

Proof. Let $v, w \in V$ be two vertices of the temporal graph. Let $P = (e_1, \dots, e_k)$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ for all $i \in [k]$ be an optimal temporal walk from v to w. Then there are the following arcs in G:

$$\begin{aligned}
s &= ((v_1)_s, (v_1)^+_{t_1}) && \in E_s \text{ due to } t_1 \in \tau^+_{v_1},\\
a_i &= ((v_i)^+_{t_i}, (w_i)^-_{t_i+\lambda_i}) && \in E_t \text{ due to } (v_i, w_i, t_i, \lambda_i) \in E \text{ for } i \in [k],\\
b_i &= ((w_i)^-_{t_i+\lambda_i}, (w_i)^+_{t_{i+1}}) && \in E_w \text{ due to } t_i + \lambda_i \le t_{i+1} \le t_i + \lambda_i + \beta(w_i) \text{ for } i \in [k-1],\\
e &= ((w_k)^-_{t_k+\lambda_k}, (w_k)_e) && \in E_e \text{ due to } t_k + \lambda_k \in \tau^-_{w_k}.
\end{aligned}$$

Thus, there is a directed path $P_G = (s, a_1, b_1, \dots, a_{k-1}, b_{k-1}, a_k, e)$ from v_s to w_e in G. We also know that each path from v_s to w_e has to have a structure similar to P_G with respect to sequence of arcs. Hence, the translation of a directed path P_G in G to a temporal walk in $\mathcal{G}$ works by extracting the arcs a_i with $i \in [k]$ from P_G. Each $a_i = ((v_i)^+_{t_i}, (w_i)^-_{t_i+\lambda_i})$ belongs to a corresponding time arc $e_i = (v_i, w_i, t_i, \lambda_i)$. Thus, there is a temporal walk P from v to w in $\mathcal{G}$ if and only if there is a walk P_G from v_s to w_e in G.

Now, it remains to analyze that the optimality value of P in $\mathcal{G}$ is equal to the cost of P_G in G for all the optimal walk definitions:

Foremost. The optimality value of the temporal path P is the arrival time in w, that is, $t_k + \lambda_k$. In G, we use the cost functions c_{f} to find the shortest path. Only arcs in E_e have costs. Thus, the cost of P_G is $c(e) = t_k + \lambda_k$.

Reverse-Foremost. The optimality value of the temporal path P is the departure time in v, that is, t_1. In G, we use the cost functions c_{r} to find the shortest path. Only arcs in E_s have costs. Thus, the cost of P_G is $c(e) = T - t_1$.

Fastest. The optimality value of the temporal path p is the duration of P. This is equivalent to the transmission times plus the waiting times in P, that is,

$$\sum_{i=1}^{k} \lambda_k + \sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i).$$

In G, we use the cost function c_{fa} to find the shortest path. Only arcs in E_t and E_w have costs. That is, $a_i \in E_t$ have cost $c_{\mathrm{fa}}(a_i) = \lambda_i$ for $i \in [k]$ and $b_i \in E_w$ have costs $c_{\mathrm{fa}}(b_i) = t_{i+1} - (t_i + \lambda_i)$ for $i \in [k-1]$. Thus, the cost of P_G are

$$\sum_{i=1}^{k} c_{\mathrm{fa}}(a_i) + \sum_{i=1}^{k-1} c_{\mathrm{fa}}(b_i) = \sum_{i=1}^{k} \lambda_k + \sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i).$$

Shortest. The optimality value of the temporal path P is the sum of transmission times, that is, $\sum_{i=1}^{k} \lambda_i$. In G, we use the cost function c_{s} to find the shortest path. Only arcs in E_t have cost. That is, $a_i \in E_t$ have cost $c_{\mathrm{s}}(a_i) = \lambda_i$ for $i \in [k]$. Hence, the cost of P_G is

$$\sum_{i=1}^{k} c_{\mathrm{s}}(a_i) = \sum_{i=1}^{k} \lambda_k.$$

Minimum Hop-Count. The optimality value of the temporal path P is the number of time-arcs in P, that is, k. In G, we use the cost function c_{h} to find the shortest path. Only arcs in E_t have cost. That is, $a_i \in E_t$ have cost $c_{\mathrm{s}}(a_i) = 1$ for $i \in [k]$. Hence, the cost of P_G is

$$\sum_{i=1}^{k} c_{\mathrm{s}}(a_i) = \sum_{i=1}^{k} 1 = k.$$

Cheapest. The optimality value of the temporal path P is the costs of time-arcs in P, that is, $\sum_{i=1}^{k} c(e_i)$. In G, we use the cost function c_{c} to find the shortest path. Only arcs in E_t have cost. That is, $a_i \in E_t$ have cost $c_{\mathrm{c}}(a_i) = c(e_i)$ for $i \in [k]$. Hence, the cost of P_G is

$$\sum_{i=1}^{k} c_{\mathrm{c}}(a_i) = \sum_{i=1}^{k} c(e_i).$$

Minimum Waiting Time. The optimality value of the temporal P is the waiting times, that is,

$$\sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i).$$

In G, we use the cost functions c_c to find a the shortest path. Only arcs in E_w have costs. That is, $b_i \in E_w$ have cost $c_{\mathrm{w}}(b_i)t_{i+1} - (t_i + \lambda_i)$ for $i \in [k-1]$. Thus, the sum of cost of P_G is

$$\sum_{i=1}^{k-1} c_{\mathrm{w}}(b_i) = \sum_{i=1}^{k-1} t_{i+1} - (t_i + \lambda_i).$$

We have shown that there is a temporal walk $\mathcal{G}$ from v to w with optimality value c^* if and only if there is a path in G from v_s to w_e with cost c^*. This completes the proof of Lemma 4.1.10 that states the following: There is an optimal temporal walk from v to w in $\mathcal{G}$ with optimality value c^* if and only if there exists a shortest path from v_s to v_e in G with minimum cost c^*. □

It remains to show the running time of Transformation 3 and the size of the resulting static graph.

Lemma 4.1.11. *Transformation 3 runs in $O(M^2)$ time. By applying Transformation 3 for the resulting static graph it holds that $|V_G| \in O(n + M)$ and $|E_G| \in O(M^2)$.*

Proof. Let $\mathcal{G} = (V, E, [T], \beta)$ be a temporal graph. For the reduction, we first have to find for every $v \in V$ the sorted lists of τ_v^- and τ_v^+. This takes $O\big(M \log M\big)$ time. Now, we have to construct the directed static graph G. We therefore create the following vertices:

- $|V_t^-| \in O(n\tau_{\max}^-) \subseteq O(M)$
- $|V_t^+| \in O(n\tau_{\max}^+) \subseteq O(M)$

- $|V_s| \in O(n)$
- $|V_e| \in O(n)$

and the following arcs:

- $|E_s| \in O(n\tau^+_{\max}) \subseteq O(M)$
- $|E_e| \in O(n\tau^-_{\max}) \subseteq O(M)$
- $|E_{s,e}| \in O(n)$
- $|E_w| \in O(M \min\{\beta_{\max}, \tau^+_{\max}\})$
- $|E_t| \in O(M)$.

Due to the sorted lists τ_v^- and τ_v^+ for $v \in V$, all these insertions can be done in time $O(n + M \min\{\beta_{\max}, \tau^+_{\max}\})$. Hence, the construction runs in

$$O\big(n + M \min\{\beta_{\max}, \tau^+_{\max}\} + M \log M\big)$$

time. For the resulting graph $G = (V_G, E_G)$, it holds that

$$|V_G| \in O(n + M) \text{ and } |E_G| \in O(M \min\{\beta_{\max}, \tau^+_{\max}\}).$$

□

Remark. In the proof of Lemma 4.1.11, it gets clear that if $\beta_{\max}$ is a constant, then the resulting graph G by Transformation 3 has only linear size with respect to the size of $\mathcal{G}$. If an incurable temporal graph is given, the simpler vertex-gadget introduced in the beginning of the subsection can be used. This results into a static graph G of linear size, as already shown in Wu et al. [Wu+16].

For a static graph $G = (V, E)$, solving Single-Source Shortest Path can be done by Dijkstra's algorithm in $O(|E| + |V| \log |V|)$ time if all arc costs are non-negative. All of our cost functions introduced in Transformation 3 have non-negative arc costs. Thus, we can conclude the following theorem:

Theorem 4.1.12. Single Source Optimal Walk *can be solved in* $O(M^2 \log M)$ *time by applying Transformation 3 and solving* Single-Source Shortest Path *on the resulting static graph with the respective cost function of the optimal walk variant.*

But given an acyclic, directed static graph, Dijkstra's algorithm has a running time in $O(|E|+|V|)$. If the transmission times are strictly greater than zero, then the resulting static graph by Transformation 3 is acyclic.

Observation 4.1.13. *Given a non-instantaneous temporal graph* $\mathcal{G} = (V, E, [T], \beta)$*, the resulting graph* G *by applying Transformation 3 to* $\mathcal{G}$ *is acyclic.*

We consider a topological ordering of the resulting static directed graph G. A topological ordering is a linear ordering of the vertices such that for all arcs (v, w) it holds that v comes before w in the ordering. There is a topological ordering for G if and only if G is acyclic.

Let $G = (V_G, E_G)$ with $V_G = V_t^- \cup V_t^+ \cup V_s \cup V_e$ according to Transformation 3. A possible topological ordering starts with the vertices in V_s. All these vertices only have out-going arcs. Afterwards, all vertices in V_t^- and V_t^+ can be ordered such that the time steps are non-decreasing and $v_t^- < v_t^+$ for $v \in V$, $t \in \tau_v^- \cap \tau_v^+$. The arcs between these vertices are always from a vertex v_t^+ to a vertex $w_{t'}^-$ with $v, w \in V$, $t \in \tau_v^+, t' \in \tau_w^-$, and $t < t'$ since the transmission times are greater than zero on the time-arcs. Or from a vertex v_t^- to a vertex $v_{t'}^+$ with $v \in V$, $t \in \tau_v^-, t' \in \tau_v^+$ if $t \leq t'$ since the non-decreasing time sequences. The ordering ends with the vertices in V_e. All these vertices only only have in-going arcs. Thus, the resulting directed static graph G has a topological ordering and consequently, is acyclic. As a consequence, Dijkstra's algorithm can solve Single Source Optimal Walk in $O(M^2)$ time by applying Transformation 3. The same does not hold for temporal graphs $\mathcal{G} = (V, E, [T], \beta)$ in general because there can exist a $t \in [T]$ for which G_t contains a cycle. In the resulting graph G, there exists a subgraph topological minor of G_t.

We have introduced Algorithm 1 and Algorithm 2 solving Single Source Optimal Walk for Foremost and Fastest in $O(M)$ and $O(M \log M)$ time respectively on instantaneous temporal graphs. Due to the Transformation 3, we can transform any temporal graph to a static graph without loosing any information on optimal walks.

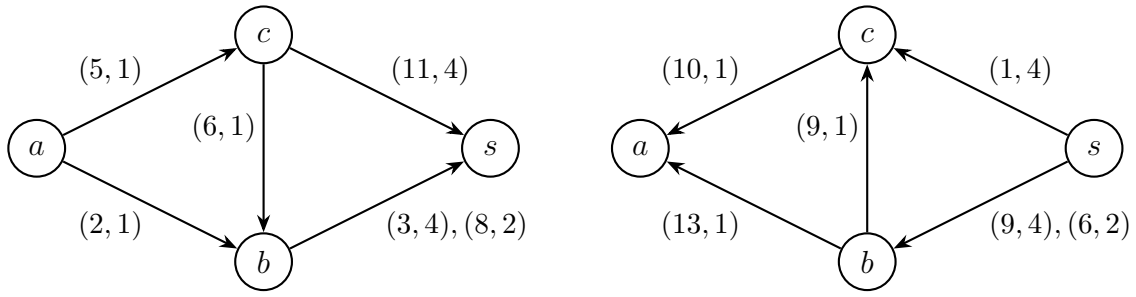
4.2 Single-Sink Optimal Walk

In this section, we look into solving Single Sink Optimal Walk. In directed static graphs, we already know that solving Single-Sink Shortest Path can be done by reversing the arcs in the graph and solving Single-Source Shortest Path on the resulting graph. We adapt this idea for temporal graphs. We will introduce a transformation of temporal graphs that reverses the time-arcs in a proper manner such that solving Single Source Optimal Walk on the resulting graph gives us the solution of Single Sink Optimal Walk in the original temporal graph. To reverse a time-arc, we

1. reverse the start- and endpoint,
2. adapt the time stamp, and
3. retain the transmission time.

Our transformation reverses all temporal walks in the graph and is done in the following way:

Transformation 4 (Reverse Temporal Graph). Let $\mathcal{G} = (V, E, [T], \alpha, \beta)$ be a temporal graph with a cost function $c\colon E \to \mathbb{N}$. We construct the *reverse temporal graph* $\mathcal{G}^r = (V, E^r, [T], \alpha, \beta)$ and $c_r\colon E^r \to \mathbb{N}$ with



(a) Temporal graph $\mathcal{G}$ with $\beta \equiv 4$ and $T = 16$

(b) Reverse temporal graph $\mathcal{G}^r$ of $\mathcal{G}$

Figure 4.4: A temporal graph with $\beta \equiv 4$ and lifetime $T = 16$ and its reverse temporal graph by Transformation 4.

(1) $E^r = \{(w, v, T - t - \lambda) \mid (v, w, t, \lambda) \in E\}$ and

(2) $c^r((v, w, T - t - \lambda, \lambda)) = c((v, w, t, \lambda))$ for all $(v, w, T - t - \lambda, \lambda) \in E^r$.

The resulting graph of Transformation 4 has obviously neither an increase in the number of vertices nor in the number of time-arcs. The transformation runs in linear time, and any sorted time-arcs list can be maintained in $O(M + M \log \min\{\lambda_{\max}, M\})$ time in a similar procedure as shown in Lemma 3.1.5. But most importantly, it maintains all temporal walks reversely. See Fig. 4.4 for an example of Transformation 4.

Example. Fig. 4.4 shows a temporal graph $\mathcal{G}$ (Fig. 4.4(a)) and its reverse temporal graph $\mathcal{G}^r$ (Fig. 4.4(b)). Consider the temporal walk $P = \big((a, b, 2, 1), (b, s, 3, 4)\big)$ from a to s. In the reverse temporal graph $\mathcal{G}^r$, the reverse temporal walk exists from s to a: $P^r = \big((s, b, 9, 4), (b, a, 13, 1)\big)$. The waiting time in vertex b is for both walks zero and the sum of transmission time is equal to 5 in both walks. The walk P is a foremost walk from a to s and P^r is a reverse-foremost walk from s to a. All these observations are not surprising and hold for every walk in $\mathcal{G}$ as we will show later in this section. Let us now have a look at the time-arc sequence $\big((a, c, 5, 1), (c, s, 11, 4)\big)$. This is not a valid temporal walk from a to s because it violates the maximum dwell time of 4 in vertex c. The same holds for the reverse time-arc sequence $\big((s, c, 1, 4), (c, a, 10, 1)\big)$ in $\mathcal{G}^r$. This is also not a temporal walk because we exceed the maximum dwell time of 4 in vertex c.

We will now show that our observations in the small example generally hold in temporal graphs and their reverse temporal graphs by Transformation 4. In the following discussion, for any time-arc $e \in E$ of an original temporal graph, we refer to the corresponding reverse time-arc with $e^r \in E^r$ in the reverse temporal graph.

Lemma 4.2.1. *Given a temporal Graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, Transformation 4 runs in linear time. For the resulting reverse temporal graph $\mathcal{G}^r = (V, E^r, [T], \alpha, \beta)$, the following holds: Let $v, w \in V$. A time-arc sequence $(e_1, \ldots, e_k)$ is a temporal walk from v to w in $\mathcal{G}$ if and only if $(e_k^r, \ldots, e_1^r)$ is a temporal walk from w to v in $\mathcal{G}^r$.*

Proof. The transformation can be applied in linear time: for every time-arc $e \in E$ we have to add a time-arc e_r to the the time-arc set E^r. The rest of the elements remain equal to the original temporal graph.

In the next step, we want to show that all temporal walks are maintained. Assume that the time-arc sequence $(e_1, \dots, e_k)$ is a temporal walk from v to w in $\mathcal{G}$ with $e_i = (v_i, w_i, t_i, \lambda_i) \in E$ for all $i \in [k]$. Recall that $v = v_1$, $w = w_k$, and for all $i \in [k-1]$ it holds that $t_i + \lambda_i + \alpha(w_i) \leq t_{i+1} \leq t_i + \lambda_i + \beta(w_i)$ and $v_i = w_{i+1}$. Thus, we already know that the sequence of vertices in $(e_k^r, \dots, e_1^r)$ is valid starting in w and ending in v due to $e_i^r = (w_i, v_i, t_i^r := T - t_i - \lambda_i, \lambda_i)$ for each $i \in [k]$. It remains to show that for all $i \in \{2, \dots, k\}$ it holds that $t_i^r + \lambda_i + \alpha(v_i) \leq t_{i-1}^r \leq t_i^r + \lambda_i + \beta(v_i)$:

$$
\begin{aligned}
& t_i^r + \lambda_i + \alpha(v_i) && \leq t_{i-1}^r \\
\Leftrightarrow\ & T - t_i - \lambda_i + \lambda_i + \alpha(v_i) && \leq T - t_{i-1} - \lambda_{i-1} \\
\Leftrightarrow\ & T - t_i + \alpha(v_i) && \leq T - t_{i-1} - \lambda_{i-1} \\
\Leftrightarrow\ & -t_i + \alpha(v_i) && \leq -t_{i-1} - \lambda_{i-1} \\
\Leftrightarrow\ & t_{i-1} + \lambda_{i-1} && \leq t_i - \alpha(v_i) \\
\Leftrightarrow\ & t_{i-1} + \lambda_{i-1} + \alpha(v_i) && \leq t_i
\end{aligned}
$$

The last inequality holds since $(e_1, \dots, e_k)$ is a valid temporal walk. We also have to show that for all $i \in \{2, \dots, k\}$ it holds that $t_{i-1}^r \leq t_i^r + \lambda_i + \beta(v_i)$:

$$
\begin{aligned}
& t_{i-1}^r && \leq t_i^r + \lambda_i + \beta(v_i) \\
\Leftrightarrow\ & T - t_{i-1} - \lambda_{i-1} && \leq T - t_i - \lambda_i + \lambda_i + \beta(v_i) \\
\Leftrightarrow\ & T - t_{i-1} - \lambda_{i-1} && \leq T - t_i + \beta(v_i) \\
\Leftrightarrow\ & -t_{i-1} - \lambda_{i-1} && \leq -t_i + \alpha(v_i) \\
\Leftrightarrow\ & t_i - \beta(v_i) && \leq t_{i-1} + \lambda_{i-1} \\
\Leftrightarrow\ & t_i && \leq t_{i-1} + \lambda_{i-1} + \beta(v_i)
\end{aligned}
$$

This inequality also holds since $(e_1, \dots, e_k)$ is a temporal walk. Thus, the time-arc sequence $(e_k^r, \dots, e_1^r)$ is a temporal walk from w to v in $\mathcal{G}^r$. The reverse direction works analogously. □

Next, we have a look at the different optimal walk definitions. We will show that for two vertices v and s it holds that a foremost walk from v to s in the original graph is a reverse-foremost walk from s to v in the reverse temporal graph and vice versa. For all other optimal walk definitions it holds that an optimal walk from v to s in the original graph is an optimal walk from s to v in the reverse temporal graph. Thus, if we want to compute an optimal walk from each vertex to a sink vertex s, it is sufficient to compute an optimal walk from s to each vertex in the reverse temporal graph.

Lemma 4.2.2. *Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, the reverse temporal graph $\mathcal{G}^r = (V, E^r, [T], \alpha, \beta)$ of $\mathcal{G}$ constructed by Transformation 4, and two vertices $v, w \in V$, it holds that:*

(1) If $(e_1, \dots, e_k)$ is a foremost walk from v to w in $\mathcal{G}$, then $(e_k^r, \dots, e_1^r)$ is a reverse-foremost walk from w to v in $\mathcal{G}^r$,

(2) If $(e_1, \dots, e_k)$ is a reverse-foremost walk from v to w in $\mathcal{G}$, then $(e_k^r, \dots, e_1^r)$ is a foremost walk from w to v in $\mathcal{G}^r$, and

(3) A temporal walk $(e_1, \dots, e_k)$ is an optimal walk from v to w in $\mathcal{G}$ if and only if $(e_k^r, \dots, e_1^r)$ is an optimal walk with respect to the same criterion from w to v in $\mathcal{G}^r$ for Fastest, Shortest, Minimum Hop-Count, Cheapest, and Minimum Waiting Time.

Proof. By Lemma 4.2.1, it holds that $P = (e_1, \dots, e_k)$ is a temporal walk in $\mathcal{G}$ from a vertex v to a vertex w if and only if $P^r = (e_k^r, \dots, e_1^r)$ is a temporal walk in $\mathcal{G}^r$ from w to v. Recall that for any $e \in E$, $e^r \in E^r$ is the reversed time-arc of e.

To *(1)*: If P is a foremost walk, then $t(e_k) + \lambda(e_k)$ is the earliest arrival time from v to w. Assume towards a contradiction that P^r is not a reverse-foremost walk in $\mathcal{G}^r$. Then there exists a reverse-foremost walk $\hat{P}^r = (\hat{e}_l^r, \dots, \hat{e}_1^r)$ in $\mathcal{G}^r$ from w to v with departure time $t(\hat{e}_l^r) = T - t(\hat{e}_l) - \lambda(\hat{e}_l) > T - t(e_k) + \lambda(e_k) = t(e_k^r)$. But then the temporal walk $\hat{P} = (\hat{e}_1, \dots, \hat{e}_l)$ is a valid walk in $\mathcal{G}$ from v to w due to Lemma 4.2.1. It has an arrival time $t(\hat{e}_l) + \lambda(\hat{e}_l) < t(e_k) + \lambda(e_k)$—a contradiction to our assumption that $t(e_k) + \lambda(e_k)$ is the earliest arrival time.

To *(2)*: If P is a reverse-foremost walk, then $t(e_1)$ is the latest departure time from v to w. Assume towards a contradiction that P^r is not a foremost walk in $\mathcal{G}^r$. Then there exists a foremost walk $\hat{P}^r = (\hat{e}_l^r, \dots, \hat{e}_1^r)$ in $\mathcal{G}^r$ from w to v with earliest arrival time $t(\hat{e}_1^r) + \lambda(\hat{e}_1^r) = T - t(\hat{e}_1) < T - t(e_k) = t(e_1^r) + \lambda(e_1^r)$. But then the temporal walk $\hat{P} = (\hat{e}_1, \dots, \hat{e}_l)$ is a valid walk in $\mathcal{G}$ from v to w due to Lemma 4.2.1. It has a departure time $t(\hat{e}_1) > t(e_1)$—a contradiction to our assumption that P is a reverse-foremost walk.

To *(3)*: By Lemma 4.2.1, we know that for each P in $\mathcal{G}$ there exists a reverse walk P^r in $\mathcal{G}^r$. We show that P and P^r have the same optimality value:

Fastest. A fastest walk minimizes the difference between arrival and start time. The temporal walk P^r takes $t(e_1^r) + \lambda(e_1^r) - t(e_k^r)$ time. That is,

$$\begin{aligned} & t(e_1^r) + \lambda(e_1^r) - t(e_k^r) \\ = \; & \big(T - t(e_1) - \lambda(e_1)\big) + \lambda(e_1) - \big(T + t(e_k) + \lambda(e_k)\big) \\ = \; & -t(e_1) + t(e_k) + \lambda(e_k)\big) \end{aligned}$$

This is the time the temporal walk P needs.

Shortest. A shortest walk minimizes the sum of transmission times. The temporal walk P^r has an optimization value of $\sum_{i \in [k]} \lambda(e_i^r) = \sum_{i \in [k]} \lambda(e_i)$ which is the sum of transmission times in P.

Minimum Hop-Count. A minimum hop-count walk minimizes the number of time-arcs. The temporal walks P and P^r have the same number of time-arcs k.

Cheapest. A cheapest walk minimizes the sum of time-arc costs. The temporal walk P^r has an optimal value of $\sum_{i\in[k]} c^r(e_i^r) = \sum_{i\in[k]} c(e_i)$ which is the sum of costs in P.

Minimum Waiting time. A minimum waiting time walk minimizes the waiting times in the intermediate vertices. The temporal walk P^r has accumulated waiting time of

$$\begin{aligned}
&\sum_{i\in[k-1]} t(e_i^r) - \big(t(e_{i+1}^r) - \lambda(e_{i+1}^r)\big) \\
=\ &\sum_{i\in[k-1]} \big(T - t(e_i) - \lambda(e_i)\big) - \Big(\big(T - t(e_{i+1}) - \lambda(e_{i+1})\big) + \lambda(e_{i+1})\Big) \\
=\ &\sum_{i\in[k-1]} \big(T - t(e_i) - \lambda(e_i)\big) - \big(T - t(e_{i+1})\big) \\
=\ &\sum_{i\in[k-1]} t(e_{i+1}) - \big(t(e_i) + \lambda(e_i)\big)
\end{aligned}$$

This is the accumulated waiting time of P.

Hence, P is an optimal walk in $\mathcal{G}$ if and only if P^r is an optimal walk in $\mathcal{G}^r$ for Fastest, Shortest, Minimum Hop-Count, Cheapest, and Minimum Waiting Time. □

We can conclude this section with the following theorem:

Theorem 4.2.3. *Given a temporal graph $\mathcal{G} = (V, E, [T], \alpha, \beta)$, we can solve* Single Sink Optimal Walk *for*

1. *Foremost in the same time as solving* Single Source Reverse-Foremost Walk,
2. *Reverse-Foremost in the same time as solving* Single Source Foremost Walk, *and*
3. *Fastest, Shortest, Minimum Hop-Count, Cheapest, and Minimum Waiting Time in the same time as solving* Single Source Optimal Walk *for the specific definition.*

Next, we will investigate finding all-pairs optimal walks in temporal graphs.

4.3 All-Pairs Optimal Walk

This section is devoted to solve All-Pairs Optimal Walk. Our approach is to adapt the famous Floyd-Warshall algorithm [Flo62; War62]—a dynamic programming algorithm for solving All-Pair Shortest Walk on static graphs without negative cycles. This algorithm compares all possible paths to compute a shortest path of all pairs of vertices. It exploits two different properties of shortest walks in static graphs without negative cycles:

(1) there exists a shortest walk which is a path and

(2) every subpath of a shortest path is a shortest path.

Due to the observation (1), using the idea of the Floyd-Warshall algorithm on temporal graphs seems impossible. As stated in Observation 3.1.6, it can happen that all temporal walks between two vertices contain a cycle if the dwell times are bounded. Also for the definition of Minimum Waiting Time, we know that there could exist a unique optimal walk containing cycles even if the maximum dwell time is unbounded, as stated in Observation 3.2.4. Consequently, we concentrate on incurable temporal graphs and on all optimal walk definitions excluding Minimum Waiting Time. In this model, we know by Lemma 3.2.5 that there always exists an optimal walk which is a path. Even though observation (2) does not hold for these optimal paths, we will show how we can work around that in our adaptation of the Floyd-Warshall algorithm.

We will start with the optimal walk definition Foremost. For this definition, we can assume by Lemma 3.2.6 that each prefix path of a foremost path is also a foremost path. We will introduce a dynamic program based on the Floyd-Warshall algorithm that computes the foremost path for each pair of vertices and for any possible departure time. This extra computation is used to overcome the fact that not every postfix-path of the foremost path is a foremost path. We at least know that every postfix-path of a foremost path is a foremost path for its departure time as shown in Lemma 3.2.7. In the resulting table of this program, we will find the optimal solution for Fastest and Reverse-Foremost as well.

Foremost, Reverse-Foremost & Fastest. In our adaptation of the Floyd-Warshall algorithm we compute not only for all vertex pairs $v, w \in V$ a foremost path from v to w, but do so for every possible start time. Of course, there can only be different foremost paths at any time step where v has an out-going time-arc, that is, $t \in \tau_v^+$. Thus, we compute a foremost path from v to w starting earliest at time step $t \in \tau_v^+$. Consequently, it also computes the reverse-foremost path and the fastest path. This is based on the following observations:

- Reverse-Foremost: The latest time step $t \in \tau_v^+$ such that there exists a foremost path from v to w starting earliest at time t is a reverse-foremost path (see Observation 3.2.2)

- Fastest: Choosing a time step $t \in \tau_v^+$ such that the difference between t and the earliest-arrival time of a foremost path from v to w starting earliest at time t is minimized will lead us to a fastest path (see Lemma 3.2.3).

The approach is to compare all foremost paths of all pairs of vertices and all possible starting times. The corresponding algorithm—Algorithm 3—works as follows: Let $\mathcal{G} = (V, E, [T])$ be a temporal graph where $V = \{v_1, v_2, \ldots, v_n\}$. After the k-th run of the outer-most for-loop (Line 6), Algorithm 3 compared for $v, w \in V$ and $t \in \tau_v^+$ all temporal

Algorithm 3: All-Pairs Optimal Walk for Foremost, Reverse-Foremost, and Fastest on temporal graphs with unbounded dwell time

```
1  function TempFloydWarshall(𝒢):
       /* Initialization for all v, w ∈ V and t ∈ τ_v^+ - see Lemma 4.3.1 */
2      for v ∈ V, t ∈ τ_v^+ do
3          d(v, v, t, 0) = t
4      for v, w ∈ V, t ∈ τ_v^+ with v ≠ w do
5          d(v, w, t, 0) = min({t' + λ' | (v, w, t', λ') ∈ E ∧ t ≤ t'} ∪ {∞})
6      for v_k ∈ {v_1, ..., v_n} do
           /* Computing the earliest arrival time from v to w starting
              earliest at time step t using the vertices {v_1, ..., v_k}   */
7          for v, w ∈ V with w ≠ v_k do
8              for t ∈ τ_v^+ do
9                  d(v, w, t, k) = min{d(v, w, t, k − 1), d(v_k, w, min{t' ∈ τ_{v_k}^+ |
                     d(v, v_k, t, k − 1) ≤ t'}, k − 1)}
10     return d
```

paths from v to w starting earliest at time t using only intermediate vertices $\{v_1, \ldots, v_k\}$. Thus, the table entry in d contains the following information:

$$d(v, w, t, k) := \text{earliest arrival time from } v \text{ to } w \text{ starting earliest at time step } t \text{ using only the vertices } \{v_1, \ldots, v_k\} \text{ as intermediate vertices.}$$

In the next loop-iteration of the outer-most for-loop (Line 6), the algorithm for any vertex-pair $v, w \in V$ (Line 7) and time step $t \in \tau_v^+$ (Line 8) computes the earliest arrival time using the intermediate vertices $\{v_1, \ldots, v_{k+1}\}$. For each pair v, w and each time step t, one of two statements holds:

1. A path does not use v_{k+1}.
   Then, $d(v, w, t, k)$ contains the earliest arrival time of any path using intermediate vertices $\{v_1, \ldots, v_{k+1}\}$.

2. A path uses v_{k+1}.
   Then, the path goes from v to v_{k+1} starting earliest in time t using only intermediate vertices $\{v_1, \ldots, v_k\}$ and then the path goes from v_{k+1} to w starting earliest at time $d(v, v_{k+1}, t, k)$ using only intermediate vertices $\{v_1, \ldots, v_k\}$.

Thus, Algorithm 3 computes the earliest arrival time by looking up the earliest arrival time from v to v_{k+1} starting at time t, that is, $d(v, v_{k+1}, t, k)$, and then, the earliest arrival time from v_k to w starting earliest in time $d(v, v_{k+1}, t, k)$, that is, $d(v_{k+1}, w, \min\{t' \in \tau_{v_{k+1}}^+ \mid d(v, v_{k+1}, t, k) \leq t'\}, k)$. Both of these entries are computed in the previous loop-iteration. The last value returns the earliest arrival from v to w if using the

vertex v_{k+1}. Now, Algorithm 3 compares the earliest arrival time using only vertices $\{v_1, \dots, v_k\}$ to the earliest arrival time when using the additional vertex v_{k+1}. The table entry $d(v, w, t, k+1)$ is set to the minimum of both values (Line 9).

After Algorithm 3 is conducted once, the table entry $d(v, w, t, n)$ contains the earliest arrival time from v to w starting earliest at time step t when using all vertices and, thus, the earliest arrival time in $\mathcal{G}$. Recall that n is the number of vertices in the temporal graph. The solution of Foremost, Reverse-Foremost, and Fastest can be found in the final table. For every $v, w \in V$, we find the optimal value of a temporal path in the following way:

- Foremost: $d(v, w, \min \tau_v^+, n)$,
- Reverse-Foremost: $\max\{t \in \tau_v^+ \mid d(v, w, t, n) < \infty\}$, and
- Fastest: $\min\{d(v, w, t, n) - t \mid t \in \tau_v^+\}$.

After introducing Algorithm 3 in detail, we next show that it solves All-Pairs Optimal Walk and that it runs $O(n^2 M)$ time. But before, we prove that the initialization step in Lines 4 and 5 runs $O(nM)$ time:

Lemma 4.3.1. *Given an incurable temporal graph $\mathcal{G} = (V, E, [T])$, setting $d(v, w, t) := \min\left(\{t' + \lambda' \mid (v, w, t', \lambda') \in E \wedge t \leq t'\} \cup \{\infty\}\right)$ for all $v, w \in V, t \in \tau_v^+$ in Lines 4 and 5 takes $O(nM)$ time.*

Proof. For every vertex pair $v, w \in V$ we can store a list of all time-arcs from v to w:

$$[(t_1, \lambda_1), (t_2, \lambda_2), \dots, (t_k, \lambda_k)]$$

We create this list for every vertex pair by going through the sorted time-arc list once. We know that $t_1 \leq t_2 \cdots \leq t_k$. We assume that all t_i with $i \in [k]$ are pairwise distinct. If not, then we can delete all entries with the same time stamp except the one with the smallest λ. This can be done in linear time with respect to the length of the list. Then, we additionally store the arrival times in w:

$$L_{v,w} = [(t_1, a_1 = t_1 + \lambda_1, \lambda_1), (t_2, a_2 = t_2 + \lambda_2, \lambda_2), \dots, (t_k, a_k = t_k + \lambda_k, \lambda_k)]$$

For each $i \in [k]$, we want to set a_i to the earliest arrival time in w when taking a v-w time-arc with time stamp at least t_i and λ_i to the smallest transmission time to meet the arrival time. In the end, it must hold that $a_1 \leq a_2 \leq \cdots, \leq a_k$. Therefore, we iterate backwards through the list $L_{v,w}$. For the entry $(t_k, a_k = t_k + \lambda_k, \lambda_k)$ we know that $t_k + \lambda_k$ is the earliest arrival time in w when starting earliest in time step t_k because there is no later time-arc to take. For every (t_i, a_i, λ_i) with $i \in [k-1]$—starting with $(t_i, a_i, \lambda_i) := (t_{k-1}, a_{k-1}, \lambda_{k-1})$ and decreasing i in each iteration—we do the following:

1. If $a_i \geq a_{i+1}$, then set $a_i := a_{i+1}$ and $\lambda_i := \lambda_{i+1}$
   because a_{i+1} is the earliest arrival time when starting earliest at time step t_i. Also, the transmission time $\lambda_i \geq \lambda_{i+1}$ because we meet the arrival time $a_i \geq a_{i+1}$ with a v-w time-arc with a later time stamp than t_i.

2. Go to the next entry (t_{i-1}, a_{i-1}).

Thus, by iterating backwards through the list, we update all time steps t_i with $i \in [k]$ to the earliest arrival time in w starting earliest in time t_i with the smallest transmission time. This runs in $O(|d_{v,w}|)$ where $|d_{v,w}|$ is the number of time-arcs from v to w. For filling the table of Algorithm 3, we do not need the knowledge about the smallest transmission time. But this information is needed in Lemma 4.3.3. Now, we have to go through the ordered list of τ_v^+ of the vertex pair v, w and update the entries according to our list $L_{v,w}$. This runs in $O(|\tau_v^+|)$ time. Thus, the whole procedure runs in $O(nM)$ time due to

$$\begin{aligned} & O\big(M + \sum_{v\in V}\sum_{w\in V} d_{v,w} + \sum_{v\in V}\sum_{w\in V} |\tau_v^+|\big) \\ \subseteq\ & O\big(M + M + \sum_{w\in V}\sum_{v\in V} d_v\big) \\ \subseteq\ & O\big(M + nM\big) \\ =\ & O\big(nM\big). \end{aligned}$$

□

Together with Lemma 4.3.1, we can finally prove the following theorem:

Theorem 4.3.2. *Algorithm 3 solves* All-Pairs Optimal Walk *on incurable temporal graphs for* Foremost, Reverse-Foremost, *and* Fastest *in* $O(n^2M)$ *time.*

Proof. We first prove the running time of the algorithm and then show its correctness.

Running Time. In the initialization, the algorithm first sets $d(v, v, t, 0) = t$ for every $v \in V$, $t \in \tau_v^+$ in Lines 2 and 3. This takes at most $O(M)$ time. For every $v, w \in V$ there are $|\tau_v^+|$ entries to fill in Lines 4 and 5. This runs in $O(nM)$ time by Lemma 4.3.1. After the initialization, we only have to look at the number of for-loops: for each vertex $v_k \in V$ and every vertex-pair $v, w \in V$, Algorithm 3 conducts $|\tau_v^+|$ table entries updates. Now it gets a bit tricky. For each update we have to find $\min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t, k) \leq t'\}$ in Line 9. Naively, that takes $O(\log |\tau_{v_k}^+|)$ time by binary search on the sorted array on $\tau_{v_k}^+$ for each $t \in \tau_v^+$. But, if we go through an sorted array $[t_1, \ldots, t_{|\tau_v^+|}]$ of τ_v^+ in Line 6, then we know that

$$\min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t_1, k) \leq t'\} \leq \cdots \leq \min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t_{|\tau_v^+|}, k) \leq t'\}$$

due to Observation 3.2.8. The earliest arrival time of any path from v to w starting earliest at time t—$d(v, v_k, t, k)$— is smaller or equal to the earliest arrival time of any path from v to w starting earliest at time t'—$d(v, v_k, t', k)$—if and only if $t < t'$. We can apply the following procedure:

1. For $t_1 \in \tau_v^+$, we go through the sorted array of $\tau_{v_k}^+$ until we find the smallest $t \in \tau_{v_k}^+$ such that $d(v, w, t_1, k) \leq t$, that is, $\min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t_1, k) \leq t'\}$. We mark the position of t in the list of $\tau_{v_k}^+$.

2. For $t_2 \in \tau_v^+$, we know that $t \leq \min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t_2, k) \leq t'\}$. Thus, we can start at our marker on the array of $\tau_{v_k}^+$ to find the proper value.

This can be done for all time-steps in τ_v^+ consecutively. Thus, for the loop on τ_v^+ in Line 8, we only have to iterate the array of $\tau_{v_k}^+$ once. All other operations are table look-ups, which can be done in constant time. Hence, Algorithm 3 runs in $O(n^2M)$ time due to

$$
\begin{aligned}
& O\Big(\sum_{v_k\in V}\sum_{v\in V}\sum_{w\in V}\Big(\sum_{t\in\tau_v^+} 1\Big) + |\tau_{v_k}^+|\Big) \\
= \; & O\Big(\sum_{v_k\in V}\sum_{v\in V}\sum_{w\in V}\sum_{t\in\tau_v^+} 1 + \sum_{v_k\in V}\sum_{v\in V}\sum_{w\in V} |\tau_{v_k}^+|\Big) \\
= \; & O\Big(\sum_{v_k\in V}\sum_{w\in V}\sum_{v\in V} |\tau_v^+| + \sum_{v\in V}\sum_{w\in V}\sum_{v_k\in V} |\tau_{v_k}^+|\Big) \\
\subseteq \; & O\Big(\sum_{v_k\in V}\sum_{w\in V}\sum_{v\in V} |d_v| + \sum_{v\in V}\sum_{w\in V}\sum_{v_k\in V} |d_{v_k}|\Big) \\
= \; & O\Big(\sum_{v_k\in V}\sum_{w\in V} M + \sum_{v\in V}\sum_{w\in V} M\Big) \\
= \; & O\big(n^2M + n^2M\big) \\
= \; & O\big(n^2M\big).
\end{aligned}
$$

Correctness. We will show that after the k-th iteration of the outer-most for-loop, a table entry $d(v, w, t, k)$ contains the earliest arrival time when using only vertices $\{v_1, \ldots, v_k\}$ as intermediate vertices. This can be done by induction on the number of iteration $k \in \{0, 1, \ldots, n\}$:

Before entering the outer-most for-loop, that is, $k = 0$, the table entries are set to $d(v, w, t, 0) = \min\big(\{t' + \lambda' \mid (v, w, t', \lambda') \in E \wedge t \leq t'\} \cup \{\infty\}\big)$. Thus, it is set to the correct earliest arrival time using no intermediate vertices. Let us now assume that after the k-th iteration, a table entry $d(v, w, t, k)$ contains the earliest arrival time using only intermediate vertices $\{v_1, \ldots, v_k\}$. In the $(k+1)$-th iteration, the table entry $d(v, w, t, k+1)$ should be set to the earliest arrival time using intermediate vertices $\{v_1, \ldots, v_{k+1}\}$. Therefore, the earliest arrival time using vertex v_{k+1} is computed:

1. the algorithm looks up $d(v, v_{k+1}, t, k)$—the earliest arrival time from v to v_{k+1} using vertices $\{v_1, \ldots, v_k\}$, due to the observation that a prefix-path of a foremost path is a foremost path (Lemma 3.2.6),

2. then, the algorithm looks up $d(v_k, w, \min\{t' \in \tau_{v_k}^+ \mid d(v, v_k, t, k) \leq t'\}, k)$—the earliest arrival time from v_{k+1} to w starting earliest in time $d(v, v_k, t, k)$, due to the observation that every postfix-path of a foremost path is a foremost path for its departure time (Lemma 3.2.7).

Hence, $d(v_{k+1}, w, \min\{t' \in \tau_{v_k}^+ \mid d(v, v_{k+1}, t, k) \leq t'\}, k)$ is the earliest arrival time from v to w via v_{k+1}. The table entry is set to

$$d(v, w, t, k+1) := \min\{d(v, w, t, k), d(v_{k+1}, w, \min\{t' \in \tau_{v_k}^+ \mid d(v, v_{k+1}, t, k) \leq t'\}, k)\},$$

the earliest arrival time using only the intermediate vertices $\{v_1, \dots, v_{k+1}\}$.

After the n-th iteration, $d(v, w, t, n)$ contains the earliest arrival time using intermediate vertices $\{v_1, \dots, v_n\} = V$, and hence, the earliest arrival time in $\mathcal{G}$. □

Remark. A closer look into the running time analysis shows that Algorithm 3 runs in $O(n^3)$ for static graphs and, thus, has an identical running time to the original Floyd-Warshall algorithm.

We continue with the three remaining optimal walk definitions where there always exists an optimal path in an incurable temporal graph: Shortest, Minimum Hop-Count, and Cheapest.

Shortest, Minimum Hop-Count & Cheapest. These three optimal walk definitions have in common that their optimization costs is the sum of the definition-specific time-arcs values of a temporal walk. The problem is thereby that no subpath of an optimal path has to be optimal. But every subpath is optimal within a certain time interval as shown by Lemma 3.2.11. Thus, for each pair of vertices v and w we will compute an optimal path for any possible start and arrival time. Of course there can only be different optimal paths from v to w for any time step where there is an out-going time-arc at v and an in-going time-arc in w, that is, for every starting time $t_v \in \tau_v^+$ and every arrival time $t_w \in \tau_w^-$. We compare all optimal paths of all pairs of vertices and all possible starting times and arrival times.

The corresponding algorithm—Algorithm 4—works as follows: Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph where $V = \{v_1, v_2, \dots, v_n\}$. After the k-th iteration of the outer-most for-loop (Line 6), the algorithms compared for $v, w \in V$, $t_v \in \tau_v^+$, and $t_w \in \tau_w^-$ all temporal path from v to w starting earliest at time t_v and arriving no later than t_w using only intermediate vertices $\{v_1, \dots, v_k\}$. Thus, the table entry in d contains the following information:

$$d(v, w, t_v, t_w, k) := \text{minimum sum of time-arc costs from } v \text{ to } w \text{ starting earliest at time step } t_v \text{ and arriving latest in } t_w \text{ using only the vertices } \{v_1, \dots, v_k\} \text{ as intermediate vertices.}$$

For Shortest the table entry is the minimum amount of transmission times, for Minimum Hop-Count it is the minimum number of time-arcs, and for Cheapest it is the minimum costs from v to w within time interval $[t_v, t_w]$ using only intermediate vertices $\{v_1, \dots, v_k\}$. In the next iteration of the outer-most for-loop (Line 6), the algorithm computes for any vertex-pair $v, w \in V$ (Line 7) and time steps $t_v \in \tau_v^+$, $t_w \in \tau_w^-$ (Line 8) an optimal path using the intermediate vertices $\{v_1, \dots, v_{k+1}\}$. For each pair $v, w \in V$ and $t_v \in \tau_v^+$, $t_w \in \tau_w^+$, one of two statements holds:

1. A path does not contain v_{k+1}.
   Then, $d(v, w, t_v, t_w, k)$ contains the minimum sum of time-arcs values of any path using intermediate vertices $\{v_1, \dots, v_{k+1}\}$.

Algorithm 4: All-Pairs Optimal Walk for Shortest, Minimum Hop-Count, and Cheapest on temporal graphs with unbounded dwell time

```
1  function TempFloydWarshallII(𝒢, opt):
       /* Initialization for all v, w ∈ V and t_v ∈ τ_v^+, t_w ∈ τ_w^-   */
2      for v ∈ V, t_v ∈ τ_v^+, t'_v ∈ τ_v^- do
3          d(v, v, t_v, t'_v, 0) = 0
       /* The symbol * is a place holder for the optimal walk
          definition---see below                                        */
4      for v, w ∈ V, t_v ∈ τ_v^+, t_w ∈ τ_w^- do
5          d(v, v, t_v, t'_v, 0) = Init*(v, w, t_v, t_w)
6      for v_k ∈ {v_1, ..., v_n} do
           /* Computing the optimal value of a path from v to w starting
              earliest at time step t_v and arriving latest in t_w using
              only the vertices {v_1, ..., v_k}                         */
7          for v, w ∈ V do
8              for t_v ∈ τ_v^+, t_w ∈ τ_w^- do
9                  d = d(v, w, t_v, t_w, k − 1)
10                 for t_{v_k} ∈ τ_{v_k}^- do
11                     d = min{d, d(v, v_k, t_v, t_{v_k}, k − 1) + d(v_k, w, min{t' ∈ τ_{v_k}^+ | t_{v_k} ≤
                           t'}, t_w, k − 1)}
12                 d(v, w, t_v, t_w, k) = d
13     return d
   /* Initialization for different optimal path definitions            */
14 function InitShortest(v, w, t_v, t_w):
15     return min({λ | (v, w, t', λ) ∈ E ∧ t_v ≤ t' ∧ t' + λ ≤ t_w} ∪ {∞})
16 function InitMinHopCount(v, w, t_v, t_w):
17     return min({1 | (v, w, t', λ) ∈ E ∧ t_v ≤ t' ∧ t' + λ ≤ t_w} ∪ {∞})
18 function InitCheapest(v, w, t_v, t_w):
19     return min({c((v, w, t', λ)) | (v, w, t', λ) ∈ E ∧ t_v ≤ t' ∧ t' + λ ≤ t_w} ∪ {∞})
```

2. A path uses v_{k+1}.
   Then, the path goes from v to v_{k+1} starting earliest in time t_v using intermediate vertices $\{v_1, \dots, v_k\}$ and then from v_{k+1} to w arriving not later than t_w using only intermediate vertices $\{v_1, \dots, v_k\}$. There is a variety of such paths. For every time step $t_{v_{k+1}} \in \tau^-_{v_{k+1}}$, there could exist a path that arrives in v_{k+1} not later than $t_{v_{k+1}}$ and continues in time step $\min\{t' \in \tau^+_{v_{k+1}} \mid t_{v_{k+1}} \leq t'\}$.

The optimal path using vertex v_{k+1} is computed by looking up for each $t_{v_{k+1}} \in \tau^-_{v_{k+1}}$ an optimal path from v to v_{k+1} starting earliest at time t_v, that is, $d(v, v_{k+1}, t_v, t_{v_{k+1}}, k)$. Then, the optimal path from v_k to w starting earliest in time $t_{v_{k+1}}$ and arriving not later than time t_w, that is, $d(v_{k+1}, w, \min\{t' \in \tau^+_{v_{k+1}} \mid t_{v_{k+1}} \leq t'\}, t_w, k)$ is looked up.

The minimum of

$$d(v, v_{k+1}, t_v, t_{v_{k+1}},) + d(v_{k+1}, w, \min\{t' \in \tau^+_{v_{k+1}} \mid t_{v_{k+1}} \leq t'\}, t_w, k)$$

for any $t_{v_{k+1}} \in \tau^-_{v_{k+1}}$ is taken (Line 11). This value is the optimal value of a path if the vertex v_{k+1} has to be used. Now, both values—the value of $d(v, w, t_v, t_w, k)$ and the current result—are compared. The table entry $d(v, w, t_v, t_w, k+1)$ is updated to the minimum of these two values (Line 12).

After the n-th iteration of the outer-most for-loop, the optimal path of any pair of vertices $v, w \in V$ can be found in the resulting table:

$$d(v, w, \min \tau^+_v, \max \tau^-_w, n)$$

After explaining Algorithm 4, we next show the correctness of the algorithm and the running time complexity of $O(nM + M^3)$. To this end, we examine the running time of the initialization steps:

Lemma 4.3.3. *Given an incurable temporal $\mathcal{G} = (V, E, [T])$, initializing $d(v, w, t_v, t_w, 0)$ for all $v, w \in V$, $t_v \in \tau^+_v$, and $t_w \in \tau^-_w$ in Lines 4 and 5 takes $O(M^2)$ time.*

Proof. First, recall how Algorithm 4 initializes the table entry $d(v, w, t_v, t_w, 0)$ for the three definitions:

Shortest
$$d(v, w, t_v, t_w, 0) := \min\left(\{\lambda \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}\right)$$

Minimum Hop-Count:
$$d(v, w, t_v, t_w, 0) := \min\left(\{1 \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}\right)$$

Cheapest:
$$d(v, w, t_v, t_w, 0) := \min\{c((v, w, t', \lambda)) \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}$$

To get this initialization, we first have to store for every vertex pair $v, w \in V$ a list of all time-arcs from v to w:

$$[(t_1, \lambda_1), (t_2, \lambda_2), \dots, (t_k, \lambda_k)]$$

We can create this list for every vertex pair by going through the sorted time-arc list once. We know that $t_1 \leq t_2 \cdots \leq t_k$ due to the sorted time-arc list. Now, we prepare the list for the different definitions in separate ways before we can efficiently initialize the table entries:

Shortest & Minimum Hop-Count. For these two definitions, we create a sorted list of all time steps in which there is a v-w-time-arc with the earliest arrival time in w and the minimum transmission time to meet the arrival time. This can be done in the same manner as described in Lemma 4.3.1. Let

$$L_{v,w} = [(t_1, a_1, \lambda_1), (t_2, a_2, \lambda_2), \ldots, (t_k, a_k, \lambda_k)]$$

be such a sorted list. It holds that $t_1 \leq t_2 \leq \cdots \leq t_k$ and $a_1 \leq a_2 \leq \cdots, \leq a_k$. It also holds for any list entry (t, a, λ) in $L_{v,w}$ that the transmission time λ is the smallest transmission time when starting earliest in time step t and arriving in time step a. Creating this list takes $O(|d_{v,w}|)$ time where $|d_{v,w}|$ is the number of time-arcs from v to w.

Cheapest. For this definition, we create a sorted list of all departure times d and arrival times a of all v-w-time-arcs together with the cheapest time-arc within $[d, a]$. We first add the costs of the time-arcs to the tuple and replace the transmission time within the triple with the arrival time in w.

$$[(t_1, a_1 := t_1 + \lambda_1, c_1), (t_2, a_2 := t_2 + \lambda_2, c_2), \ldots, (t_k, a_k := t_k + \lambda_k, c_k)]$$

In the next step, we sort the time-arcs with the same time stamp by their arrival time in ascending order. Next, we sort out all time-arcs where there exists another time-arc with the same time stamp and an earlier arrival time and lower cost. The procedure of sorting and deleting takes at most $O(|d_{v,w}| \log |d_{v,w}|)$ time. We end up with a sorted list

$$L_{v,w} := [(t_1, a_1, c_1), (t_2, a_2, c_2), \ldots, (t_k, a_k, c_k)]$$

We know that $t_1 \leq t_2 \leq \cdots \leq t_k$ and if $t_i = t_{i+1}$, then $a_i < a_{i+1}$ and $c_i > c_{i+1}$ for any $i \in [k-1]$. Now, we iterate through the list $L_{v,w}$ starting at the end of the list. For the entry (t_k, a_k, c_k) we know that c_k is the cheapest cost to arrive in w when starting in time step t_k and arriving in time step a_k because there is no other time-arc to take. For every (t_i, a_i, c_i) with $i \in [k-1]$—starting with $(t_i, a_i, c_i) := (t_{k-1}, a_{k-1}, c_{k-1})$ and decreasing i in each iteration— we do the following:

1. If $a_i \geq a_{i+1}$ and $c_i > c_{i+1}$, then set $a_i := a_{i+1}$ and $c_i := c_{i+1}$
   because a_{i+1} is an earlier arrival time with cheaper cost when starting earliest at time step t_i.

2. Go to the next entry $(t_{i-1}, a_{i-1}, c_{i-1})$.

Thus, by iterating backwards through the list, we update all time steps t_i with $i \in [k]$ with the cheapest earliest arrival time. Last, we iterate once more over the list $L_{v,w}$ and delete all duplication if they exist and then sort the whole list by arrival times in ascending order. This takes $O(|d_{v,w}| \log |d_{v,w}|)$ time. Thus, the whole procedure takes $O(|d_{v,w}| \log |d_{v,w}|)$ time.

Initialization. Now, we can fill the table of size $|\tau_v^+||\tau_w^-|$ with the help of $L_{v,w}$. We iterate through the sorted list of τ_v^+ in ascending order. For a $t_v \in \tau_v^+$, we first delete all list entries $(t, a, \lambda|c)$ of $L_{v,w}$ where $t < t_v$ and end up with a list

$$L_{v,w} := [(t_1, a_1, \lambda_1|c_1), (t_2, a_2, \lambda_2|c_2), \dots, (t_l, a_l, \lambda_l|c_l)]$$

This takes $O(|d_{v,w}|)$ time. The deleted list entries will not be needed for any following time step in τ_v^+ since all time-arcs are considered in ascending departure times. If $L_{v,w}$ is empty, then for all $t_w \in \tau_w^-$ the table entries $d(v, w, t_v, t_w, 0)$ are set to ∞ because there are no v-w time-arcs that starts earliest in time t_v. Else, we can iterate forward through the sorted array $[t_1^w, \dots, t_{k_w}^w]$ of τ_w^-.

1. If for t_i^w, it holds that $t_i^w < t_v$ or $t_i^w < a_1$, then $d(v, w, t_v, t_w, 0) := \infty$.
   There is no time-arc from v to w that starts earliest at time t_v and arrives no later than t_i^w.

2. Else, we look in $L_{v,w}$ for the largest earliest arrival time a_j such that $a_j \leq t_i^w$. We set the table entry
   - Shortest: $d(v, w, t_v, t_i^w, 0) := \min\{d(v, w, t_v, t_{i-1}^w), \lambda_j\}$.
   - Minimum Hop-Count: $d(v, w, t_v, t_i^w, 0) := 1$.
   - Cheapest: $d(v, w, t_v, t_i^w, 0) := \min\{d(v, w, t_v, t_{i-1}^w), c_j\}$

It holds that $d(v, w, t_v, t_1^w, 0) \geq d(v, w, t_v, t_2^w, 0) \geq \dots \geq d(v, w, t_v, t_{k_w}^w, 0)$. To fill all these entries, it is sufficient to iterate at most once over $L_{v,w}$ due to the ascending order of τ_w^-. Thus, it runs in $O(|\tau_v^+|(|\tau_w^-| + |d_{v,w}|)$ time. The whole procedure runs in $O(M^2)$ time due to

$$\begin{aligned}
& O\big(M + \sum_{v\in V}\sum_{w\in V} |d_{v,w}| \log |d_{v,w}| + |\tau_v^+|(|\tau_w^-| + |d_{v,w}|)\big) \\
\subseteq\ & O\big(M + \sum_{v\in V}\sum_{w\in V} |d_{v,w}| \log |d_{v,w}| + |d_v||d_w|\big) \\
=\ & O\big(M + \sum_{v\in V}\sum_{w\in V} |d_{v,w}| \log |d_{v,w}| + \sum_{v\in V}\sum_{w\in V} |d_v||d_w|\big) \\
\subseteq\ & O\big(M + M \log M + \sum_{v\in V} |d_v| \sum_{w\in V} |d_w|\big) \\
\subseteq\ & O\big(M \log M + M^2\big) \\
=\ & O\big(M^2\big).
\end{aligned}$$

□

With Lemma 4.3.3, we can finally prove the following theorem:

Theorem 4.3.4. *Algorithm 4 solves* All-Pairs Optimal Walk *on incurable temporal graphs for* Shortest, Minimum Hop-Count, *and* Cheapest *in* $O(nM + M^3)$ *time.*

Proof. We begin with proving the running time of the algorithm.

Running Time. In the initialization, the algorithm first sets $d(v, v, t_v, t_w, 0) = 0$ for every $v \in V$, $t_v \in \tau_v^+, t_w \in \tau_v^-$ in Lines 2 and 3. This takes at most $O(nM)$ time. In the initialization, for every $v, w \in V$ there are $|\tau_v^+||\tau_w^-|$ entries to fill. This runs in $O(M^2)$ time as shown in Lemma 4.3.3.

After the initialization, we only have to look at the number of for-loops: for each vertex $v_k \in V$ and every vertex-pair $v, w \in V$, Algorithm 4 conducts $|\tau_v^+||\tau_w^-|$ table entries updates. For each $t_w \in \tau_v^+, t_w \in \tau_w^-$, the algorithm has to iterate over all possible arrival times in v_k, that is, a sorted array $[t_1^{v_k}, \dots, t_l^{v_k}]$ of $|\tau_{v_k}^-|$. The algorithm looks up the optimal value $d(v, v_k, t_v, t_i^{v_k}, k)$ and $d(v_k, w, \min\{t' \in \tau_{v_k}^+ \mid t_i^{v_k} \leq t'\}, t_w, k)$ for each $i \in [l]$. To find $\min\{t' \in \tau_{v_k}^+ \mid t_i^{v_k} \leq t'\}$, Algorithm 4 has to iterate through $\tau_{v_k}^+$ only once for all arrival times in $\tau_{v_k}^-$ due to

$$\min\{t' \in \tau_{v_k}^+ \mid t_1^{v_k} \leq t'\} \leq \min\{t' \in \tau_{v_k}^+ \mid t_2^{v_k} \leq t'\} \leq \cdots \leq \min\{t' \in \tau_{v_k}^+ \mid t_l^{v_k} \leq t'\}$$

We can sum up the computation effort by:

$$\begin{aligned}
& O\Big(\sum_{v_k \in V}\sum_{v \in V}\sum_{w \in V}\sum_{t_v \in \tau_v^+}\sum_{t_w \in \tau_w^-}\Big(\sum_{t_k \in \tau_{v_k}^-} 1\Big) + |\tau_{v_k}^+|\Big) \\
= \; & O\Big(\sum_{v_k \in V}\sum_{v \in V}\sum_{w \in V} |\tau_v^+||\tau_w^-|(|\tau_{v_k}^-| + |\tau_{v_k}^+|)\Big) \\
\subseteq \; & O\Big(\sum_{v \in V}|\tau_v^+| \sum_{w \in V}|\tau_w^-| \sum_{v_k \in V}|d_{v_k}|\Big) \\
\leq \; & O\Big(\sum_{v \in V} d_v \sum_{w \in V} d_w \sum_{v_k \in V} d_{v_k}\Big) \\
= \; & O\big(M^3\big)
\end{aligned}$$

Hence, Algorithm 4 runs in $O(nM + M^3)$ time.

Correctness. We will show that after the k-th iteration of the outer-most for-loop, the table entry $d(v, w, t_v, t_w, 0)$ contains the optimal path value when using only vertices $\{v_1, \dots, v_k\}$ as intermediate vertices. This can be done by induction on the number of iterations $k \in \{0, 1, \dots, n\}$:

Before entering the most-outer for-loop, the table entries are set correctly to

- Shortest:
  $d(v, w, t_v, t_w, 0) := \min\big(\{\lambda \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}\big)$
- Minimum Hop-Count:
  $d(v, w, t_v, t_w, 0) := \min\big(\{1 \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}\big)$
- Cheapest:
  $d(v, w, t_v, t_w, 0) := \min\big(\{c((v, w, t', \lambda)) \mid (v, w, t', \lambda) \in E \wedge t_v \leq t' \wedge t' + \lambda \leq t_w\} \cup \{\infty\}\big)$

This is an optimal path value using no intermediate vertices.

Let us now assume that after the k-th iteration, $d(v, w, t_v, t_w, k)$ contains the optimal value using only intermediate vertices $\{v_1, \ldots, v_k\}$. In the $(k+1)$-st iteration, the table entry $d(v, w, t_v, t_w, k+1)$ should be set to the optimal path value using intermediate vertices $\{v_1, \ldots, v_{k+1}\}$. Therefore, the optimal path value using vertex v_{k+1} is computed. For any possible arrival time in v_{k+1}, that is, $t_{v_{k+1}} \in \tau^-_{v_{k+1}}$, the algorithm does the following:

1. the algorithm looks up $d(v, v_{k+1}, t_v, t_{v_{k+1}}, k)$—the optimal path value from v to v_{k+1} arriving in $t_{v_{k+1}}$ using vertices $\{v_1, \ldots, v_k\}$,
2. then, the algorithm looks up $d(v_{k+1}, w, \min\{t' \in \tau^+_{v_k} \mid t_{v_{k+1}} \leq t'\}, t_w, k)$—the optimal path value from v_{k+1} to w starting earliest in time $t_{v_{k+1}}$.

The value

$$c = \min_{t_{v_{k+1}} \in \tau^+_{v_{k+1}}} \{d(v, v_{k+1}, t_v, t_{v_{k+1}}, k) + d(v_{k+1}, w, \min\{t' \in \tau^+_{v_k} \mid t_{v_{k+1}} \leq t'\}, t_w, k)\}$$

is the optimal path value if v_{k+1} is on an optimal path. Every subpath of such an optimal path is an optimal path within its time interval as shown in Lemma 3.2.11. The optimal path using vertices $\{v_1, \ldots, v_{k+1}\}$ is either using v_{k+1}, that is, value c, or uses only the vertices $\{v_1, \ldots . v_k\}$, that is, $d(v, w, t_v, t_w, k)$. Thus, the table entry $d(v, w, t_v, t_w, k+1)$ is set to $\min\{d(v, w, t_v, t_w, k), c\}$. This is the optimal path value using intermediate vertices $\{v_1, \ldots, v_{k+1}\}$.

After the n-th iteration, $d(v, w, t_v, t_w, n)$ contains the optimal path value using intermediate vertices $\{v_1, \ldots, v_n\} = V$, and hence, the optimal path value in $\mathcal{G}$. □

We have introduced our adaptation of the Floyd-Warhsal algorithm to incurable temporal graphs. In Table 4.1, we compare the running time of Algorithm 3 and Algorithm 4 to the running time of the algorithms for Single Source Optimal Walk. The algorithms for All-Pairs Optimal Walk cannot compete with conducting n times the Single Source Optimal Walk algorithms. Note that with our algorithms we compute not only an optimal walk in the whole temporal graph, but also an optimal walk for any possible starting time and arrival time within the graph. Such information can be useful in the computation of radius, diameter, and closeness centrality in temporal graphs [KA12; PS11; San+11; Tan+13].

When computing a foremost path for each vertex $v \in V$ and each starting time $t \in \tau^+_v$, Algorithm 3 can compete with the Single Source Optimal Walk algorithm that runs in

$$O(\sum_{v \in V} |\tau^+_v| M) \subseteq O(M^2)$$

time if $n^2 \leq M$ on incurable temporal graph.

Hence, Algorithm 3 has the best running time that we know for solving All-Pairs Optimal Walk within all time intervals $[t, t'] \in [T]$.

Table 4.1: Comparison of the running times of Algorithm 3 and Algorithm 4 for All-Pairs Optimal Walk (APOP) to the running times of the Single Source Optimal Walk (SSOP) algorithms on incurable temporal graphs.

Optimality Criterion	APOP	SSOP
foremost	n^2M	M
reverse-foremost	n^2M	$M \log M$
fastest	n^2M	
shortest	M^3	$M \log M$
minimum hop-count	M^3	
cheapest	M^3	

5 Polynomial Fixed-Parameter Algorithms

We now discuss the potentials and boundaries of fixed-parameter algorithms for the polynomial time solvable problem of finding optimal walks in temporal graphs. The main motivation herein is to find efficient algorithms for problems that have an unattractive polynomial running time [GMN17]. The goal is to identify appropriate parameters k and to derive algorithms that run in $f(k) \cdot \text{poly}(n)$ time and $f(k) \cdot \text{poly}(M)$ time for Single Source Optimal Walk and All-Pairs Optimal Walk, respectively. A function $\text{poly}(\cdot)$ is a polynomial function only depending on the input variable and a function $f(k)$ is an arbitrary function only depending on parameter k. Three design goals for such algorithms were formulated by Giannopoulou, Mertzios, and Niedermeier [GMN17]:

(1) The running time should have polynomial dependency on the parameter k.

(2) The running time should be close to linear if the parameter k is a constant.

(3) The parameter, or a good approximation of it, should be computable in polynomial time (preferable in linear time).

We start by showing a fundamental limitation for this approach in finding optimal walks in temporal graphs: There can not exist an algorithm that runs in $f(n)$ time for any function f. We will show that the statement even holds in incurable temporal graphs with a directed path as underlying graph. Next, we will look into the potentials of fixed-parameter algorithms. This is a first try to adapt the idea of polynomial fixed-parameter algorithm to finding optimal walks in temporal graphs. Therefore, we limit our focus to Foremost on instantaneous, incurable temporal graphs. We describe a fixed-parameter algorithm with respect to the vertex cover number of the underlying graph for finding all-pairs optimal walks in a temporal graph. Afterwards, we introduce a data structure based on the tree-decomposition of the underlying graph that allows us to conduct fast foremost walk queries in an incurable temporal graph.

5.1 Parameter Restriction

One important difference between temporal graphs and static graphs is that the number of time-arcs cannot be upper bounded by a function in the number of vertices. We will even show that no deterministic algorithm exists for finding an optimal walk in a

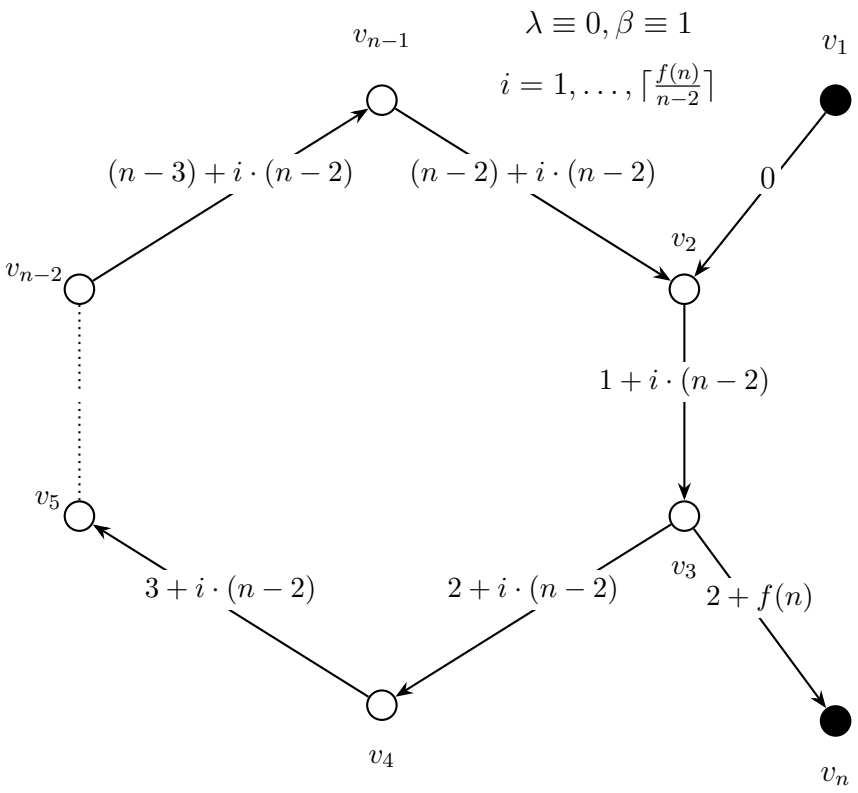


Figure 5.1: A temporal graph with n vertices and a temporal walk from v_1 to v_n of length greater than $f(n)$.

temporal graph with a running time only depending on the number of vertices in the graph. We will conclude that any parameter k that is upper bounded by a function in the number of vertices cannot be used for a fixed-parameter algorithm for Single Source Optimal Walk that runs in $f(k) \cdot \text{poly}(n)) \subseteq O(g(n))$ time, where g is a function only depending on the number n of vertices.

In our approach, we show that the length of an optimal temporal walk cannot be upper bounded by any function solely depending on the number of vertices.

Theorem 5.1.1. *For finding an optimal walk in a temporal graph with n vertices, there is no deterministic algorithm for which the number of time-arc queries can be upper bounded by a function $f\colon \mathbb{N} \to \mathbb{N}$ in n.*

Proof. Assume towards a contradiction that there is a deterministic algorithm that queries at most $f(n)$ time-arcs for finding an optimal temporal walk in a graph with n vertices. We show that such an algorithm can be fooled by an adversary. An adversary answers the time-arc requests of the algorithm. It can always add or delete time-arcs

that have not yet been queried by the algorithm. With these changes it can affect the existence of a temporal walk or the value of an optimal temporal walk.

To proof Theorem 5.1.1, the strategy of an adversary is to build a graph with n vertices in which the only temporal walk between two vertices consists of more than $f(n)$ time-arcs. Thus, the algorithm cannot find this walk with only $f(n)$ time-arc queries. A construction of an optimal temporal walk of length $> f(n)$ can be done by exploiting the maximum dwell time β. A possible construction is displayed in Figure 5.1:

The labels on the arcs display the time steps at which the arcs appear. Hence, for each $j = 2, \dots, n-2$, from v_j to v_{j+1} there are time-arcs at time steps $(j+1)+i\cdot(n-2)$ for $i = 1, \dots, \lceil \frac{f(n)}{n-2} \rceil$, and from v_{n-1} to v_2 there are time-arcs at time steps $(n-2)+i\cdot(n-2)$ for $i = 1, \dots, \lceil \frac{f(n)}{n-2} \rceil$. The transmission time of every time-arc is 1. The maximum dwell time in all vertices is restricted to 1 time step. The only temporal walk from v_1 to v_n uses $2 + f(n) > f(n)$ time-arcs. Starting in v_1 at time step 0, we have to take at every time step a new time-arc to avoid violating the dwell time constraints. Hence, the only way to reach v_n is to walk around the circle until reaching v_3 at time step $1 + f(n)$ from where we can go to v_n in time step $2 + f(n)$.

To ensure the existence of the temporal walk the algorithm has to request all time-arcs on the temporal walk. In our construction the only temporal walk from vertex v_1 to v_n consists of $2 + f(n)$ time-arcs. After $f(n)$ time-arc requests the algorithm has not seen all time-arcs on the walk from v_1 to v_n and, therefore, cannot decide whether a temporal walk from v_1 to v_n exists.

For Minimum Waiting Time, the same construction can be used in an incurable temporal graph with $\lambda \equiv 1$ to show that the only optimal walk consists of more than $f(n)$ time-arcs. In Figure 5.1, if we assume that $\lambda \equiv 1$, then there is only one temporal walk from v_1 to v_n without waiting time. This walk consists of $f(n) + 2$ time-arcs. Hence, the minimum waiting time walk walk cannot be found by an algorithm with only $f(n)$ time-arc requests. □

The theorem holds for temporal walks in temporal graphs in general and for Minimum Waiting Time in incurable temporal graphs. Next, we analyse whether there can exist algorithms for the remaining optimal walk definitions that run with only $f(n)$ time-arc queries in incurable temporal graphs. In such temporal graphs, most of our optimal walks can be reduced to optimal paths (see Lemma 3.2.5) and thus, are restricted to at most $n-1$ time-arcs. Hence, the idea behind the proof of Theorem 5.1.1 is not applicable. We will examine for which time-arc queries and optimal walks it is possible to find a deterministic algorithm that runs with $f(n)$ time-arc queries. To this end, we distinguish three different kinds of time-arc queries:

- A *next-departure query* $Q_{\mathrm{nd}}(v, w, t)$ with $(v, w, t) \in V \times V \times T$ returns a set of time-arcs from v to w that have the smallest time stamp $t' \geq t$ among all time-arcs from v to w, that is,

$$\{(v, w, t', \lambda) \in E \mid t' = \min\{t' \mid (v, w, t', \lambda) \in E \wedge t \leq t' \wedge \lambda \in \mathbb{N}\}\}.$$

  In Fig. 5.2, $Q_{\mathrm{nd}}(v, w, 1)$ returns the set $\{(1, 3), (1, 4)\}$.

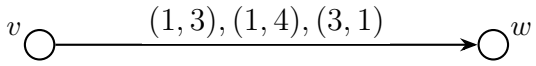


Figure 5.2: Several time-arcs from a vertex v to a vertex w. Recall that a tuple (t, λ) on an arc represents a time-arc with time stamp t and transmission time λ.

- An *earliest-arrival query* $Q_{\text{ea}}(v, w, t)$ with $(v, w, t) \in V \times V \times T$ returns a time-arc from v to w that has the earliest arrival time among all time-arcs from v to w with time stamp $t' \geq t$, that is, a time-arc

$$(v, w, t', \lambda) \in E \text{ s.t. } t' \geq t \wedge \nexists (v, w, t'', \lambda') \in E \colon t'' \geq t \wedge t'' + \lambda' < t' + \lambda$$

  In Fig. 5.2, $Q_{\text{ea}}(v, w, 1)$ returns for example the time-arc $(v, w, 3, 1)$.

- A *latest-departure query* $Q_{\text{ld}}(v, w, t)$ with $(v, w, t) \in V \times V \times T$ returns a time-arc from v to w that has the largest time stamp among all time-arcs from v to w that arrive latest in time step t, that is, a time-arc

$$(v, w, t', \lambda) \in E \text{ s.t. } t' + \lambda \leq t \wedge \nexists (v, w, t'', \lambda') \in E \colon t'' + \lambda' \leq t \wedge t'' > t'$$

  In Fig. 5.2, $Q_{\text{ld}}(v, w, 4)$ also returns the time-arc $(v, w, 3, 1)$.

For the next-departure query, for every $(v, w) \in V \times V$ one stores a sorted list of all time-arcs $(v, w, t, \lambda) \in E$ sorted by time stamp in ascending order.

For the earliest-arrival query, for every $(v, w) \in V \times V$ one stores a sorted list of all time steps $t \in T$ such that there is a time-arc $(v, w, t, \lambda) \in E$. For each time step t in the list, a adequate time-arc from v to w is stored with the earliest arrival time among all time-arcs from v to w starting in time step t or later.

For the latest-departure query, for every $(v, w) \in V \times V$ one stores a sorted list of all time steps $t + \lambda \in T$ such that there is a time-arc $(v, w, t, \lambda) \in E$. For each time step t in the list, an adequate time-arc from v to w is stored with the latest departure time among all time-arcs from v to w with an arrival time latest in time step t.

With these data structures at hand, all these queries can be answered efficiently by binary search on the sorted time step list. Nevertheless, we will assume that such queries can by processed and answered in constant time. But before, we briefly explain why we only consider these three query types.

However, for some optimal walk definitions it is difficult to find an appropriate time-arc query that makes it easier to find an optimal walk. Let us consider shortest walks as an example: The first query that comes into mind is a shortest-transmission-time query. Given two vertices $v, w \in V$ and a time step $t \in T$, it returns a time-arc from v to w that has the shortest transmission time among all time-arcs from v to w that do not start before t. The time-arc returned could appear at the end of the lifetime of a temporal graph and, therefore, cannot be used in any shortest walk. Thus, such queries can be tricked by an adversary quite easily. Due to this problem, a reasonable query type in finding an optimal walk is the next-departure query.

We start with Foremost, Reverse-Foremost, Minimum Hop-Count, and Fastest, where the more fine-grained earliest-arrival query and latest-departure query can be employed. We will show that there is an algorithm for finding an optimal walk in an incurable temporal graph that runs with n^3 time-arc queries for Foremost and Reverse-Foremost, and with $n!$ time-arc queries for Minimum Hop-Count by exploiting the earliest-arrival query and latest-departure query. However, on temporal graphs with bounded dwell time and a directed path as underlying graph these queries do not lead to an algorithm with time-arc queries upper bounded by a function in the number of vertices. Also for Fastest, these queries do not provide an algorithm which time-arc queries are bounded by a function in the number of vertices anymore. Next, we will show that for the next-departure query there is no algorithm for finding an optimal walk in an incurable temporal graph that runs with $f(n)$ time-arc queries for any of the optimal walk definitions even on temporal graphs with a directed path as underlying graph.

5.1.1 Earliest-Arrival Query & Latest-Departure Query

In incurable temporal graphs, recall that any foremost, reverse-foremost, minimum hop-count, and fastest walk can be transformed to a temporal path without loosing optimality as shown in Lemma 3.2.5. Thus, these walks consist of at most $n-1$ time-arcs because each vertex is visited at most once.

Foremost, Reverse-Foremost & Minimum Hop-Count. We already made an important observation concerning foremost walks: If there is a temporal walk from a vertex v to a vertex w, then there is a foremost path such that all prefix-paths are foremost paths (Lemma 3.2.6). Something similar can be observed for reverse-foremost paths: If there is a temporal walk from a vertex v to a vertex w, then there is a reverse-foremost path such that all postfix-paths are reverse-foremost paths as well (Lemma 3.2.9). Last but not least, if we look at all minimum hop-count paths from a vertex v to a vertex w that visit the vertices v_1 to v_k successively, then there is a path so that any vertex v_i is visited earliest as possible for all $i \in [k]$. Based on these observations we show the following:

Proposition 5.1.2. *Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph. There exists a deterministic algorithm that can find a foremost and reverse-foremost path with $O(n^3)$ earliest-arrival queries and latest-departure queries respectively. A minimum hop-count path can be found with $O(n!)$ earliest-arrival queries.*

Proof. We will describe the algorithm for finding a foremost path and describe the adaptions that have to be made for a reverse-foremost path. Afterwards, we briefly explain the brute-force algorithm for a minimum hop-count path.

Let $\mathcal{G} = (V, E, [T])$ be an incurable temporal graph. Let $v, w \in V$ be two vertices. For Foremost, the algorithm builds up a set of all vertices to which there is a path from v, together with the earliest arrival time to each of these vertices. The algorithm takes one vertex after another, always looking for the next vertex that can be reached earliest in

time. In this way, the algorithm finds the earliest arrival time for all vertices including the vertex w. More precisely, the algorithm works with two sets:

- $A \subseteq V \times T$: A set of vertex-time step tuples (w', t) such that a foremost walk from v to w' arrives at time step t
- $B \subseteq V$: A set of vertices for which no foremost path has been found yet

It starts with the set $A = \{(v, 0)\}$ and the set $B = V \setminus \{v\}$. In each round, the algorithm runs the following steps:

1. For each vertex-time step tuple $(v', t) \in A$ and each $w' \in B$ an earliest-arrival query $Q_{\text{ea}}(v', w', t)$ is conducted. There are at most n^2 such queries in each round.

   For a vertex $w' \in B$ that has the earliest arrival time t' among all conducted earliest-arrival queries, a foremost path from v to w' has arrival time t'.

2. The vertex-time step tuple (w', t') is added to the set A and w' is removed from B, formally, $A \leftarrow A \cup \{(w', t)\}$ and $B \leftarrow B \setminus \{w'\}$.

   If no (w', t') exists that can be added to A, then all remaining vertices in B cannot be reached from v. Thus, the algorithm stops.

The algorithm needs at most n of these rounds until the vertex w is removed from B. Thus, finding a foremost path from v to w needs at most n^3 earliest-arrival queries.

Concerning the correctness of the algorithm, assume that in an arbitrary round the set A consists only of vertex-time step tuples $(w', t) \in A$ such that a foremost walk from v to w' arrives at time step t, and no vertex in B can be reached earlier in time than the vertices in A. This holds at least in the first round where $A = \{(v, 0)\}$. Let (w', t') with $w' \in B$ be the next vertex-time step tuple that is added to the set A by the algorithm. Thus, all other vertices $w'' \in B$ cannot be reached earlier than t' and w' cannot be reached earlier than t' via one of the other vertices in B. Hence, t' is the earliest possible time step in which a path from v to w' can arrive.

For Reverse-Foremost, the main idea is quite similar: The algorithm starts with the set $A = \{(w, \infty)\}$ and $B = V \setminus \{w\}$ and conducts latest-departure queries to find the next vertex that can reach w with the latest departure time. Thereby, we compute reverse-foremost paths from any vertex—including v—to w.

For Minimum Hop-Count, the algorithm consecutively examines for each $k = [n - 2]$ each possible sequence $(v_1, \ldots, v_k)$ of pairwise-distinct vertices in $V \setminus \{v, w\}$ whether there is a temporal path from v to w visiting the vertices $(v_1, \ldots, v_k)$. This can be done by conducting earliest arrival queries. There are at most $(n-2)!$ such sequences. A temporal path with the smallest k is a minimum hop-count path from v to w the algorithm was looking for. The algorithm runs in $O(n!)$ time with earliest-arrival queries. □

We have already seen in Theorem 5.1.1 that there is no algorithm for finding an optimal walk in temporal graphs in general that runs in $O(f(n))$ time due to the maximum dwell time. The construction (see Figure 5.1) was a temporal graph with a cycle in the

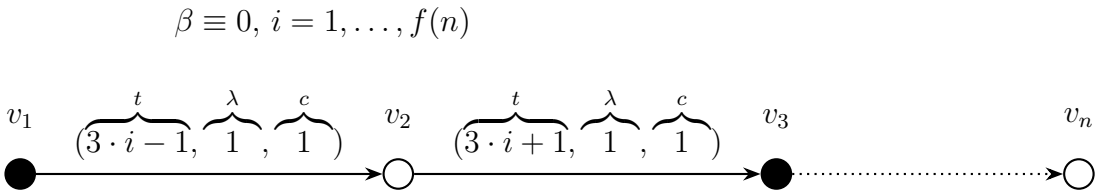


Figure 5.3: A temporal graph with n vertices where the underlying directed graph is a path.

underlying graph. But even if we restrict our consideration to temporal graphs with a directed path as underlying graph, then the earliest-arrival and latest-departure query do not lead to an algorithm for Foremost, Reverse-Foremost, and Minimum Hop-Count that is upper bounded by a function in the number of vertices—even though all temporal walks are restricted to $n - 1$ time-arcs.

Proposition 5.1.3. *If the underlying graph of a temporal graph is a path, then there is no deterministic algorithm for finding a foremost-walk, a reverse-foremost walk, or a minimum hop-count walk where the number of earliest-arrival queries or latest-departure queries can be upper bounded by a function $f\colon \mathbb{N} \to \mathbb{N}$ in the number of vertices.*

Proof. Assume towards a contradiction that there is an algorithm for Foremost that needs $O(f(n))$ earliest-arrival queries to find an optimal walk in any temporal graph with a directed path as underlying graph.

We construct such a graph and a strategy for the adversary such that any deterministic algorithm will need more than $f(n)$ earliest-arrival queries to find a foremost path from a vertex v_1 to a vertex v_3. Let us look at the graph shown in Figure 5.3. In this proof, we assume that the the maximum dwell time is 0 in all vertices—$\beta \equiv 0$. From v_1 to v_2 there are time-arcs at time steps $3 \cdot i - 1$, and from v_2 to v_3 there are time-arcs at time steps $3 \cdot i + 1$ for $i = 1, \ldots, f(n)$. All time-arcs have transmission time 1 and cost 1. The two vertices v_1 and v_3 that are colored black are the start- and endpoint of the optimal walk the algorithm is looking for. Notice that any temporal walk from v_1 to v_3 can only consist of two time-arcs—(v_1, v_2, t, λ) and (v_2, v_3, t', λ')—such that $t + \lambda = t'$ due to $\beta \equiv 0$. The strategy of an adversary depends on the approach of the algorithm:

Case 1. The algorithm conducts earliest-arrival queries from vertex v_1 to v_2 consecutively increasing in the time stamp. That is, it starts by executing the earliest-arrival query $Q_{\text{ea}}(v_1, v_2, 0)$. The query returns time-arc $(v_1, v_2, 2, 1)$. This time-arc is not part of a temporal walk from v_1 to v_3 in our graph. The algorithm continues to request the next possible time-arc: $Q_{\text{ea}}(v_1, v_2, 3)$. The query returns time-arc $(v_1, v_2, 5, 1)$. This time-arc is again not part of a temporal walk from v_1 to v_3 in our graph. The algorithm conducts earliest-arrival queries $Q_{\text{ea}}(v, w, i)$ for $i = 0, 3, \ldots, 3 \cdot (f(n) - 1)$.

After $f(n)$ queries, the algorithm has seen all time-arcs from vertex v_1 to v_2 up to time step $3 \cdot f(n) - 1$. Until now, all of them are not part of a temporal walk from v_1 to v_3. Now, the adversary can add one more time-arc $(v_1, v_2, 3 \cdot f(n), 1)$. With this time-arc, there exists a temporal walk from v_1 to v_3, that is, $((v_1, v_2, 3 \cdot f(n), 1), (v_2, v_3, 3 \cdot f(n)+1, 1))$. This temporal walk cannot be found by the algorithm with only $f(n)$ queries.

Case 2. The algorithm missed executing an earliest-arrival query $Q_{\text{ea}}(v_1, v_2, j)$ for a $j \in \{0, 3, \ldots, 3 \cdot (f(n) - 1)\}$. Now, the adversary can add the time-arc $(v_1, v_2, j, 1)$ to our temporal graph. With this time-arc there exists a temporal walk from v_1 to v_3, that is, $((v_1, v_2, 3 \cdot j, 1), (v_2, v_3, 3 \cdot j + 1, 1))$.

Thus, each deterministic algorithm with only $O(f(n))$ earliest-arrival queries is not optimal. Due to the adversary, the algorithm did not find any walk from v_1 to v_3 after $f(n)$ earliest-arrival queries. Thus, it cannot find a minimum hop-count walk either. The proof idea and the temporal graph can be used to show the corollary for reverse-foremost paths with latest-departure queries. □

According to Proposition 5.1.3 there is no hope to improve the running time upper bounds for finding an optimal walk for Foremost, Reverse-Foremost, and Minimum Hop-Count for any non-trivial temporal graph class. Next, we have a look at the last definition in which a more fine-grained query type seems reasonable: Fastest.

Fastest. A fastest path P is a foremost path in the induced temporal graph starting at the time step in which the temporal path P begins (Lemma 3.2.3). Thus, earliest-arrival queries are a legitimate choice. We are, however, able to show that finding a fastest temporal walk cannot be done by an algorithm that conducts $f(n)$ earliest-arrival queries. It is not even possible in incurable temporal graphs with a directed path as an underlying graph. The statement also holds for Shortest and Cheapest.

Proposition 5.1.4. *For finding a fastest, shortest, or cheapest walk in a temporal graph with n vertices, there is no deterministic algorithm for which the number of earliest-arrival queries can be upper bounded by any function $f\colon \mathbb{N} \to \mathbb{N}$ in n.*

Proof. The proof scheme is the same as in Proposition 5.1.3. We start with proving the proposition for Fastest.

Assume towards a contradiction that there is an algorithm that needs $O(f(n))$ earliest-arrival queries for finding a fastest walk in any incurable temporal graph with n vertices. We construct a temporal graph with n vertices and a strategy for the adversary such that any deterministic algorithm will need more than $f(n)$ earliest-arrival queries to find a fastest path from v_1 to v_3. Let us look at the temporal graph that is displayed in Figure 5.3. We assume that there is unbounded dwell time—$\beta \equiv \infty$. Recall that any temporal walk from vertex v_1 to vertex v_3 starts in vertex v_1, passes through v_2 and ends in v_3. Thus, the first time-arc of any temporal walk starts with a time-arc from v_1 to v_2. The strategy of the adversary is a case distinction depending on the approach of the algorithm:

Case 1. The algorithm queries the time-arcs from vertex v_1 to v_2 step by step increasing in the time stamps. Thus, the algorithm executes earliest-arrival queries $Q_{\text{ea}}(v_1, v_2, i)$ for $i = 0, 3, \ldots, 3 \cdot (f(n) - 1)$. The queries return the time-arcs $(v_1, v_2, j, 1, 1)$ for $j = 2, 5, \ldots, 3 \cdot f(n) - 1$. After $f(n)$ queries, the algorithm has seen all time-arcs from vertex v_1 to v_2 up to time step $3 \cdot f(n) - 1$. Until now all possible temporal paths, that could have been found, have a duration of at least 3 time steps because when starting at time $3 \cdot i - 1$ it is only possible to arrive at vertex v_3 in time $3 \cdot i + 1$. But then, the adversary can add one more time-arc $(v_1, v_2, 3 \cdot f(n), 0, 0)$. With this time-arc the fastest path from v_1 to v_3 starts at time $3 \cdot f(n)$ and arrives at time $3 \cdot f(n) + 1$. It has a duration of 2 time steps and is therefore the fastest path from v_1 to v_3.

Case 2. The algorithm missed executing an earliest-arrival query from v_1 to v_2 at a time step $j \in \{0, 3, \ldots, 3 \cdot (f(n) - 1)\}$. The adversary can now add the time-arc $(v_1, v_2, j, 0, 0)$. With this time-arc the fastest path from v_1 to v_3 starts at time $3 \cdot j$ and arrives at time $3 \cdot j + 1$. It has a duration of 2 time steps and is therefore the fastest path from v_1 to v_3.

Thus, each deterministic algorithm with only $f(n)$ earliest-arrival queries is not optimal. The newly created walk in both cases is not only the fastest walk, but also the shortest and cheapest walk among all temporal walks from v_1 to v_3. □

We have seen so far that for Foremost, Reverse-Foremost, and Minimum Hop-Count in incurable temporal graphs, there exists an algorithm that runs with n^3 and $n!$ earliest-arrival queries and latest-departure queries. We will now show that there is no deterministic algorithm that conducts at most $f(n)$ next-departure queries for finding any optimal walk in an incurable temporal graph.

5.1.2 Next-Departure Query

We will show that for finding an optimal walk we cannot have an upper bound on the number of time-arcs queries depending only on the number of vertices in a temporal graph. More formally:

Proposition 5.1.5. *For finding an optimal walk in an incurable temporal graph with n vertices, there is no deterministic algorithm for which the number of next-departure queries can be upper bounded by a function $f \colon \mathbb{N} \to \mathbb{N}$ in n.*

Proof. We begin with the optimal walk definitions Fastest, Shortest, and Cheapest. Afterwards, we give a proof of our statement for Foremost and Reverse-Foremost. Finally, we discuss Minimum Hop-Count.

Fastest, Shortest & Cheapest. We have already seen the proof for these three optimal walk variants. In Proposition 5.1.4, if we replace earliest-arrival queries with next-departure queries, then the returned time-arcs are identical in the incurable temporal graph of Figure 5.3.

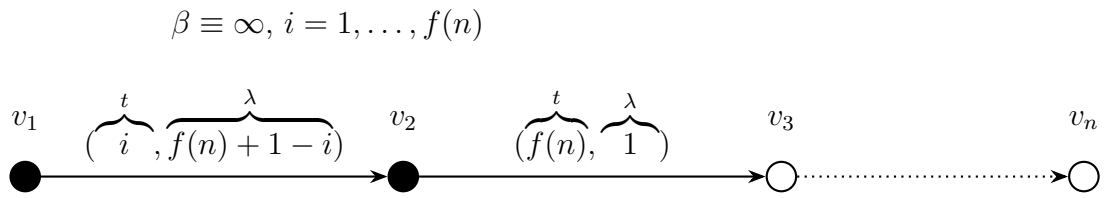


Figure 5.4: An incurable temporal graph with n vertices where the underlying directed graph is a path.

Foremost & Reverse-Foremost. Assume towards a contradiction that there is an algorithm that needs $O(f(n))$ next-departure queries for finding a foremost walk in any incurable temporal graph with n vertices. We construct a temporal graph with n vertices and a strategy for the adversary such that any deterministic algorithm will need more than $f(n)$ next-departure queries to find a foremost walk from v_1 to v_2. Let us look at the temporal graph displayed in Figure 5.4. From v_1 to v_2 there are time-arcs at time step i with transmission time $f(n) + 1 - i$ for $i = 1, \ldots, f(n)$. Any temporal walk from vertex v_1 to vertex v_2 consists of only one time-arc from v_1 to v_2. The strategy of an adversary depends on the approach of the algorithm:

Case 1. The algorithm queries the time-arcs from vertex v_1 to v_2 step by step increasing in the time stamps. Thus, the algorithm executes next-departure queries $Q_{\mathrm{nd}}(v_1, v_2, i)$ for $i = 0, \ldots, f(n) - 1$. The queries return the time-arcs $(v_1, v_2, i, f(n) + 1 - i)$ for $j = 0, \ldots, f(n) - 1$. After $f(n)$ queries, the algorithm has seen all time-arcs from vertex v_1 to v_2 up to time step $f(n) - 1$. Until now all possible temporal walks arrive at time step $f(n) + 1$. But then, the adversary can add one more time-arc $(v_1, v_2, f(n), 0)$. With this time-arc the walk arrives at time step $f(n)$.

Case 2. The algorithm missed executing a next-departure-query from v_1 to v_2 at a time step $j \in \{0, \ldots, f(n)\}$. The adversary can now add the time-arc $(v_1, v_2, j, 0)$. With this time-arc the foremost walk from v_1 to v_2 arrives at time $j \leq f(n)$.

Thus, each deterministic algorithm with only $f(n)$ next-departure queries is not optimal. The same proof idea works for reverse-foremost walks.

Minimum Hop-Count. If we look at the proof for foremost walks, no algorithm found a temporal walk from v_1 to v_2 that arrives earlier than time step $f(n)+1$ after $f(n)$ next-departure queries. Thus, the temporal walk from v_1 to v_3 cannot be found after $f(n)$ next-departure queries because the only time-arc from v_2 to v_3 is at time step $f(n)$, see Figure 5.4. □

In Proposition 5.1.5 we have shown that there is no algorithm for finding an optimal walk in an incurable temporal graph that runs with $f(n)$ next-departure queries—even

when the underlying graph is a directed path. We have seen in Proposition 5.1.2 that if we allow earliest-arrival and latest departure queries, then there are algorithms that run with n^3 and $n!$ such queries in incurable temporal graphs. But already in general temporal graphs with a directed path as underlying graph this upper bound by a function in the number of vertices does not hold anymore as shown in Proposition 5.1.3.

We can conclude that there is no algorithm for finding optimal walks in temporal graphs that has a running time that is upper bounded by a function in the number of vertices, even in temporal graphs with a directed path as underlying graph. Thus, there is no algorithm for Single Source Optimal Walk that runs in $f(k) \cdot \text{poly}(n)$ time for any parameter k that is upper bounded by a function in the number of vertices.

5.2 Fixed-Parameter Algorithms

We have elaborated in Section 5.1 the limitations of fixed-parameter algorithms for finding optimal walks in temporal graphs. Now, we will look into the potentials of fixed-parameter algorithms. This is a first try to adapt the idea of polynomial fixed-parameter algorithm to finding optimal walks in temporal graphs. Therefore, we limit our focus to the optimal walk variant Foremost on instantaneous, incurable temporal graphs.

We will introduce a fixed-parameter algorithms with respect to the vertex cover number k—a parameter on the top of the graph parameter hierarchy [SW13]. The algorithm runs in $O((k|\tau|)^2 n + kn^2)$ time. Last but not least, we introduce a data structure based on the tree-decomposition with treewidth k of the underlying undirected graph that allows us to conduct fast foremost walk queries in $O(k^2|\tau| \log n)$ time in an incurable temporal graph.

5.2.1 Temporal Graphs with Bounded Vertex Cover Number

We will start with looking at the parameter vertex cover number of the underlying graph and how we can exploit this parameter for finding optimal walks. The vertex cover number is usually a quite large parameter. Nevertheless, in some networks such as flight networks that we briefly discussed in Chapter 1, this parameter can be small.

Given a static undirected graph $G = (V, E)$, a *vertex cover* is a vertex subset $V_{\text{VC}} \subseteq V$ such that for all edges $e \in E$ it holds that e has at least one endpoint in V_{VC}, that is, $e \cap V_{\text{VC}} \neq \emptyset$. The *vertex cover number* is the size of a minimum vertex cover. In a temporal graph $\mathcal{G}$, we consider the vertex cover number of the underlying undirected graph $G_u[\mathcal{G}]$. It is well known that finding a minimum vertex cover is NP-hard. But there is a simple 2-approximation that runs in linear time: For an edge, both endpoints are taken into the vertex cover and then are removed from the graph with all adjacent edges. Repeatedly applying this procedure leads to a vertex cover of size at most twice as large the minimum vertex cover. Thus, the parameter is suitable for a polynomial fixed-parameter algorithms.

This parameter combined with the number of vertices is not sufficient for a polynomial fixed-parameter algorithm. But combining the vertex cover number k with the

parameter $|\tau|$, that is, the number of time steps in which at least one time-arc exist, allows us to upper bound the number of time-arcs and, thus, leads to an algorithm for All-Pairs Foremost Walk. First, note that the number of time-arcs can be upper bounded by $O(k|\tau|n)$, that is, $M \in O(k|\tau|n)$ because in a static graph with vertex cover number k there can be at most $k \cdot n$ edges. Each static graph G_t for $t \in \tau$ is a subgraph of the underlying graph $G_d[\mathcal{G}]$. Thus, each G_t can have at most $k \cdot n$ arcs and, thus, there can be at most $O(k|\tau|n)$ time-arcs in the temporal graph.

In Theorem 5.2.1, we will show that solving All-Pairs Foremost Walk in instantaneous, incurable temporal graphs runs in $O\big((k \cdot |\tau|)^2 n + kn^2\big)$ time where k is the vertex cover number of the underlying undirected graph of the temporal graph. The algorithm fulfills the three design goals that we introduced in the beginning of this chapter. However, in most datasets we have seen so far the parameter $|\tau|$ is larger than the number of vertices in the graph. Hence, solving n times the Single Source Foremost Walk which is solvable in $O(M) \subseteq O(k|\tau|n)$ time by Algorithm 1 can lead to a better running time. Consequently, this result is more of a classification result than of practical use.

Algorithm. Given a temporal graph $\mathcal{G} = (V, E, [T])$ with vertex cover number k, and a vertex cover V_{VC} with $|V_{\text{VC}}| \in O(k)$, the idea of Algorithm 5 is as follows: First, it computes for each $v \in V_{\text{VC}}$ a foremost walk to all vertices in $\mathcal{G}$, that is, solving Single Source Foremost Walk with source vertex v on $\mathcal{G}$ (Lines 2 and 3). Then, for each $v \in V_{\text{VC}}$ and for each $t \in \tau_v^+$ a foremost walk is computed to all vertices within the time interval $[t, T]$, that is, solving Single Source Foremost Walk with source vertex v on $\mathcal{G}[[t, T]]$ (Lines 4 to 6). Afterwards, Algorithm 5 computes for each vertex $v \in V \setminus V_{\text{VC}}$ a foremost walk to all vertices. Therefor, for each $v \in V \setminus V_{\text{VC}}$, a new acyclic, directed static graph $G_v = (V_v, E_v)$ is created (Line 8):

The vertex set V_v consists of the vertices V and the additionally the vertex cover vertices $\{w_{\text{VC}} \mid w \in V_{\text{VC}}\}$. For each $w \in V_{\text{VC}}$, it determines a v-w time-arc with the earliest arrival time in w, that is, $(v, w, t, 0) \in E$ such that $t = \min\{t' \mid (v, w, t', 0) \in E\}$, if it exists. If such an time-arc exists, then an arc (v, w_{VC}) is added to E_v (Lines 15 and 16). The cost of this arc is set to zero (Line 17). Next, for each vertex $x \in V$ an arc (w_{VC}, x) is added to E_v. If $w_{\text{VC}} = x$, then the cost of this arc is set to t. Else, the cost of this arc is set to the earliest arrival time from w_{VC} to x starting earliest at time t (Lines 18 to 20). These earliest arrival times have already been computed.

Now, the algorithm solves Single-Source Shortest Path for source v on this sparse acyclic, directed static graph by using Dijkstra's algorithm (Line 9). The cost of a shortest path from v to a vertex $w \in V_v \cap V$ in G_v is the arrival time of a foremost walk from v to w in $\mathcal{G}$.

We will prove the correctness of Algorithm 5 and analyze its running time to conclude:

Theorem 5.2.1. *Algorithm 5 solves* All-Pairs Foremost Walk *on instantaneous, incurable temporal graphs with underlying vertex cover number k in $O\big((k \cdot |\tau|)^2 n + kn^2\big)$ time.*

Algorithm 5: All-Pairs Foremost Walk for instantaneous, incurable temporal graphs

```
/* Description of Variables:
   opt(v,t) stores a list of all w ∈ V with the earliest arrival time
            from v to w within [t,T];
   opt(v,t,w) stores the earliest arrival time from v to w within [t,T].
                                                                      */
1  function AllPairForemostPath(G = (V,E,[T]), V_VC ⊆ V):
       /* Computing Single Source Foremost Walk for all vertices in the
          vertex cover.                                               */
2      for v ∈ V_VC do
3          opt(v,1) ← SingleSourceForemostWalk(G, v)
       /* Computing Single Source Foremost Walk for all vertices in the
          vertex cover and for all time steps in which the vertex has
          an in-going time-arc.                                       */
4      for v ∈ V_VC do
5          for t ∈ τ_v^+ do
6              opt(v,t) ← SingleSourceForemostWalk(G[[t,T]], v)
       /* Computing Single Source Foremost Walk for all vertices not in
          the vertex cover                                            */
7      for v ∈ V \ V_VC do
8          G_v, c ← generateGraph(G, V_VC, opt, v)
9          opt(v,1) ← Dijkstra(G_v, c, v)
10     return opt
11 function generateGraph(G, V_VC, opt, v):
12     V_v ← V ∪ {v_VC | v ∈ V_VC}
13     E_v ← ∅
       /* Adding an arcs from s to every vertex cover vertex and a arc
          from those to every vertex in the graph with the earliest
          arrival time represented by costs on the arcs.              */
14     for w ∈ V_VC do
15         if ∃(v,w,t,0): (v,w,t,0) ∈ E ∧ t = min{t' | (v,w,t',0) ∈ E} then
16             E_v ← E_v ∪ {(v, w_VC)}
17             c((v, w_VC)) = 0
18             for x ∈ V do
19                 E_v ← E_v ∪ {(w_VC, x)}
20                 c((w_VC, x)) = { opt(w, min{t' ∈ τ_w^+ | t' ≥ t}, x)   , if w ≠ x
                                  { t                                     , else
21     return (V_v, E_v), c
```

Proof. We start with showing the correctness of Algorithm 5, and continue to prove the running time of it.

Correctness. Given an instantaneous, incurable temporal graph $\mathcal{G} = (V, E, [T])$ with vertex cover number k, and a vertex cover $V_{\text{VC}} \subseteq V$ of size $O(k)$, Algorithm 5 computes Single Source Foremost Walk for each vertex $v \in V_{\text{VC}}$ correctly.

Given a vertex not in the vertex cover—$v \in V \setminus V_{\text{VC}}$—we know the second vertex of any temporal walk starting at v has to be a vertex cover vertex. (If not, then the arc from v to the second vertex would not be covered by the vertex cover—a contradiction to the definition of a vertex cover.)

By Lemma 3.2.6, we know that if there exists a walk from v to a vertex $w \in V$, then there exists a foremost path $P = (e_1, \dots, e_l)$ with $e_i = (v_i, w_i, t_i, 0) \in E$ from v to w such that every prefix-path is a foremost path. Hence, (e_1) is a foremost path from v_1 to w_1. Additionally, we know by Lemma 3.2.7 that the postfix-path $(e_2, \dots, e_l)$ of P is a foremost path within $\mathcal{G}[[t_1, T]]$. In the generated graph, for $(w_1)_{\text{VC}} \in V_{\text{VC}}$ there is an arc $(v, (w_1)_{\text{VC}})$ due to e_1 with cost zero and an arc from $(w_1)_{\text{VC}}$ to w representing a foremost walk from $(w_1)_{\text{VC}}$ to w starting earliest at time step t_1 with cost t_l. Hence, the foremost walk P between v and w exists by the arc $(v, (w_1)_{\text{VC}})$ representing e_1 and the arc $((w_1)_{\text{VC}}, w)$ with cost t_l representing the foremost path $(e_2, \dots, e_l)$ within time interval $[t_1, T]$. If $l = 1$, then the arc $((w_1)_{\text{VC}}, w)$ has cost t_1.

The path $((v, (w_1)_{\text{VC}}), ((w_1)_{\text{VC}}, w)$ is the shortest path with t_l in the generated graph, and thus, will be found by running Dijkstra's algorithm.

Running Time. First, note that V_{VC} is at most twice as large as the minimum vertex cover when using the 2-approximation algorithm and thus, still of size $O(k)$.

For all $v \in V_{\text{VC}}$, and all $t \in \tau_v^+$ where $|\tau_v^+| \leq \tau_{\max}^+ \leq |\tau|$, the algorithm solves Single Source Foremost Walk in $O(k|\tau|n)$ time with Algorithm 1. Hence, there is a total running time of $O\big((k|\tau|)^2 n\big)$.

Afterwards, for each vertex $v \in V \setminus V_{\text{VC}}$ an acyclic, directed graph G_v is generated with at most one arc from v to an vertex in the vertex cover, that is, $O(k)$ arcs, and at most one arc from the vertex cover vertices to any vertex in the graph, that is, $O(k \cdot n)$ arcs. In the directed graph there are at most $O(n)$ vertices. Hence, the whole generated acyclic graph is of size $O(kn)$. The whole construction of G_v runs in linear time. The costs of the arcs have to be extracted from the earliest arrival times from the vertex cover vertices to all vertices in the graph. This runs in $O(k|\tau|n)$ during the whole run of the algorithm.

Solving Single-Source Shortest Path with the Dijkstra's algorithm on G_v for each $v \in V \setminus V_{\text{VC}}$ runs in $O(kn)$ because G_v is acyclic. We can sum up the running time with

$$\begin{aligned} & O((k\tau_{\max}^+)^2 n) + O(k\tau_{\max}^+ n) + O(kn^2)\big) \\ \subseteq\ & O((k|\tau|)^2 n + kn^2) \end{aligned}$$

Consequently, Algorithm 5 runs in $O\big((k|\tau|)^2 n + kn^2\big)$. □

Algorithm 5 is only faster than n times Single Source Foremost Walk on the incurable temporal graph, if $k|\tau| < n$ due to $M \in O(k|\tau|n)$. It seems hard to transfer this algorithm to other optimal walk definitions without significantly increasing the running time.

5.2.2 Temporal Graphs with Bounded Treewidth

In this subsection, we will study the application of the well-established concept of treewidth and tree-decomposition for finding foremost walks in instantaneous, incurable temporal graphs. This concept has been used as a theoretical approach to speed up shortest path computation in road networks. Using a tree-decomposition of a static graph, a data structure can be implemented that allows fast and exact distance queries—also referred to as distance oracle—if the treewidth is small [Abr+16]. We take the same approach used to derive this result to exploit the tree-decomposition of the underlying graph for fast and exact foremost walk oracle in instantaneous, incurable temporal graphs. The treewidth is bounded by the vertex cover number [SW13] and so it seems appropriate to also study this parameter.

Before we get into the algorithmic idea, we briefly recap the definition of tree-decomposition for static graphs that we apply to the underlying undirected graph of a temporal graph.

Definition 5.2.2. A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{X}, \mathcal{T} = (\mathcal{X}, \mathcal{E}, R))$ consisting of a rooted tree $\mathcal{T}$ on $\mathcal{X}$ with root $R \in \mathcal{X}$ and a family $\mathcal{X}$ of sets $X \subseteq V$(called bags), such that

(1) for all $v \in V$ the set
$$\mathcal{X}^{-1}(v) := \{X \in \mathcal{X} \mid v \in X\}$$
is non-empty and induces a subtree of $\mathcal{T}$ and

(2) for every edge $e \in E$ there is a bag $X \in \mathcal{X}$ with $e \subseteq X$.

The *width* $w((\mathcal{X}, \mathcal{T}))$ of the tree-decomposition $(\mathcal{X}, \mathcal{T})$ is $w((\mathcal{X}, \mathcal{T})) := \max\{|X| - 1 \mid X \in \mathcal{X}\}$. The treewidth $tw(G)$ of a graph G is defined as the minimal width of any tree-decompositions of G. The *depth* of a rooted tree-decomposition is the depth of the rooted tree $\mathcal{T}$.

Among the wide range of results concerning tree-decompositions we make use of a result of Bodlaender [Bod89] to derive a faster way of computing foremost walks.

Theorem 5.2.3 ([Bod89])**.** *If G has treewidth k, then G has a binary tree-decomposition with width $3k + 2$ and depth $O(\log n)$.*

Bodlaender et al. [Bod+16] introduce an algorithm that takes a graph G and a $k \in \mathbb{N}$ and returns either that the tree width of G is larger than k or a tree-decomposition of G with width at most $5k + 4$. The algorithm runs in $O(2^k n)$ which is linear in the input size if k is a constant. Fomin et al. [Fom+17] give a polynomial time approximation

with running time $O(k^7 n \log n)$ that, given a graph G and a $k \in \mathbb{N}$, either provides a tree decomposition of G of width at most $O(k^2)$, or correctly concludes that $tw(G) \geq k$.

We introduce some notation that help us describe the techniques: if there are two bags $X, Y \in \mathcal{X}$ where X lies on the shortest path from the root R to Y, then we will call X an ancestor of Y and Y a descendent of X. Furthermore, let $\mathcal{T}_X$ with $X \in \mathcal{X}$ be the induced subtree of $\mathcal{T}$ rooted in X containing all descendents of X. Let $V(\mathcal{T}_X)$ be the set of all vertices $v \in V$ such that there exists a bag Y in $\mathcal{T}_X$ with $v \in Y$. For a vertex $v \in V$ let X_v be the closest bag to the root with $X_v \in \mathcal{X}^{-1}(v)$. For a time-arc $(v, w, t, 0) \in E$ let $X_{\{v,w\}}$ be the bag closest to the root with $\{v, w\} \in X_{\{v,w\}}$.

Foremost Walk Oracle. In this paragraph, we introduce a technique that exploits a tree-decomposition of the underlying, undirected graph of an instantaneous, incurable temporal graph to create a data structure that allows fast foremost walk queries between two vertices and within a chosen time interval. This method is a trade-off between storage and query time:

(1) The data structure needs less memory than the information about all foremost walks between any pairs of vertices within any arbitrary time interval.

(2) The query takes less time than the computation of the foremost walk from scratch, if the treewidth is small.

Our idea is based on the work of Abraham et al. [Abr+16] for shortest path computation in static graphs with bounded treewidth. We adapt this idea to instantaneous, incurable temporal graphs. The main challenge lies in maintaining a foremost walk between two vertices for any possible starting time within the lifetime of a temporal graph.

We start by introducing the data structure that is based on a tree-decomposition of the underlying undirected graph of a temporal graph that allows fast foremost walk queries. We will briefly discuss the running time of the preprocessing that leads to the desired data structure and the size of the data structure. Next, we will explain how this data structure can be exploited to answer foremost walk queries more efficiently. As we have already seen for the parameter vertex cover, the parameter treewidth k is not enough for a fixed-parameter algorithm. We also include the parameter $|\tau|$ to our analysis. Note that we can upper bound the number of time-arcs M by $O(k|\tau|n)$, that is, $M \in O(k|\tau|n)$. We summarize our results in the following theorem before we go into the details:

Theorem 5.2.4. *There exists a data-structure of size $O(k^2|\tau|n)$ that can be computed in $O(k^3|\tau|^2 n)$ preprocessing time such that finding a foremost walk between two vertices $v, w \in V$ takes $O(k^2|\tau| \log n \log(k|\tau| \log n))$ time on instantaneous, incurable temporal graphs with underlying treewidth k.*

Preprocessing & Data Structure. As a first step towards proving Theorem 5.2.3, we consider a given rooted tree-decomposition $(\mathcal{X}, \mathcal{T} = (\mathcal{X}, \mathcal{E}, R \in \mathcal{X}))$ of the underlying, undirected graph $G_u[\mathcal{G}]$ where $\mathcal{T}$ is a binary tree with depth $O(\log n)$ and width $O(k)$.

First, we compute a data structure based on the tree-decomposition that allows us to run fast foremost walk queries for any two vertices within any time interval of our original instantaneous, incurable temporal graph.

We compute for every bag $X \in \mathcal{X}$ and for every two vertices $v, w \in X$ and $t \in \tau_v^+$ the earliest arrival time from v to w starting earliest in t in $\mathcal{G}[V(\mathcal{T}_X)]$, that is, the temporal graph induced by all vertices in bags in $\mathcal{T}_X$. We refer to this earliest arrival time as $d_X(v, w, t)$. Let us first consider the running time of computing this information:

Lemma 5.2.5. *The preprocessing runs in $O(k^3|\tau|^2 n)$ time.*

Proof. For the preprocessing, assume that for a bag $X \in \mathcal{X}$ we already processed the two children X_l and X_r of X in a bottom-up approach. We then create a temporal graph over the vertex set X and add all time-arcs of the original graph $\mathcal{G}$ between any two vertices $v, w \in X$. We also add for all two vertices $v, w \in X_l \cap X$ and $v, w \in X_r \cap X$ all foremost walks for any possible starting time $t \in \tau_v^+$ computed in X_l and X_r. That is, for v and w, and all $t \in \tau_v^+$ we add a time-arc from v to w with time stamp t and transmission time $\min\{d_{X_l}(v, w, t) - t, d_{X_r}(v, w, t) - t\}$.

The resulting graph $\mathcal{G}_X$ has $O(k)$ vertices and at most $O(k^2\tau^+_{\max})$ time-arcs. This is enough information to simulate the whole temporal graph $\mathcal{G}[V(\mathcal{T}_X)]$ with only these k vertices because $X_l \cap X$ is a separator between vertices in $V(\mathcal{T}_{X_l})$ and vertices in X. The same holds for vertices in $X_r \cap X$.

The resulting temporal graph $\mathcal{G}_X$ has transmission times greater than zero. These time-arcs with transmission time greater zero can be transformed in linear time into two time-arcs without transmission time by Transformation 2. We further have to sort the time-arc by time stamp in $O\big(k^2\tau^+_{\max} \log(k\tau^+_{\max})\big)$ time.

Afterwards, we solve for every vertex $v \in X$ and for every $t \in \tau_v^+$ Single Source Foremost Walk on the instantaneous, incurable temporal graph $\mathcal{G}_X[[t, T]]$ with source vertex v which leads to $k\tau^+_{\max}$ calls of Algorithm 1 on $\mathcal{G}_X$. This has to be done for every bag in $\mathcal{T}$ in a bottom-up approach. There are at most $O(n)$ bags in $\mathcal{T}$. Consequently, the whole preprocessing leads to a running time in $O(k^3{\tau^+_{\max}}^2 n) \subseteq O(k^3|\tau|^2 n)$. □

After preprocessing, we have to store the structure of $\mathcal{T}$, the information we computed for every bag, and some additional information to allow fast foremost walk queries.

Lemma 5.2.6. *The size of the data structure is in $O(k^2|\tau|n)$.*

Proof. The structure of $\mathcal{T}$ only requires $O(n)$ memory for the $O(n)$ bags and $O(n)$ edges. For every bag $X \in \mathcal{X}$, we have to store for every $v, w \in X$ and $t \in \tau_v^+$ the earliest arrival time from v to w within $[t, T]$ in $\mathcal{G}[V(\mathcal{T}_X)]$, that is, $d_X(v, w, t)$. These are $O(k^2\tau^+_{\max} n)$ earliest arrival times that have to be stored. Furthermore, we have to store X_v for every $v \in V$ and $X_{\{v,w\}}$ for every $(v, w, t, 0) \in E$ which requires $O(kn)$ space. □

We can see that the preprocessing takes $O(k^3|\tau|^2 n)$ time whereas computing a foremost walk runs in $O(M) \subseteq O(k|\tau|n)$ time. But the size $O(k^2|\tau|n)$ of the data structure is smaller than storing the information of size $O(|\tau|n^2)$ about the earliest arrival time between any pairs of vertices and all possible starting times, if the treewidth is small.

Foremost Walk Query. We will now look at a foremost walk query. We want to compute the foremost walk between two vertices $v, w \in V$ for a certain starting time $t \in [T]$ by using only the data structure we discussed above.

Therefore, let $\mathcal{X}(v, w)$ be the set of all bags $X \in \mathcal{X}$ such that it holds that X is an ancestor of X_v or X_w. Let $V(\mathcal{X}(v, w)) := \{u \mid u \in X \wedge X \in \mathcal{X}(v, w)\}$.

Let $\mathcal{G}(v, w, t) = (V', E', [t, T])$ be the temporal graph consisting of the vertex set $V' = V(\mathcal{X}(v, w))$. Construct E' as follows: For all two vertices $v', w' \in V'$ and $t' \in \tau_v^+$ with $t \leq t'$ such that there exists a bag $X \in \mathcal{X}(v, w)$ with $v', w' \in X$, we add time-arcs from v' to w' starting at t' and arriving at time step $t_{\min} = \min\{d_X(v', w', t') \mid X \in \mathcal{X}(v, w) \wedge v', w' \in X\}$, that is, time-arc $(v', w', t', t_{\min} - t')$.

Observe that the number of bags in $\mathcal{X}(s, t)$ is at most $O(\log n)$ (due to the binary tree-decomposition with depth $O(\log n)$). Hence, the temporal graph $\mathcal{G}(v, w, t)$ consists of $O(k|\tau| \log n)$ vertices and $O(k^2|\tau| \log n)$ time-arcs. These time-arcs with transmission time greater than zero can also be transformed into time-arcs without transmission time by Transformation 2 in linear time. We have to sort the time-arcs by time stamp in $O(k^2|\tau| \log n \log(k|\tau| \log n))$ time. Now, we can solve Single Source Foremost Walk for the instantaneous, incurable temporal graphs $\mathcal{G}(v, w, t)$ with source v. This takes $O(k^2|\tau| \log n)$ time with Algorithm 1. We can conclude:

Lemma 5.2.7. *The foremost walk query takes $O(k^2|\tau| \log n \log(k|\tau| \log n))$ time.*

It further holds that the earliest arrival time from v to w in $\mathcal{G}(v, w, t)$ is an earliest arrival time in $\mathcal{G}[[t, T]]$. To prove this statement and, thus, the correctness of our method, we will make use of a lemma that has already been proven by Abraham et al. [Abr+16]:

Lemma 5.2.8 ([Abr+16], Claim 1)**.** *For any time-arc $(v, w, t) \in E$, if $\text{depth}(X_v) \geq \text{depth}(X_w)$, then it holds that $X_{\{v,w\}} = X_v$.*

With Lemma 5.2.8 in hand, we can start the correctness proof of our foremost walk oracle. We refer to the earliest arrival time from v to w in $\mathcal{G}(v, w, t)$ as $d(v, w, \mathcal{G}(v, w, t))$ and to the earliest arrival time from v to w in $\mathcal{G}[[t, T]]$ as $d(v, w, \mathcal{G}[[t, T]])$.

Lemma 5.2.9. *Let $v, w \in V$ be two vertices and $t \in T$ be a time step. The earliest arrival time from v to w in $\mathcal{G}(v, w, t)$ is equal to the earliest arrival time in $\mathcal{G}[[t, T]]$, that is, $d(v, w, \mathcal{G}(v, w, t)) = d(v, w, \mathcal{G}[[t, T]])$.*

Proof. The proof is by induction on the number of time-arcs in the foremost walk from v to w starting in time t. Let us assume that $P = (e_1, \ldots, e_k)$ with $e_i = (v_i, w_i, t_i)$ is a foremost path from v to w in $\mathcal{G}[[t, T]]$ with k time-arcs such that all prefix-paths of P are foremost paths by Lemma 3.2.6.

For the induction basis, let us consider $k = 1$. Then, there exist the time-arc e_1 from v to w within $[t, T]$ such that t_1 is the earliest arrival time. Without loss of generality, let $\text{depth}(X_v) \geq \text{depth}(X_w)$, then we know by Theorem 5.2.8 that $X_{\{v,w\}} = X_v$. So, $(v, w, t_1, 0) \in \mathcal{G}[V(\mathcal{T}_{X_v})]$ and, consequently, $(v, w, t_1, 0) \in \mathcal{G}(v, w, t)$.

For the induction hypothesis, let us assume that for all $v, w \in V$, and all $t \in T$, if $(e'_1, \ldots, e'_l)$ with $l < k$ be a foremost walk from v to w starting at time t with an earliest arrival time $d(v, w, \mathcal{G}[[t, T]])$, then $d(v, w, \mathcal{G}(v, w, t)) = d(v, w, \mathcal{G}[[t, T]])$.

Now, we consider two vertices $v, w \in V$ and a time step $t \in T$ such that a foremost walk $P = (e_1, \dots, e_k)$ has exactly k time-arcs and all prefix-paths of P are foremost paths by Lemma 3.2.6. We further assume that $\text{depth}(X_v) \geq \text{depth}(X_w)$ without loss of generality. Note that for all $i \in [k]$ it holds that $t_i \in [t, T]$. We distinguish two cases:

Case 1: We consider the case that for all $i \in [k]$ it holds that the time-arc $e_i \in \mathcal{G}[V(\mathcal{T}_{X_v})]$. In particular, this means that $e_k \in \mathcal{G}[V(\mathcal{T}_{X_v})]$ and, consequently, $w \in V(\mathcal{T}_{X_v})$. It follows that $w \in X_v$ due to the assumption that $\text{depth}(X_v) \geq \text{depth}(X_w)$ and property (1) of the tree-decomposition. Hence,

$$d(v, w, \mathcal{G}(v, w, t)) = d_{X_v}(v, w, t) = d(v, w, \mathcal{G}[[t, T]]).$$

Case 2: In the second case, let e_j with $j \in [k]$ be the first time-arc in the foremost walk P such that $e_j \notin \mathcal{G}[V(\mathcal{T}_{X_v})]$.

Note that for the vertex v_2 it holds that $v_2 \in V(\mathcal{T}_{X_v})$. We know that all bags that contain v are in $\mathcal{T}_{X_v}$. Thus, all time-arcs from or to v have to be in $\mathcal{G}[V(\mathcal{T}_{X_v})]$. It follows that $e_1 \in \mathcal{G}[V(\mathcal{T}_{X_v})]$ and, consequently, $v_2 \in V(\mathcal{T}_{X_v})$.

It follows that $j > 1$. We further know that $e_{j-1} \in \mathcal{G}[V(\mathcal{T}_{X_v})]$. Thus, $w_{j-1} = v_j \in V(\mathcal{T}_{X_v})$. Even more, we know that $v_j \in X_v$ because there is a bag in $\mathcal{T}_{X_v}$ that contains v and there exists another bag Y not in $\mathcal{T}_{X_v}$ with $v_j, w_j \in Y$ due to the fact that $e_j \notin \mathcal{G}[V(\mathcal{T}_{X_v})]$. Hence, $v_j \in X_v$ by property (1) of the tree-decomposition.

Hence, we know that the prefix-path $(e_1, \dots, e_{j-1})$ is represented in $\mathcal{G}(v, w, t)$ by an edge from v to v_j starting at t_1 and arriving at t_{j-1}. Recall that $v, v_j \in X_v$ and $e_1, \dots, e_{j-1} \in \mathcal{G}[V(\mathcal{T}_{X_s})])$. Thus,

$$d(v, v_j, \mathcal{G}(v, w, t)) = d_{X_v}(v, v_j, t) = d(v, v_j, \mathcal{G}[[t, T]]).$$

Due to the induction hypothesis, we further know that for the subwalk $(e_j, \dots, e_k)$ it holds that

$$d(v_j, w, \mathcal{G}(v_j, w, t_j)) = d(v_j, w, \mathcal{G}[[t_j, T]]).$$

It remains to show that $\mathcal{G}(v_j, w, [t_j, T]) \subseteq \mathcal{G}(v, w, t)$. But this follows from the fact that $v_j \in X_v$ and, consequently, X_{v_j} is an ancestor of X_v because X_{v_j} is the closest bag to the root among all bags containing v_j (by construction of $\mathcal{G}(v, w, t)$). Hence,

$$d(v_j, w, \mathcal{G}[[t_j, T]]) = d(v_j, w, \mathcal{G}(v_j, w, t_j)) \geq d(v_j, w, \mathcal{G}(v, w, t_j)) = d(v_j, w, \mathcal{G}[[t_j, T]]).$$

We can conclude that

$$d(v, w, \mathcal{G}(v, w, t)) = d(v, w, \mathcal{G}[[t, T]]).$$

This proves that $\mathcal{G}(v, w, t)$ contains enough information to compute the earliest arrival time from v to w in $\mathcal{G}[[t, T]]$. □

If we compare the running time of a foremost walk query in $O(k^2|\tau| \log n \log(k|\tau| \log n))$ to solving Single Source Foremost Walk in $O(M) \subseteq O(k|\tau|n)$, we can see that there is a noticeable improvement for instantaneous incurable temporal graphs with small underlying treewidth k.

6 Conclusion

This thesis contributes to a better understanding of temporal walks in temporal graphs. In particular, temporal walks with arbitrary transmission times on the time-arcs and with minimum and maximum dwell time restrictions in the vertices have not received much attention in the literature before. We discussed the impact of these parameters on temporal walks to gain a better understanding of the structure of temporal walks.

Our main algorithmic focus was finding optimal walks in temporal graphs. We considered not only the most common optimal walk variants—*Foremost*, *Reverse-Foremost*, *Fastest*, and *Minimum Hop-count*—but also variants such as *Shortest*, *Cheapest*, and *Minimum Waiting Time*. We analyzed properties and structures of these optimal walks. Due to the strong impact of maximum dwell time on optimal walks, we distinguish between optimal walks in temporal graphs in general and optimal walks in temporal graphs with unbounded maximum dwell time in particular.

In our algorithmic investigations, we introduced algorithms solving Single Source Optimal Walk on temporal graphs for the variants *Foremost* and *Fastest* with running time $O(M)$ and $O(M \log M)$ respectively. To our surprise, bounded maximum dwell time had no influence on the efficiency of the algorithms. One main take-away of our algorithmic investigations is the awareness of the potentially exponential size of the lifetime T of a temporal graph with respect to the input size. As a consequence, one difficulty that arises is the choice of suitable data structures to store intermediate results of algorithms finding optimal walks.

We analyzed the potentials and limitations of fixed-parameter algorithms for finding optimal walks. We concluded that fixed-parameter algorithms have to include parameters $|\tau|$ or M in the running time—even for graph classes of the underlying graph that contain a directed path of length three.

Future Work. We conclude the thesis with a list of further research directions.

Implementation. Examining the performance of Algorithm 1 and Algorithm 2 for finding single-source optimal walks on real-world datasets should be a next step in the algorithmic investigation of optimal walks—especially in combination with Transformation 1 and Transformation 2. The results can be used as an indicator whether specific algorithms for non-instantaneous temporal graphs have to be developed in the context of temporal graphs with maximum dwell time.

It is also of interest to examine whether our Floyd-Warshall adaptation—Algorithm 3—can compete with the algorithms of Wu et al. [Wu+16] on non-instantaneous, incurable temporal graphs for computing foremost, reverse-foremost and fastest walks between

any two pairs of vertices and within any time interval. This also holds for our second Floyd-Warshall adaptation—Algorithm 4—for the remaining optimal walk variants.

Enumerating Walks. In public transport networks, as an example, it is not only necessary to find one fastest transportation route but to find all fastest transportation routes through the system to offer a selection of possibilities. Finding second fastest routes that are only slightly longer than the optimum can also increase the information gained. Hence, constructing efficient algorithms for enumerating all optimal walks and finding second optimal walks in temporal graphs can be a future research goal.

Finding pareto-optimal walks can also be of major interest in some fields of application such as flight networks as discussed briefly in Chapter 1. Passengers are not solely interested in optimizing one criterion such as cost, but finding a good balance between flight duration and cost. Hence, finding and enumerating pareto-optimal walks can be of major benefit for a plethora of applications of optimal walks.

Fixed-Parameter Algorithms for Finding Optimal Walks. Finding structural graph parameters that are small in temporal graphs is a first step to extract potential parameters that are well-motivated considering for fixed-parameter algorithms. One example for such parameters is the diameter which tends to be small in social networks. Temporal graph parameters such as the maximum dwell time β also warrant consideration. As seen in Chapter 1, this parameter is constant in proximity networks when analyzing transmission routes of infectious diseases.

Literature

[Abr+16] I. Abraham, S. Chechik, D. Delling, A. V. Goldberg, and R. F. Werneck. "On Dynamic Approximate Shortest Paths for Planar Graphs with Worst-case Costs". In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*. SIAM, 2016, pp. 740–753 (cit. on pp. 95, 96, 98).

[AF16] K. Axiotis and D. Fotakis. "On the size and the approximability of minimum temporally connected subgraphs". In: *arXiv preprint arXiv:1602.06411* (2016) (cit. on p. 15).

[AVL62] G. Andelson-Velskii and E. Landis. "An algorithm for the organisation of information". In: *Soviet Mathematics-Doklady* 3 (1962), pp. 1259–1262 (cit. on p. 32).

[Baj+11] P. Bajardi, A. Barrat, F. Natale, L. Savini, and V. Colizza. "Dynamical Patterns of Cattle Trade Movements". In: *PLOS ONE* 6.5 (2011), pp. 1–19 (cit. on p. 9).

[Ber96] K. A. Berman. "Vulnerability of scheduled networks and a generalization of Menger's theorem". In: *Networks* 28.3 (1996), pp. 125–134 (cit. on p. 15).

[BF14] A. Barrat and J. Fournet. *DATASET: High school dynamic contact networks.* `http://www.sociopatterns.org/datasets/high-school-dynamic-contact-networks/`. 2014 (cit. on p. 21).

[Bod+16] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. "A c^kn 5-Approximation Algorithm for Treewidth". In: *SIAM Journal on Computing* 45.2 (2016), pp. 317–378 (cit. on p. 95).

[Bod89] H. L. Bodlaender. "NC-algorithms for graphs with small treewidth". In: *Proceedings of the 14th International Workshop on Graph-Theoretic Conecpts in Computer Science (WG '88)*. Springer, 1989, pp. 1–10 (cit. on p. 95).

[Cas+12] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. "Time-varying graphs and dynamic networks". In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (2012), pp. 387–408 (cit. on p. 10).

[Dea04] B. C. Dean. "Algorithms for minimum-cost paths in time-dependent networks with waiting policies". In: *Networks* 44 (2004), pp. 41–46 (cit. on pp. 15, 57).

[Flo62] R. W. Floyd. "Algorithm 97: Shortest Path". In: *Communications of the ACM* 5.6 (1962), pp. 345–348 (cit. on p. 66).

[Fom+17] F. V. Fomin, D. Lokshtanov, M. Pilipczuk, S. Saurabh, and M. Wrochna. "Fully polynomial-time parameterized computations for graphs and matrices of low treewidth". In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*. SIAM. 2017, pp. 1419–1432 (cit. on p. 95).

[GBC14] V. Gemmetto, A. Barrat, and C. Cattuto. "Mitigation of infectious disease at school: targeted class closure vs school closure". In: *BMC Infectious Diseases* 14.1 (2014), p. 1 (cit. on p. 21).

[GMN17] A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. "Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs". In: *Theoretical Computer Science* 689 (2017), pp. 67–95 (cit. on p. 81).

[Goe11] R. Goerke. *Email Network of KIT Informatics*. http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata. 2011 (cit. on p. 21).

[Hol15] P. Holme. "Modern temporal network theory: a colloquium". In: *The European Physical Journal B* 88.9 (2015), p. 234 (cit. on pp. 9, 11, 15).

[Hol16] P. Holme. "Temporal network structures controlling disease spreading". In: *Physical Review E* 94.2 (2016), p. 022305 (cit. on p. 9).

[HS12] P. Holme and J. Saramäki. "Temporal networks". In: *Physics Reports* 519.3 (2012), pp. 97–125 (cit. on pp. 9, 11, 12, 15).

[Ise+11] L. Isella, J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton, and W. Van den Broeck. "What's in a crowd? Analysis of face-to-face behavioral networks". In: *Journal of Theoretical Biology* 271.1 (2011), pp. 166–180 (cit. on p. 21).

[KA12] H. Kim and R. Anderson. "Temporal node centrality in complex networks". In: *Physical Review E* 85.2 (2012), p. 026107 (cit. on pp. 15, 78).

[KKK00] D. Kempe, J. Kleinberg, and A. Kumar. "Connectivity and Inference Problems for Temporal Networks". In: *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*. ACM, 2000, pp. 504–513 (cit. on p. 15).

[KLS02] E. Köhler, K. Langkau, and M. Skutella. "Time-Expanded Graphs for Flow-Dependent Transit Times". In: *Proceedings of the 10th Annual European Symposium on Algorithms (ESA '15)*. Springer, 2002, pp. 599–611 (cit. on p. 15).

[Mer+13] G. B. Mertzios, O. Michail, I. Chatzigiannakis, and P. G. Spirakis. "Temporal Network Optimization Subject to Connectivity Constraints". In: *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP '13)*. Springer, 2013, pp. 657–668 (cit. on pp. 11, 15, 23).

[MH13] N. Masuda and P. Holme. "Predicting and controlling infectious disease epidemics using temporal networks". In: *F1000prime Reports* 5 (2013) (cit. on p. 9).

[ML+16] M. Moslonka-Lefebvre, C. A. Gilligan, H. Monod, C. Belloc, P. Ezanno, J. A. N. Filipe, and E. Vergu. "Market analyses of livestock trade networks to inform the prevention of joint economic and epidemiological risks". In: *Journal of The Royal Society Interface* 13.116 (2016) (cit. on p. 9).

[Nic+12] V. Nicosia, J. Tang, M. Musolesi, G. Russo, C. Mascolo, and V. Latora. "Components in time-varying graphs". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22.2 (2012), p. 023101 (cit. on p. 15).

[Nic+13] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. "Graph Metrics for Temporal Networks". In: *Temporal Networks*. Understanding Complex Systems. Springer, 2013, pp. 15–40 (cit. on pp. 9, 11, 12, 23).

[PS11] R. K. Pan and J. Saramäki. "Path lengths, correlations, and centrality in temporal networks". In: *Physical Review E* 84.1 (2011), p. 016105 (cit. on pp. 15, 78).

[Sal+10] M. Salathé, M. Kazandjieva, J. W. Lee, P. Levis, M. W. Feldman, and J. H. Jones. "A high-resolution human contact network for infectious disease transmission". In: *Proceedings of the National Academy of Sciences* 107.51 (2010), pp. 22020–22025 (cit. on p. 9).

[San+11] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard. "Time-varying graphs and social network analysis: Temporal indicators and metrics". In: *arXiv preprint arXiv:1102.0629* (2011) (cit. on pp. 10, 12, 15, 23, 78).

[Sku09] M. Skutella. "An introduction to network flows over time". In: *Research Trends in Combinatorial Optimization*. Springer, 2009, pp. 451–482 (cit. on p. 15).

[Ste+11] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. Van den Broeck, C. Régis, B. Lina, et al. "High-resolution measurements of face-to-face contact patterns in a primary school". In: *PLoS ONE* 6.8 (2011), e23176 (cit. on p. 21).

[SW13] M. Sorge and M. Weller. "The Graph Parameter Hierarchy". Manuscript. 2013. URL: https://manyu.pro/assets/parameter-hierarchy.pdf (cit. on pp. 91, 95).

[Tan+13] J. Tang, I. Leontiadis, S. Scellato, V. Nicosia, C. Mascolo, M. Musolesi, and V. Latora. "Applications of temporal graph metrics to real-world networks". In: *Temporal Networks*. Springer, 2013, pp. 135–159 (cit. on pp. 15, 78).

[Van+13] P. Vanhems, A. Barrat, C. Cattuto, J.-F. Pinton, N. Khanafer, C. Régis, B. Kim, B. Comte, and N. Voirin. "Estimating potential infection transmission routes in hospital wards using wearable proximity sensors". In: *PLoS ONE* 8.9 (2013), e73970 (cit. on p. 21).

[VLM16] J. Viard, M. Latapy, and C. Magnien. "Computing maximal cliques in link streams". In: *Theoretical Computer Science* 609.1 (2016), pp. 245 –252 (cit. on p. 9).

[War62] S. Warshall. "A Theorem on Boolean Matrices". In: *Journal of the ACM* 9.1 (1962), pp. 11–12 (cit. on p. 66).

[Wu+16] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu. "Efficient Algorithms for Temporal Path Computation". In: *IEEE Transactions on Knowledge and Data Engineering* 28.11 (2016), pp. 2927–2942 (cit. on pp. 12, 15, 23, 35, 37, 57, 61, 101).

[XFJ03] B. B. Xuan, A. Ferreira, and A. Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks". In: *International Journal of Foundations of Computer Science* 14.02 (2003), pp. 267–285 (cit. on pp. 10–12, 15, 23, 37).

[Zsc+17] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. "The Computational Complexity of Finding Separators in Temporal Graphs". In: *arXiv preprint arXiv:1711.00963* (2017) (cit. on p. 15).