Technical University of Berlin
Faculty IV: Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Algorithmic Aspects of Betweenness Centrality in Temporal Networks

Sebastian Buß

Thesis submitted in fulfillment of the requirements for the degree
"Bachelor of Science" (B. Sc.) in the field of Computer Science

September 2019

Supervisor and first reviewer:	Prof. Dr. Rolf Niedermeier
Second reviewer:	Prof. Dr. Markus Brill
Co-Supervisor:	Hendrik Molter

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 eigenhändige Ausfertigung versichert an Eides statt

Berlin, den _____ _____

Datum Unterschrift

Zusammenfassung

Die Betweenness Centrality eines Knotens in einem Netzwerk ist ein Maß für die relative Häufigkeit, mit der kürzeste Pfade im Netzwerk diesen Knoten passieren. Sie gilt in der Netzwerkanalyse als Maß dafür, wie wichtig dieser Knoten im Vergleich zu den anderen im jeweiligen Netzwerk ist. Ein temporales Netzwerk ist, im Gegensatz zum statischen Netzwerk, eines, bei dem im Laufe einer endlichen, diskreten Zeitfolge Kanten hinzukommen und verschwinden können. Brandes' Algorithmus berechnet für statische Netzwerke mit n Knoten und m Kanten die Betweenness Centrality aller Knoten in $\mathcal{O}(n \cdot m)$ Zeit. Die vorliegende Arbeit betrachtet verschiedene Varianten temporaler Betweenness und untersucht die jeweilige algorithmische Komplexität. Für einige der Varianten stellt sich heraus, dass sich Brandes' Ansatz auf den temporalen Fall übertragen lässt, und es werden effiziente Algorithmen angegeben. Bei anderen Varianten wird bewiesen, dass die entsprechenden Probleme #P-schwer sind, so dass unter gängigen Annahmen der Komplexitätstheorie kein Algorithmus mit polynomieller Laufzeit existieren kann.

Abstract

The *betweenness centrality* of a vertex measures how often this vertex is visited on shortest paths between other vertices of the graph. In the analysis of technical, social or biological networks, betweenness centrality of a node is used as an indicator for the relative importance of a node in the network. Brandes' algorithm computes the betweenness centrality of all vertices in a static graph with n vertices and m edges in $\mathcal{O}(n \cdot m)$ time. In recent years, a growing number of real-world networks are modeled as *temporal graphs* instead of conventional (static) graphs because the latter are incapable of reflecting the dynamics of a network that changes over time. In a temporal graph, there is a finite discrete set of time steps and every edge may be present at some time steps and absent at others. While shortest paths are a central concept in conventional graph theory, temporal paths can be considered 'optimal' with respect to many different aspects, including length, arrival time, and overall travel time (shortest, foremost, and fastest paths). The *temporal betweenness centrality* of a vertex can be defined based on any concept of optimal paths. While this allows closer modeling of dynamic processes, it poses new challenges on the algorithmic side. We show that counting foremost and fastest paths is intractable and conclude that the computation of the corresponding temporal betweenness is intractable as well. For shortest paths and two selected special cases of foremost paths, we adapt Brandes' algorithm and devise polynomial-time algorithms for the computation of the corresponding temporal betweenness.

Contents

Chapter 1

Introduction

Graphs and various graph metrics such as the betweenness centrality are studied and applied in a wide variety of fields such as social and technological network analysis [Ley07; Tan+09], wireless routing [DH07], machine learning [ŞB09] and neuroscience [Heu+10]. The *betweenness centrality* of a vertex in a graph measures how many shortest paths in the graph go through this vertex. High betweenness centrality scores are usually associated with vertices that can be seen as more important for the network in some sense. In static graphs, the betweenness centrality is a well-studied concept. In particular, Brandes' algorithm [Bra01] computes the betweenness centrality of all vertices in a given static graph with n vertices and m edges in $\mathcal{O}(n \cdot m)$ time and $\mathcal{O}(n + m)$ space.

On temporal graphs, the notion of betweenness centrality can be defined in a similar fashion, but there are more options how to choose 'optimal' paths. Depending on the application, a path may be optimal if it minimizes the number of edges (shortest), the arrival time (foremost), or the overall travel time (fastest), and for any of these path classes we can define and study a variant of temporal betweenness centrality. In addition to that, combinations of optimality criteria such as *shortest foremost* paths can be considered. Furthermore, we distinguish between temporal paths with strictly or non-strictly ascending time labels.

We investigate the algorithmic aspects of temporal betweenness variants based on strict and non-strict shortest, foremost and fastest paths. In addition to that, we consider two subclasses of foremost paths, namely the shortest foremost and the prefix-foremost paths, and define temporal betweenness classes based on those paths as well.

	strict	non-strict
Shortest	$\mathcal{O}(n \cdot M)$	$\mathcal{O}(n \cdot M)$
Foremost	#P-hard	#P-hard
Fastest	#P-hard	#P-hard
Prefix-foremost	$\mathcal{O}(n \cdot M \cdot \log M)$	#P-hard
Shortest foremost	$\mathcal{O}(n \cdot M \cdot \log M)$	$\mathcal{O}(n \cdot M \cdot \log M)$

Table 1.1: An overview of the complexity of the temporal betweenness variants we consider. Here, n refers to the number of vertices and M to the number of time edges.

Our main research question is: how hard is the computation of the different variants of temporal betweenness centrality?

The different betweenness variants show surprising differences in their computational complexity: while some of them can be computed in polynomial running time, others are computationally hard. More specifically, we show that counting foremost and fastest temporal paths is #P-hard. In Table 1.1, we give an overview of our findings. For the cases where the computation is hard, we give formal hardness proofs. In the cases where we state polynomial-time computability, we explicitly state algorithms that compute the temporal betweenness scores of all vertices in a given temporal graph.

1.1 Related work

The betweenness centrality in static graphs is a well-studied concept. Defined as early as 1977 by Freeman [Fre77], the betweenness centrality has since been used in a broad variety of studies related to network analysis. The computation of the betweenness centrality, however, was limited to comparatively small networks for a long time due to the cubic running times of the best known algorithms. In 2001, Brandes presents a significantly faster algorithm that takes advantage of the typically sparse graphs that occur in network analysis. Brandes' algorithm [Bra01] computes the betweenness centrality of all vertices for a given static graph with n vertices and m edges in $\mathcal{O}(n \cdot m)$ time and $\mathcal{O}(n+m)$ space. In the area of exact algorithms, this algorithm remains state-of-the-art to this day and our own algorithms are devised in a similar fashion. In 2007, however, Bader et al. [Bad+07] present a faster approximation algorithm that approximates the betweenness centrality of a given vertex in a given graph.

The theory of temporal graphs is considerably younger than graph theory. In 1991, Göbel et al. [GCV91] did an early work on temporal paths. While the terms temporal path or temporal graph are not used there, the concept of labeled edges and paths with ascending labels very closely resembles our notion of (strict) temporal paths, and is motivated by modeling an information flow problem. Berman [Ber96] considers temporal networks with edges with start time and finish time and shows that deciding the k-connectivity of temporal networks is NP-complete even for $k = 2$. Kempe et al. [KKK02] investigate non-strict paths, that is, paths with non-decreasing time labels, and analyze how central graph properties change with the addition of time. Xuan et al. [XFJ03] did an early work on algorithms that find optimal temporal paths (called 'journeys' there). In particular, algorithms for shortest, fastest and foremost paths are discussed. Wu et al. [Wu+16] find state-of-the-art algorithms for optimal paths. Based on the well-known breadth-first search which finds shortest paths in static graphs, Wu et al. show that shortest, foremost, fastest and reverse-foremost (strict) paths can be found in a similar fashion. This is a non-trivial result, because unlike shortest static paths, not every optimal temporal path is composed of optimal subpaths. Himmel et al. [Him+19] expand on the aforementioned work and study a more complex variation of temporal graphs with constraints on the waiting time in each vertex as well as non-strict walks, finding algorithms that solve this wider range of problems without substantial loss of speed when compared to Wu et al. [Wu+16]. Himmel et al. find efficient algorithms to find optimal *walks* but show that finding optimal *paths* is NP-complete in settings with

upper bounds on the waiting time.

While betweenness centrality in static graphs is a well studied concept, the study of betweenness centrality in temporal graphs is rather young. Tang et al. [Tan+10] argue that temporal graphs are more suitable to represent the dynamics of social and technical networks and introduce temporal variants of centrality metrics such as closeness and betweenness centrality. Building up on this, Tang et al. [Tan+11] use their notion of temporal closeness to analyze the containment of malware in mobile phone networks. Nicosia et al. [Nic+13] discuss temporal variants of betweenness and closeness centralities, as well as other temporal graph metrics.

Kim and Anderson [KA12] introduce the *time-ordered graph* as a representation of temporal graphs. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, the time-ordered graph is a directed acyclic graph where the vertex set contains T copies of the original vertex set V. Each copy is meant to represent a specific time step of the temporal graph's lifetime. The directed edges connect vertices of subsequent time steps such that only paths of strictly ascending times are possible. Based on this model, Kim and Anderson then study temporal centrality measures and use known algorithms for static directed graphs to give efficient algorithms. In particular, Kim and Anderson define the temporal betweenness centrality of a vertex based on shortest paths in the time-ordered graph. Notably, a path in the time-ordered graph may represent a non-path walk in the underlying temporal graph. We further discuss the difference in Section 2.3.1.

1.2 Organization of this thesis

We investigate how well the strategies found by Brandes [Bra01] translate to temporal graphs and the various concepts of optimal temporal paths. In Chapter 2, we formally introduce the definitions used throughout the thesis. In Chapter 3 we compare the concept of (static) shortest paths to the temporal counterparts. Most notably, we find that the recursive computations done by Brandes fail in the case of foremost and fastest paths. In fact, we show in Chapter 4 that the counting problems of foremost and fastest paths are #P-hard, and we show that this implies hardness for the corresponding temporal betweenness. This negatively answers our research question for these cases, since it is considered very unlikely that #P-hard problems are polynomial-time solvable. Nevertheless, we find a polynomial-time algorithm for shortest path betweenness in Chapter 5. We also introduce two subclasses of foremost paths for which we solve the counting problem in polynomial time as well. These results are used in Chapter 6 to devise algorithms for the betweenness variants based on shortest, shortest foremost, and prefix-foremost paths. In Chapter 7, we summarize our results and briefly discuss possible future research.

Chapter 2

Preliminaries

In this section, we present the central mathematical definitions and terminology that are used in the thesis.

Definition 2.1 (Temporal graph)**.** An undirected *temporal graph* is a triple $(V, \mathcal{E}, T)$ such that

- V is a set of vertices,
- $\mathcal{E} \subseteq \{ (\{ u, v \}, t) \mid u, v \in V, u \neq v, t \in [T] \}$ is a set of time edges, and
- $T \in \mathbb{N}$. We consider $[T] = \{ 1, \ldots, T \}$ as the set of time steps.

For a temporal graph $\mathcal{G}$, we use $V(\mathcal{G})$ to denote the set of vertices, $E(\mathcal{G})$ for the set of time edges, and $E_t(\mathcal{G})$ to denote the set of edges of $\mathcal{G}$ which are present at time step t, i.e., $E_t(\mathcal{G}) \coloneqq \{ \{ u, v \} \mid (\{ u, v \}, t) \in E(\mathcal{G}) \}$. For a time edge e, we use $t(e)$ to denote the time label of e.

In our definition of temporal graphs, only edges may appear and disappear over time, whereas vertices are present throughout the lifetime of the temporal graph. While the vertex itself does not change over time, its properties within the graph may do so. Hence, we will be interested in pairs of the form $(v, t) \in V(\mathcal{G}) \times [T]$. These are called *vertex appearances*.

Definition 2.2 (Temporal walk)**.** A *temporal walk* W on a temporal graph $\mathcal{G}$ from vertex s to z is an ordered sequence of transitions $(e_1, \ldots, e_k) \in E^k$ such that the endpoint of e_i is the starting point of e_{i+1} and $t(e_i) \leq t(e_{i+1})$ for each $i \in \{ 1, \ldots, k-1 \}$. We call a temporal walk *strict* if $t(e_i) < t(e_{i+1})$ for each $i \in \{ 1, \ldots, k-1 \}$.

A temporal walk may visit the same vertex more than once. In contrast to that, a temporal *path* visits each vertex at most once. This is analogous to the definitions for static graphs.

Definition 2.3 (Temporal path)**.** A *temporal path* $P = (e_i)_{i \in \{ t_1, \ldots, t_2 \}}$ is a temporal walk such that no vertex $v \in V(\mathcal{G})$ is start or endpoint of more than one transition e_i.

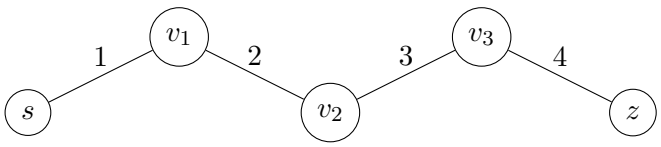

Figure 2.1: In this simple temporal graph, there is only one path from s to z. Notably, there is no path (and no walk) in the opposite direction due to the time labels.

We only consider undirected temporal graphs. However, a temporal walk is implicitly directed because of the ascending time labels; the opposite direction is no valid walk in general. Hence, we need a notion for directed *transitions* on a walk which indicate not only which time edge is used but also in which direction.

Definition 2.4 (Transition)**.** For any time edge $e = (\{v, w\}, t)$ we call (v, w, t) the *transition* from v to w at time step t. We call v the starting point and w the endpoint of the transition.

For readability, we use the notation $v \xrightarrow{t} w$ instead of the triple (v, w, t). Since the endpoint of a transition is equal to the starting point of the next one for any walk, we use a shortened notation omitting the doubled vertices. For instance, we denote the only temporal path from s to z in Figure 2.1 by:

$$P = (s \xrightarrow{1} v_1 \xrightarrow{2} v_2 \xrightarrow{3} v_3 \xrightarrow{4} z).$$

In this example, P is a *path* since all involved vertices are visited only once. Moreover, P is a *strict* path because the time labels are strictly ascending.

Brandes [Bra01] uses $P_s(v)$ to denote the set of vertices $u \in V$ such that u is a direct predecessor of v on a shortest path from s to v. We adapt this definition to temporal graphs.

Definition 2.5. Let $v \in V$ be any vertex. Then the set of predecessors is denoted as:

- $P_s^{(\mathrm{sh})}(v)$ for the predecessors of v on shortest paths
- $P_s^{(\mathrm{fm})}(v)$ for the predecessors of v on foremost paths
- $P_s^{(\mathrm{fa})}(v)$ for the predecessors of v on fastest paths

In order to count temporal paths, we will also need a notion of predecessors for vertex appearances instead of vertices. We define the predecessor of a vertex appearance (w, t') on a temporal path as the vertex appearance (v, t) where the path arrived last. For instance, on the path $(a \xrightarrow{1} b \xrightarrow{4} c)$, the vertex appearance $(b, 1)$ is the predecessor of $(c, 4)$.

Definition 2.6. Let $v \in V$ be any vertex. Then the set of predecessors is denoted as:

- $P_s^{(\mathrm{sh})}(v, t)$ for the predecessors of (v, t) on shortest paths
- $P_s^{(\mathrm{fm})}(v, t)$ for the predecessors of (v, t) on foremost paths
- $P_s^{(\mathrm{fa})}(v, t)$ for the predecessors of (v, t) on fastest paths

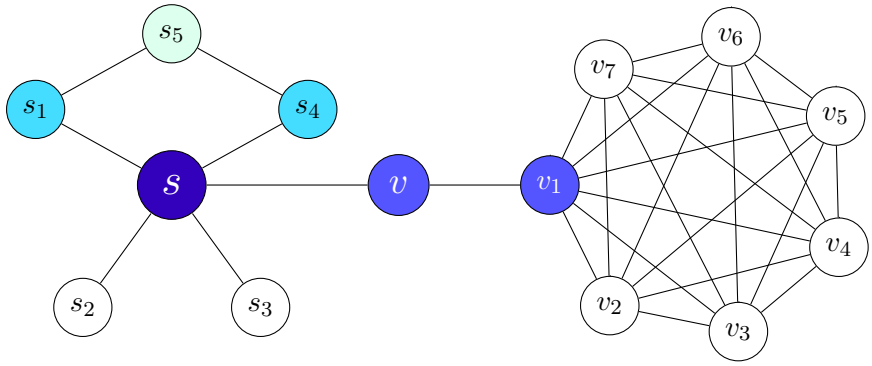

Figure 2.2: The colors indicate the betweenness centrality; the darker the color, the higher the betweenness. Vertices with high betweenness scores are typically those that serve as connections between many others.

2.1 Betweenness centrality

In static graphs, the *betweenness centrality* of a vertex measures how often this vertex is passed on shortest paths between pairs of vertices in the graph. Freeman [Fre77] defines the *betweenness centrality* $C_B(v)$ of a vertex v as

$$C_B(v) = \sum_{s \neq v \neq z} \frac{\sigma_{sz}(v)}{\sigma_{sz}}.$$

An example graph with betweenness values is shown in Figure 2.2. The separators s, v, and v_1 have the highest betweenness values, because each of them lies on any path between the left and the right part of the graph; the betweenness of s is the highest because it also connects $s_1, \ldots, s_4$ with each other. Notably, the example showcases that the betweenness does not generally depend on the degree—inside a clique, the shortest paths are always single edges, so no vertex is ever used as an intermediary step on shortest paths there. As a result, the clique vertices $v_2, \ldots, v_7$ all have a betweenness of zero, although they are connected to a high number of vertices. In contrast to that, the degree-2-vertex v has a very high betweenness because it is the only connection between the left and the right half of the graph. The vertex s_5 connects s_1 and s_4 but nothing else. There is another shortest path between s_1 and s_4, hence the betweenness of s_5 is 0.5. The betweenness values of s_1 and s_2 are higher, because either of them is on every path from s_5 to any other vertex in the graph.

2.2 Optimality of temporal paths

In static graphs, shortest paths are a central concept. In temporal graphs, there are different concepts of optimal paths. Figure 2.3 illustrates three of the most common optimization criteria.

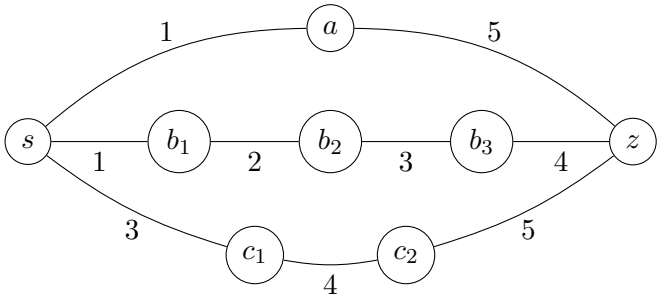

Figure 2.3: This temporal graph features a shortest, foremost, and fastest temporal path from s to z. Depending on the application, any of them may be considered optimal.

There are three paths from s to z:

- $P_1 = (s \xrightarrow{1} a \xrightarrow{5} z)$,
- $P_2 = (s \xrightarrow{1} b_1 \xrightarrow{2} b_2 \xrightarrow{3} b_3 \xrightarrow{4} z)$, and
- $P_3 = (s \xrightarrow{3} c_1 \xrightarrow{4} c_2 \xrightarrow{5} z)$.

Each of the paths examplifies one of the three most commonly used optimization criteria: the *shortest* path is P_1 because it uses the least amount of edges. The *foremost* path is P_2 because it has the earliest arrival time. Finally, P_3 is the *fastest* path because it minimizes the overall transition time. More formally, we use the following definitions:

Definition 2.7. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. Let $s, z \in V$ and let W be a temporal walk from s to z.

- W is a *shortest* walk if there is no walk W' from s to z such that W' contains less transitions than W.
- W is a *foremost* walk if there is no walk W' from s to z such that W' has a lower arrival time than W.
- W is a *fastest* walk if there is no walk W' from s to z such that the difference between arrival and start time is smaller for W' than it is for W.

Brandes [Bra01] uses σ_{sz} to denote the number of shortest paths from s to z in a static graph. We adapt this definition to temporal graphs and the notions of optimal paths defined above.

Definition 2.8. Let $\mathcal{G}$ be a temporal graph. For any $s, z \in V(\mathcal{G})$,

- $\sigma_{sz}^{(\mathrm{sh})}$ is the number of *shortest* paths,
- $\sigma_{sz}^{(\mathrm{fm})}$ is the number of *foremost* paths, and
- $\sigma_{sz}^{(\mathrm{fa})}$ is the number of *fastest* paths.

In addition to that, Brandes [Bra01] uses $\sigma_{sz}(v)$ to denote the number of shortest paths from s to z that pass through v in a static graph. Again, we adapt the notation to temporal graphs and the temporal optimality concepts.

Definition 2.9. Let $v \in V$ be any vertex, $t \in [T]$ a time step, and let $\star \in \{\,\mathrm{sh}, \mathrm{fm}, \mathrm{fa}\,\}$. Then,

- $\sigma_{sz}^{(\star)}(v)$ is the number of $\star$-optimal paths that pass through v, and
- $\sigma_{sz}^{(\star)}(v,t)$ is the number of $\star$-optimal paths that pass through v exactly at time step t, i.e., the paths that contain the transition $u \xrightarrow{t} v$ for some $u \in V$.

2.3 Temporal betweenness

As we have seen above, there are different notions of optimal paths (i.e., fastest, shortest, foremost) in temporal graphs. Thus, there are several options how to define *temporal betweenness centrality* based on any of these notions. We define temporal betweenness with respect to these different concepts of path optimality.

Definition 2.10. The temporal betweenness of any vertex $v \in V$ is given by:

$$C_B^{(\star)}(v) = \sum_{s \neq v \neq z} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}}, \star \in \{\,\mathrm{sh}, \mathrm{fa}, \mathrm{fm}\,\}.$$

Intuitively, betweenness centrality can be regarded as an indicator for the relative importance of a vertex in a graph. When we adapt this concept to temporal graphs, then we may want to measure how important a vertex is at a specified time. Thus, we define the betweenness centrality $C_B(v,t)$ of a vertex appearance (v,t).

Definition 2.11. The temporal betweenness of any vertex appearance $(v,t) \in V \times [T]$ is given by:

$$C_B^{(\star)}(v,t) = \sum_{s \neq v \neq z} \frac{\sigma_{sz}^{(\star)}(v,t)}{\sigma_{sz}^{(\star)}}, \star \in \{\,\mathrm{sh}, \mathrm{fa}, \mathrm{fm}\,\}.$$

2.3.1 Paths vs. walks

In our definition of temporal betweenness centrality we use the number of optimal paths as opposed to the number of optimal walks. This is natural for static graphs because in that case, shortest walks are always paths. In the temporal case, this is not always true—it is possible that there is a non-path walk that arrives at the same time as the foremost path, so the number of foremost paths and foremost walks between two vertices can be different. Kim and Anderson [KA12] define temporal betweenness based on the number of shortest paths in the time-ordered graph. Implicitly, this means that their definition uses strict temporal walks, not paths, because paths in the time-ordered graph do not necessarily represent paths in the original temporal graph.

Consider the graph shown in Figure 2.4. Clearly, every walk from the left half to the right passes either v_1 or v_2 and since the edges on the right are only present at exactly

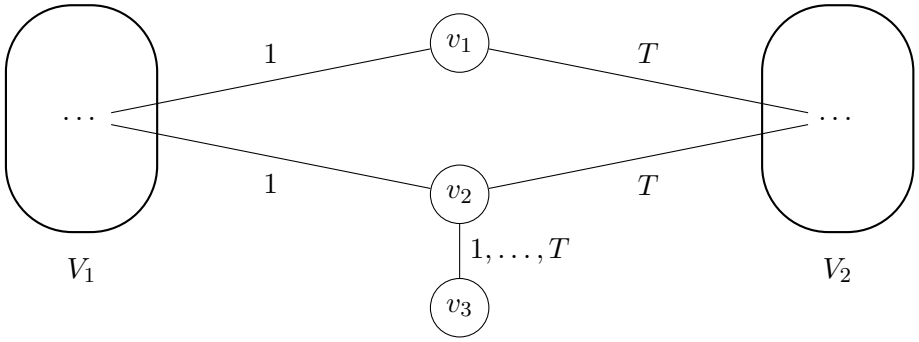

Figure 2.4: Assume V_1 and V_2 are sufficiently large sets of vertices that are adjacent to v_1 and v_2 at time steps 1 and T, respectively. Then v_1 and v_2 are vertices with high betweenness, v_3 has very low betweenness but high closeness.

one time step, every walk going from left to right is a foremost walk. Intuitively, v_1 and v_2 should have very similar, high betweenness scores, whereas v_3 should be close to zero. But if we use walks instead of paths for our definitions, we get a very high number of walks alternating between v_2 and v_3 before arriving on the right side, so v_2 and v_3 would get a high centrality score.

We conclude that paths are more suitable than walks as a base for the definition of temporal betweenness centrality.

2.3.2 Closeness centrality

We are mainly focused on temporal betweenness, but want to mention the similar *closeness centrality* measure that is also based on shortest paths. While betweenness measures how many shortest paths traverse a specified vertex, closeness measures how close that vertex is to all other vertices in the graph. More specifically, the closeness centrality of a vertex is calculated by taking the reciprocal of the average distance to all other vertices in the graph [Bav50].

We use the same example graph in Figure 2.4 to illustrate some differences of betweenness centrality and closeness centrality. As mentioned above, the vertices v_1 and v_2 are central in the graph in the sense that they connect the left and the right half. The betweenness of v_3 equals zero, since no path goes through v_3 at all. The closeness centrality of v_3, however, would be almost as high as that of the adjacent v_2, because it reaches every vertex in the network on paths with only one additional edge. Depending on the application, either of these two centrality measures can be the most appropriate to model the respective problem.

From an algorithmic point of view, betweenness centrality is presumably harder to compute. To compute the closeness, it is sufficient to determine the distances between vertices by finding some shortest path. In contrast to this, betweenness needs the count of optimal paths. In Chapter 4, we find a surprisingly strong discrepancy between finding optimal paths and counting them.

Chapter 3

Properties of temporal paths

In this chapter, we compare temporal graphs to static graphs with special regard to the respective notions of optimal paths. In particular, we identify some fundamental differences between shortest paths in static graphs on the one hand, and foremost/fastest paths in temporal graphs on the other hand, indicating that the optimal path counting problem for temporal graphs can be harder than for the static counterpart. We also discuss similarities between static shortest paths and temporal shortest paths. In particular, we consider two properties that are crucial for Brandes' algorithm [Bra01]: In Section 3.1, we discuss that optimal temporal paths may not be composed of optimal subpaths [Wu+16], which is an important difference to shortest static paths. In Section 3.2, we discuss the directed *predecessor graph* representation of a single-source walk through a temporal graph and show that this graph is acyclic.

In an undirected static graph, there is virtually no difference between a path from vertex v to w or the other way around. In a temporal graph, this is not the case. For example, the temporal graph shown in Figure 3.1 does have a path from s to z but no path from z to s due to the times in which the edges are present. The same example shows that temporal reachability is not transitive either: z can reach c and c can reach b, but z cannot reach b.

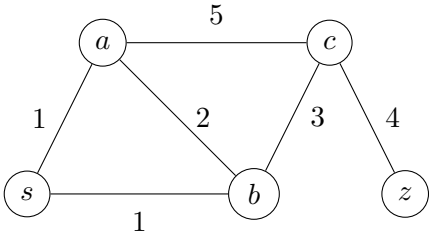

Figure 3.1: Although the edges are undirected, reachability is typically not symmetric in temporal graphs. In this example, s can reach z but z cannot reach s.

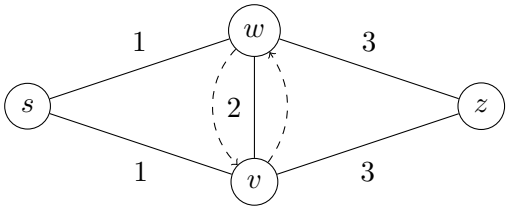

Figure 3.2: Consider the paths from s to z. All of them are foremost since the edges that are incident to z are only present at time 3. We observe that there are paths such that v is a predecessor of w and vice versa—the transitions $v \xrightarrow{2} w$ and $w \xrightarrow{2} v$ form a cycle. This is a major difference between foremost paths in temporal graphs and shortest paths in static graphs.

3.1 Subpath optimality

For shortest paths in static graphs, it is known that any subpath is a shortest path as well. This property is fundamental for many fast greedy algorithms such as breadth-first search since it allows us to find shortest paths by iteratively extending paths that have already been found, until the desired vertex is reached. In temporal graphs, this is typically not the case for many interesting path classes including fastest, foremost and even shortest paths [Wu+16]. Consider the graph shown in Figure 3.1 as an example.

- $(s \xrightarrow{1} a \xrightarrow{2} b \xrightarrow{3} c)$ is a foremost (and fastest) path from s to c, but its subpath $(s \xrightarrow{1} a \xrightarrow{2} b)$ is not foremost (nor fastest) because $(s \xrightarrow{1} b)$ arrives earlier.
- $(a \xrightarrow{2} b \xrightarrow{3} c \xrightarrow{4} z)$ is a shortest path from a to z, but its subpath $(a \xrightarrow{2} b \xrightarrow{3} c)$ is not shortest since $(a \xrightarrow{5} c)$ is shorter.

This example demonstrates that temporal optimal paths show less desirable properties with regard to computation than shortest paths in static graphs do. However, we show that for shortest paths there is a form of subpath optimality based on the vertex appearances instead of the vertices.

Lemma 3.1 (Prefix property for shortest temporal paths)**.** *Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. Let $P = (s \xrightarrow{t_1} \dots \xrightarrow{t'} y \xrightarrow{t} z)$ be a shortest path from s to (z, t). Then the prefix $P' = (s \xrightarrow{t_1} \dots \xrightarrow{t'} y)$ is a shortest path to (y, t').*

Proof. Assume $P = (s \xrightarrow{t_1} \dots \xrightarrow{t} z)$ is a shortest path but $P' = (s \xrightarrow{t_1} \dots \xrightarrow{t'} y)$ is not. Then there is a shorter path $P'' = (s \xrightarrow{t_1'} \dots \xrightarrow{t'} y)$ and $(P'' \xrightarrow{t} z)$ is shorter than P, contradiction. □

3.2 Acyclic predecessor relation

The second important property of shortest paths in static graphs is the fact that, given a fixed source $s \in V(\mathcal{G})$, the vertex set can be partially ordered by the distance from s. This is important for Brandes' algorithm [Bra01] because the recursive dependency acculation

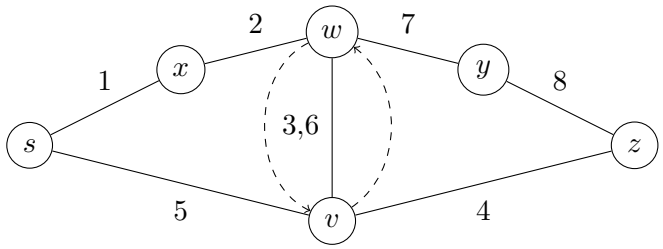

Figure 3.3: Consider the paths $(s \xrightarrow{5} v \xrightarrow{6} w \xrightarrow{7} y \xrightarrow{8} z)$ and $(s \xrightarrow{1} x \xrightarrow{2} w \xrightarrow{3} v \xrightarrow{4} z)$. Both are shortest paths and on the former, v is a predecessor of w, while on the latter, w is the predecessor of v. Note that $(s \xrightarrow{5} v \xrightarrow{4} z)$ is no valid path since $4 < 5$.

needs a base case which is given by the vertices with the highest distances. For the case of static shortest paths, Brandes [Bra01] discusses the *directed graph of shortest paths* from a fixed source s to any other vertex v in the graph and shows that this directed graph is always acyclic.

For foremost and fastest paths, this does not hold. In Figure 3.2 we demonstrate that for foremost (and thus for fastest) paths it is possible to get symmetric predecessor relationships for the paths from s to z. In our example, the paths $(s \xrightarrow{1} v \xrightarrow{2} w \xrightarrow{3} z)$ and $(s \xrightarrow{1} w \xrightarrow{2} v \xrightarrow{3} z)$ are both fastest and foremost. Clearly, in the former path v is a predecessor of w and in the latter w is a predecessor of v.

For shortest temporal paths, the same effect occurs as well, but the issue is salvageable. Figure 3.3 gives an example for symmetric predecessor relations on shortest paths. The two shortest paths use the transitions $v \xrightarrow{6} w$ and $w \xrightarrow{3} v$, i.e., go from v to w in one case and the opposite in the other one. Note that the two transitions have different time labels, which is not the case in our counter-example for foremost and fastest paths.

We define the directed *predecessor graph* for shortest paths and show that it is always acyclic.

Given a temporal graph $\mathcal{G}$, the vertex set of the predecessor graph is the set of *vertex appearances* in $\mathcal{G}$. The arc set is given by the ordered pairs of vertex appearances such that there is an optimal path that arrives in these vertex appearances in that order. That is, we add an arc from a vertex appearance (v, t) to another vertex appearance (w, t') if (v, t) is the predecessor of (w, t') on any shortest temporal path from s to any vertex appearance (z, t'').

Definition 3.2 (Predecessor graph)**.** Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. Let $s \in V$. The *predecessor graph* of $\mathcal{G}$ is the directed static graph given by

$$G_{\text{pre}}(s) = (V \times [T], E),$$

where

$$E = \{((v,t),(w,t')) \mid \exists \text{ shortest temporal path in } \mathcal{G} \text{ from } s \text{ to } (w,t') \\ \text{on which } (v,t) \text{ is a predecessor of } (w,t')\}.$$

An example for the construction is shown in Figure 3.4. In the example, the predecessor graph is acyclic. In fact, we show next that this is the case for the predecessor graph of any temporal graph.

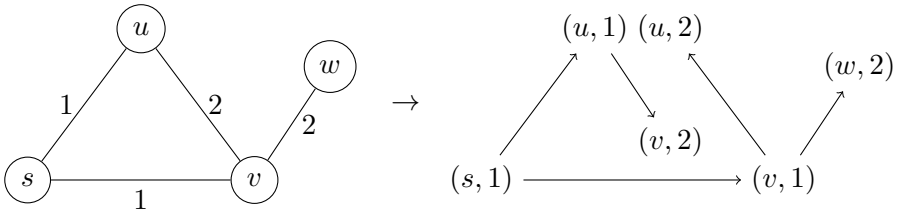

Figure 3.4: For any temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ and any source $s \in V$, the *shortest paths graph* $G^{(\mathrm{sh})}(s)$ is a directed acyclic graph (DAG). For simplicity's sake, unreachable vertex appearances are omitted in the figure.

Lemma 3.3 (Acyclic shortest paths)**.** *Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. Fix a source s. The directed temporal graph of the shortest paths from s to all reachable vertex appearances (v, t) is acyclic.*

Proof. Proof by contradiction. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph with source s. Let $\mathrm{dist} : V \times [T] \to \mathbb{N}$ be the function that gives the length of the shortest path in the predecessor graph $G_{\mathrm{pre}}(s)$ from s to any vertex appearance. By definition, for any vertex appearances (x, t) and (y, t'), if the predecessor graph contains the arc $((x, t), (y, t'))$, then $\mathrm{dist}(x, t) < \mathrm{dist}(y, t')$. Assume that $G_{\mathrm{pre}}(s)$ contains a cycle $C = ((v, t), \ldots, (v, t))$. Then $\mathrm{dist}(v, t) < \mathrm{dist}(v, t)$, contradiction. □

Together, Lemma 3.3 and Lemma 3.1 show that shortest paths in temporal graphs display similar behavior as shortest paths in static graphs. We use this in Chapter 5 to find an efficient algorithm for the corresponding betweenness measure. For foremost and fastest paths, however, we have given a first intuition that the respective counting problems are harder than for shortest paths. In fact, we prove in the next chapter that they are computationally hard.

Chapter 4

Hard counting problems in temporal graphs

In the previous chapter, we have seen that foremost and fastest paths have fundamentally different properties in comparison with shortest paths, and we have argued that this makes them hard to count at least on the base of greedy algorithms. In this chapter, we give a very brief overview of the theory of the computational complexity of counting problems. In particular, we discuss the complexity class #P and the notion of #P-hardness; we then show that several counting problems in the context of temporal betweenness centrality are #P-hard, implying that it is very unlikely that any of them can be solved in polynomial time.

4.1 Computational complexity of counting problems

For any function $f : \Sigma^* \to \mathbb{N}$, the corresponding *counting problem* is the following task: given input x, compute $f(x)$. Valiant [Val79] defines the complexity class #P as the class of counting problems corresponding to functions f such that there is a nondeterministic Turing machine with polynomial running time that has $f(x)$ accepting runs for a given input x. For example, the number of Hamiltonian paths in a graph or the number of satisfying truth assignments of a Boolean formula are in #P. The class #P is closed under polynomial-time Turing reductions, and a problem X is #P-hard if every problem A in #P Turing-reduces to X. A problem is called #P-complete if it is contained in #P and #P-hard.

Whether there are polynomial-time algorithms for #P-complete problems is a major unresolved research question, but it is strongly believed that there are none. The existence of a polynomial-time algorithm for any #P-hard problem would imply P = NP since counting the number of solutions clearly allows deciding whether there is any at all [Val79]. One of the most surprising results of computational complexity theory is the observation that easily polynomial-time-solvable decision problems correspond to hard counting problems [Val79].

4.2 #P-hard counting problems

In this section, we state counting problems related to temporal betweenness, including the computation of temporal betweenness itself, and show that they are #P-hard. In particular, counting all temporal paths is #P-hard for both strict and non-strict paths. The same holds true for foremost and fastest temporal paths, but not for shortest temporal paths. Our hardness results are based on the following two counting problems for which Valiant [Val79] shows #P-completeness:

Paths
Input: Static graph G, vertices s, z
Task: Count the number of paths from s to z.

Imperfect Matchings
Input: Bipartite static graph G
Task: Count the number of matchings (of any size) in G.

We use polynomial-time reductions from the two problems above to prove that counting problems related to temporal betweennes are #P-hard.

More specifically, we show the #P-hardness of the following problems:

Temporal Paths
Input: Temporal graph $\mathcal{G}$, vertices s, z
Task: Count the number of temporal paths from s to z in $\mathcal{G}$.

Strict Temporal Paths
Input: Temporal graph $\mathcal{G}$, vertices s, z
Task: Count the number of strict temporal paths from s to z in $\mathcal{G}$.

Foremost (Strict) Paths
Input: Temporal graph $\mathcal{G}$, vertices s, z
Task: Count the number of foremost (strict) temporal paths from s to z in $\mathcal{G}$.

Fastest (Strict) Paths
Input: Temporal graph $\mathcal{G}$, vertices s, z
Task: Count the number of fastest (strict) temporal paths from s to z in $\mathcal{G}$.

(Foremost/Fastest) (Strict) Temporal Betweenness
Input: Temporal graph $\mathcal{G}$, vertex v
Task: Calculate the betweenness centrality of v in $\mathcal{G}$.

For non-strict paths, the #P-hard problem Paths is contained as a special case in Temporal Paths, since any static graph can be transformed into an equivalent temporal graph with lifetime $T = 1$. Hence, we get the following result:

Proposition 4.1. Temporal Paths *is #P-hard.*

Proof. We reduce from Paths. Given a static graph $G = (V, E)$, consider the temporal graph $\mathcal{G} = (V, \mathcal{E}, 1)$ with $\mathcal{E} = \{ u \overset{1}{—} v \mid \{ u, v \} \in E \}$. Clearly, the number of temporal non-strict paths in $\mathcal{G}$ equals the number of paths in G. □

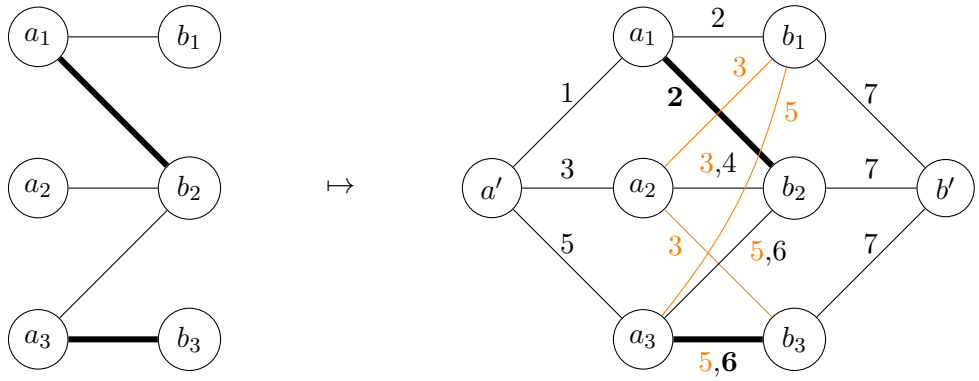

Figure 4.1: Given a static bipartite graph G, we construct a temporal graph $\mathcal{G}$ such that the number of matchings in G equals the number of strict paths from a' to b' in $\mathcal{G}$. In this example, the matching $\{\{a_1, b_2\}, \{a_3, b_3\}\}$ (highlighted in bold) translates to the path $(a' \xrightarrow{1} a_1 \xrightarrow{2} b_2 \xrightarrow{5} a_3 \xrightarrow{6} b_3 \xrightarrow{7} b')$.

Proposition 4.1 immediately implies the following:

Corollary 4.2. Foremost Paths *and* Fastest Paths *are #P-hard.*

Proof. Reduction from Temporal Paths. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ with a source s and a target z, construct $\mathcal{G}' = (V', \mathcal{E}', T')$ such that $T' = T+2$, $V' = V \cup \{s', z'\}$ and $\mathcal{E}' = \{u \overset{t+1}{—} v \mid u \overset{t}{—} v \in \mathcal{E}\} \cup \{s \overset{1}{—} s', z \overset{T+2}{—} z'\}$. By construction, every path from s' to z' is a foremost and fastest path and their number equals the number of s-z-paths in $\mathcal{G}$. □

This corollary is more surprising than Proposition 4.1. While counting all temporal paths was expected to be at least as hard as counting all static paths—which is #P-hard—the problem of counting all *shortest* paths in static graphs can be solved in polynomial time. Hence, the #P-hardness of Foremost Paths and Fastest Paths is a significant difference between optimal temporal paths on the one hand and shortest static paths on the other hand. However, counting the shortest temporal paths is possible in polynomial time.

The counting problem Strict Temporal Paths is #P-hard as well, but the proof is more technical since strict paths are fundamentally different from static paths, whereas non-strict paths could be regarded as a generalization of static paths, allowing for the simple argument used above. We show the #P-hardness of Strict Temporal Paths by reduction from Non-empty Matchings, where, given a static bipartite graph G, the task is to count all non-empty matchings in G. Valiant [Val79] only shows the #P-hardness of Imperfect Matchings, but clearly, the two problems are equally hard, since every (bipartite) graph has exactly one empty matching.

Theorem 4.3. Strict Temporal Paths *is #P-hard.*

Proof. We reduce from Matchings. Given a bipartite graph $G = (A \cup B, E)$, we construct a temporal graph $\mathcal{G} = (A \cup B \cup \{a', b'\}, \mathcal{E}, T)$ such that the number of matchings in G is equal to the number of strict paths from a' to b' in $\mathcal{G}$. An example for the transformation is shown in Figure 4.1.

The temporal edge set $\mathcal{E}$ is constructed as follows: For each edge $\{ a_i, b_j \} \in E$, we create a temporal edge $(\{ a_i, b_j \}, 2 \cdot i)$. These edges are meant to represent the edges of the original graph and will be called *forward-edges*. The vertices $a_i \in A$ are connected to a' at time step $2 \cdot i - 1$. All $b_j \in B$ are connected to b at the last time step T. For $i > 1$ we connect each a_i to each b_j at time step $2 \cdot i - 1$; these edges are shown in orange in Figure 4.1 and we will refer to them as *back-edges*.

We justify the terms *forward-edge* and *back-edge* by showing that for any path from a' to b', every transition with an even time label goes from an $a \in A$ to a $b \in B$ (*forward*) and exactly the other way (*back*) for every transition with an odd time label. By construction, each vertex $a_i \in A$ is incident to time edges with at most two time labels: $2 \cdot i - 1$ and $2 \cdot i$. Therefore, any path containing a transition $b_j \xrightarrow{2 \cdot i} a_i$ ends in a_i since no time edge with a higher time label will be available. By an anologous argument, back-edges cannot be used forward because it is impossible to arrive in a_i before time $2 \cdot i - 1$.

As a consequence, on any a'-b'-path every back-edge is followed by a forward-edge and every forward-edge is followed either by a back-edge or by the final edge to b'. Thus, for any matching $M = \{ \{ a_{i_1}, b_{j_1} \}, \ldots, \{ a_{i_m}, b_{j_m} \} \}$ of size $m \in \mathbb{N}^+$ there is exactly one a'-b'-path containing exactly the forward edges corresponding to M, and conversely, for each a'-b'-path P there is exactly one matching corresponding to the forward-edges in P. Thus, the number of non-empty matchings in G equals the number of a'-b'-paths in $\mathcal{G}$. □

Analogously to the case of non-strict paths, the #P-hardness of Strict temporal paths implies the #P-hardness of Strict Foremost Paths and Strict Fastest Paths.

Corollary 4.4. Strict Foremost Paths *and* Strict Fastest Paths *are #P-hard.*

We have shown that counting strict and non-strict temporal paths is #P-hard. This allows us to prove this chapter's main result.

Theorem 4.5. Temporal Betweenness *based on foremost or fastest, strict or non-strict paths is #P-hard.*

Proof. We prove the hardness by reduction from Temporal Paths. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph with vertices a and b. Let p be the number of temporal paths from a to b. We construct a temporal graph $\mathcal{G}' = (V', \mathcal{E}', T')$ with $V' = V \cup \{ a', b', v' \}$, lifetime $T' = T + 2$, and $\mathcal{E}' = \{ u \overset{t+1}{—} v \mid u \overset{t}{—} v \in \mathcal{E} \} \cup \{ a' \overset{1}{—} a, a' \overset{1}{—} v, v \overset{T+2}{—} b', b \overset{T+2}{—} b' \}$. The construction is illustrated in Figure 4.2.

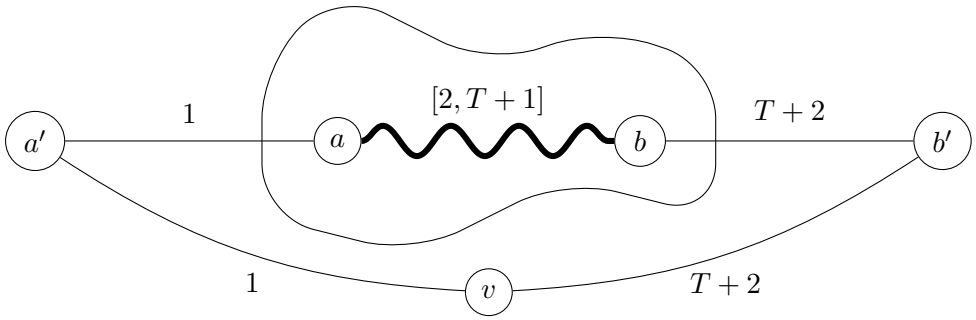

Figure 4.2: Given a temporal graph $\mathcal{G}$, we construct a temporal graph $\mathcal{G}'$ such that we can compute the number of a-b-paths in $\mathcal{G}$ from the betweenness of v in $\mathcal{G}'$.

By construction, every a'-b'-path is both fastest and foremost and there are exactly four (fastest/foremost) paths going through v, connecting the pairs (a, b),(a, b'),(a', b), and (a', b'), respectively:

1. $(a' \xrightarrow{1} v \xrightarrow{3} b')$,
2. $(a \xrightarrow{1} a' \xrightarrow{1} v \xrightarrow{3} b')$,
3. $(a \xrightarrow{1} a' \xrightarrow{1} v \xrightarrow{3} b' \xrightarrow{3} b)$, and
4. $(a' \xrightarrow{1} v \xrightarrow{3} b' \xrightarrow{3} b)$

We observe that for each of these four pairs, there are $p+1$ foremost (and fastest) paths, and for any other pair of vertices, no path goes through v at all. This allows us to compute p from the betweenness centrality of v.

$$C_B^{(\mathrm{fa})}(v) = C_B^{(\mathrm{fm})}(v) = \sum_{s \neq v \neq z} \delta_{sz}^{(\mathrm{fm})}(v) = 4 \cdot \delta_{ab}^{(\mathrm{fm})}(v) = \frac{4}{p+1}$$
$$\Rightarrow p = \frac{4}{C_B^{(\mathrm{fm})}(v)} - 1$$

□

Corollary 4.6. *Foremost and fastest, strict and non-strict dependencies and pair-dependencies are #P-hard.*

Proof. By definition, the betweenness centrality can be computed from the dependencies in linear time or from the pair-dependencies in quadratic time. Thus, calculating those is #P-hard as well. □

Theorem 4.5 negatively answers our research question for the cases of fast and foremost paths. For shortest paths, however, there is a polynomial-time algorithm. In the next chapter, we adapt Brandes' algorithm [Bra01] to the temporal case and give a polynomial-time algorithm for temporal betweenness centrality based on shortest paths.

Chapter 5

Adaptation of Brandes' algorithm

In the previous chapter, we have shown that temporal betweenness based on fastest or foremost paths is #P-hard, and we have identified properties of those path classes that are fundamentally different from static shortest paths, making them harder to count. There are, however, other concepts of optimal paths that are more promising, including shortest paths and specialized combinations of optimality. In this chapter, we investigate ways to adapt the approach of Brandes' algorithm [Bra01] to variants of temporal betweenness based on path classes for which the counting problem is not intractable in the first place. We discuss the main idea of Brandes' algorithm in Section 5.1. We then adapt the principle to the case of shortest temporal paths in Section 5.2. In Section 5.3, we consider subclasses of the generally intractable foremost paths—shortest foremost and prefix-foremost paths—and show that these subclasses can be counted efficiently.

5.1 Main idea

By definition, the betweenness centrality of a vertex v depends on the number of shortest paths between pairs of vertices which pass through v. Recall the definition of betweenness centrality:

$$C_B(v) = \sum_{s \neq v \neq z} \frac{\sigma_{sz}(v)}{\sigma_{sz}},$$

where σ_{sz} is the number of all shortest paths from s to z and $\sigma_{sz}(v)$ is the fraction of these paths which pass through v. Brandes [Bra01] calls the latter the *pair-dependency* of s and z on v. Brandes' main contribution is the observation that the betweenness centrality can be computed faster by first computing partial sums of the form

$$\delta_{s\bullet}(v) = \sum_{z \in V} \frac{\sigma_{sz}(v)}{\sigma_{sz}},$$

the so-called *dependency* of s on v. Figure 5.1 shows an example graph to illustrate the concepts of dependencies and pair-dependencies. The important result shown by Brandes is the fact that the dependencies obey a recursive relation. Hence, a dynamic program can compute the dependency of a vertex without the need to explicitly compute all pair-dependencies on that vertex. This reduces the running time on sparse graphs.

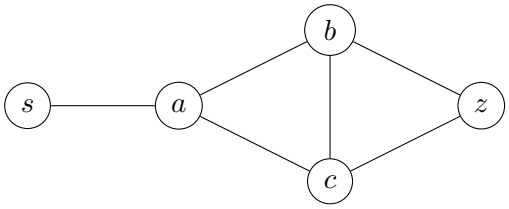

Figure 5.1: Consider the vertex s as the starting point and z as the target. Then a has a *pair-dependency* of $\delta_{sz}(a) = 1$, because both shortest s-z-paths go through a. The vertex b has the pair-dependency $\delta_{sz}(b) = 0.5$, because only one of the two shortest s-z-path goes through b (the other one goes through c). The *dependency* of s on a is $\delta_{s\bullet}(a) = 3$, because every shortest path to b, c and z goes through a. In contrast to that, the dependency of s on b is only $\delta_{s\bullet}(b) = \delta_{sz}(b) = 0.5$, because z is the only target where b lies on a shortest path.

Both concepts, dependencies and pair-dependencies, naturally generalize to optimal paths in temporal graphs.

Brandes [Bra01] defines the *pair-dependency* $\delta_{sz}(v)$ of vertices $s, z \in V$ on a vertex v as the fraction of shortest paths from s to z going through v, i.e., $\delta_{sz}(v) = \frac{\sigma_{sz}(v)}{\sigma_{sz}}$. Brandes then introduces the notion of *dependency* and shows a recursive relation. We adapt these definitions to the different notions of optimal temporal paths.

Definition 5.1 (Temporal dependencies)**.** Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. Let $s \in V$ be a source and let $z \in V$ be a target. Let $v \neq s$ and $v \neq z$. The temporal *pair-dependency* of s and z on v and the *dependency* of s on v are given by:

$$\left.\begin{aligned} \delta_{sz}^{(\star)}(v) &\coloneqq \tfrac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} \\ \delta_{s\bullet}^{(\star)}(v) &\coloneqq \textstyle\sum_{z \in V} \delta_{sz}^{(\star)}(v) \end{aligned}\right\} \star \in \{\,\mathrm{sh}, \mathrm{fa}, \mathrm{fm}\,\}.$$

In some cases, we may be interested in the dependency on a vertex at a specific time instead of the whole lifetime. Hence, we introduce dependencies on vertex appearances:

Definition 5.2 (Temporal dependencies)**.** Let s be a source and let z be a target. Let $v \neq s$ and $v \neq z$. For any $t \in [T]$, the temporal *pair-dependency* of s and z on (v, t) and the *dependency* of s on (v, t) are given by:

$$\left.\begin{aligned} \delta_{sz}^{(\star)}(v,t) &\coloneqq \tfrac{\sigma_{sz}^{(\star)}(v,t)}{\sigma_{sz}^{(\star)}} \\ \delta_{s\bullet}^{(\star)}(v,t) &\coloneqq \textstyle\sum_{z \in V} \delta_{sz}^{(\star)}(v,t) \end{aligned}\right\} \star \in \{\,\mathrm{sh}, \mathrm{fa}, \mathrm{fm}\,\}.$$

It is easy to see that the pair-dependency of any vertex v is the sum of the pair-dependencies of all appearances (v, t) of that vertex.

Lemma 5.3. *For any source s and target z, the pair-dependency of any given vertex v is given by:*

$$\delta_{sz}^{(\star)}(v) = \sum_{1 \leq t \leq T} \delta_{sz}^{(\star)}(v,t).$$

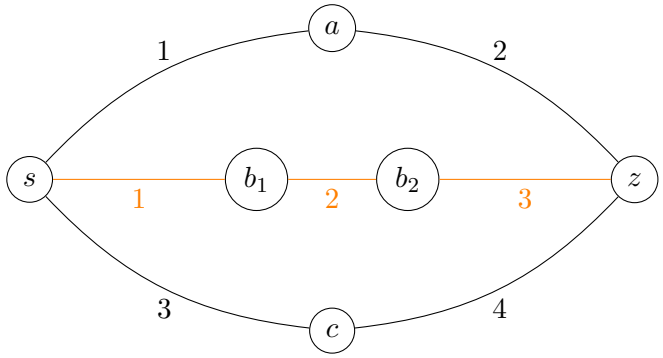

Figure 5.2: In this temporal graph, there are two shortest path from s to z, arriving at times 2 and 4, respectively. The path highlighted in orange is the shortest path from s to $(z, 3)$, but no shortest path from s to z, because it is longer than the other two.

Proof. Let $v \in V$. By definition,

$$\delta_{sz}^{(\star)}(v) = \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} = \frac{\sum_{1 \le t \le T} \sigma_{sz}^{(\star)}(v,t)}{\sigma_{sz}^{(\star)}} = \sum_{1 \le t \le T} \frac{\sigma_{sz}^{(\star)}(v,t)}{\sigma_{sz}^{(\star)}} = \sum_{1 \le t \le T} \delta_{sz}^{(\star)}(v,t).$$

□

Corollary 5.4. *For any source s, the dependency on a vertex v is given by:*

$$\delta_{sz\bullet}^{(\star)}(v) = \sum_{1 \le t \le T} \delta_{sz\bullet}^{(\star)}(v,t).$$

Lemma 5.5 (Temporal dependency)**.** *If there is exactly one optimal (i.e., shortest, fastest or foremost) path from $s \in V$ to each $v \in V$, then*

$$\delta_{s\bullet}^{(\star)}(v) = \sum_{w : v \in P_s^{(\star)}(w)} (1 + \delta_{s\bullet}^{(\star)}(w)).$$

Proof. If there is only one optimal path from s to each v, then $P_s^{(\star)}(v)$ contains exactly one element which is the unique predecessor of v. This implies that $\delta_{sv}^{(\star)}(u) \in \{0, 1\}$ for any vertex $u \in V$ since u can either lie on the unique path or not. If v is the predecessor of w, then it also lies on the unique path to every successor of w. This yields the recursion above. □

In the general case, the recursion is more complicated and we have to treat the different classes of optimal paths independently. We discuss shortest paths first, followed by two subclasses of foremost paths, for which the counting problem is tractable although counting all foremost paths is #P-hard.

5.2 Shortest paths

As discussed in Chapter 3, the predecessor relation of vertices on shortest temporal paths is not guaranteed to be anti-symmetric, but the predecessor relation of vertex appearances is. This allows us to find a recursion for the shortest paths to a vertex appearance (z,t), i.e., the shortest paths from s that arrive in z exactly at time t. We refer to those paths as t-shortest. To avoid confusion, we call shortest paths from s to z *all-time shortest*.

Definition 5.6. A path P from s to z is an *all-time shortest* path if there is no path P' from s to z which is shorter than P.
A path from s to z arriving at time t is a t-shortest path if there is no path P' from s to z which arrives at time t and is shorter than P.

Let $P_s^{\mathrm{sh}}((z,t)) = \{\, (w,t') \mid \exists \text{shortest path } (s \to \ldots \xrightarrow{t'} w \xrightarrow{t} z) \,\}$.

Lemma 5.7 (Counting of t-shortest paths)**.** *Let s be a source and let (z,t) be a vertex appearance with $s \neq z$. The number of t-shortest paths from s to z is given by:*

$$\sigma^{(\mathrm{sh})}_{s(z,t)} = \sum_{(w,t') \in P_s^{\mathrm{sh}}((z,t))} \sigma^{(\mathrm{sh})}_{s(w,t')}.$$

Proof. For any shortest path to (z,t) there is exactly one predecessor (w,t') and a final transition $w \xrightarrow{t} z$. We have shown in Lemma 3.1 that every prefix of a shortest path is a shortest path itself. Thus, the number of shortest paths to (z,t) equals the number of shortest paths to any of the predecessors. □

With the recursion above, we count the number of t-shortest paths. To compute the number of all-time shortest paths to the vertex v, we need to add up the numbers of t-shortest paths for all t where the shortest paths are also all-time shortest. This is not trivial because the length of shortest paths between two vertices can vary depending on the specified arrival time. In particular, this length is not monotonic in the time: Figure 5.2 illustrates a case where there are two paths from s to z of length 2 that arrive at times 2 and 4,respectively, whereas the only path arriving at time 3 has a length of 3.

Let $T_s^{(\mathrm{sh})}(v)$ be the set of time steps t where there is a globally shortest path from s to (z,t).

Lemma 5.8 (Temporal shortest path counting)**.** *Let s be a source and let z be a vertex appearance with $s \neq z$. The number of shortest paths from s to z arriving at any time is given by:*

$$\sigma^{(\mathrm{sh})}_{sz} = \sum_{t \in T_s^{(\mathrm{sh})}(z)} \sigma^{(\mathrm{sh})}_{s(z,t)}.$$

Lemma 5.9 (Dependency accumulation for shortest paths)**.** *Fix a source $s \in V$. For any vertex appearance $(v,t) \in V \times [T], v \neq s$ it holds:*

$$\delta^{(\mathrm{sh})}_{s\bullet}((v,t)) = \sum_{(w,t'):(v,t) \in P_s^{\mathrm{sh}}((w,t'))} \frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(w,t')}} \cdot (1 + \delta^{(\mathrm{sh})}_{s\bullet}((w,t'))).$$

Proof. Since we have shown the transition acyclicity in Chapter 3, the number of paths through v can be rewritten as the summed number of paths using any transition from v to any vertex w at some time step:

$$\delta^{(\mathrm{sh})}_{s\bullet}((v,t)) = \sum_{(z,t')\in V\times[T]} \delta^{(\mathrm{sh})}_{s(z,t')}((v,t)) = \sum_{(z,t')\in V\times[T]} \sum_{(w,t''):(v,t)\in P^{\mathrm{sh}}_s((w,t''))} \delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t''),$$

where $\delta^{(\mathrm{sh})}_{s(z,t)}(v,w,t'')$ is the fraction of shortest paths from vertex s to (z,t') that use the transition $v \xrightarrow{t''} w$. Analogously to Brandes' [Bra01] proof for Theorem 6, we distinguish three cases:

If $w = z$ and $t' = t''$, then $\delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t'') = \frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(z,t')}} = \frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(w,t'')}}$.

If $w = z$ but $t' \neq t''$, then $\delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t'') = 0$.

Otherwise, $\delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t'') = \frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(w,t'')}} \cdot \frac{\sigma^{(\mathrm{sh})}_{s(z,t')}((v,t))}{\sigma^{(\mathrm{sh})}_{s(z,t')}}$.

Inserting these cases into the term above yields the recursion we stated:

$$\begin{aligned}
&\sum_{(z,t')\in V\times[T]} \sum_{(w,t''):(v,t)\in P^{\mathrm{sh}}_s(w,t'')} \delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t'') \\
&= \sum_{(w,t''):(v,t)\in P^{\mathrm{sh}}_s(w,t'')} \sum_{(z,t')\in V\times[T]} \delta^{(\mathrm{sh})}_{s(z,t')}(v,w,t'') \\
&= \sum_{(w,t''):(v,t)\in P^{\mathrm{sh}}_s(w,t'')} \left(\frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(w,t')}} + \sum_{(z,t')\in(V\setminus\{w\})\times[T]} \frac{\sigma^{(\mathrm{sh})}_{s(z,t')}((v,t))}{\sigma^{(\mathrm{sh})}_{s(z,t')}} \right) \\
&= \sum_{(w,t''):(v,t)\in P^{\mathrm{sh}}_s(w,t'')} \frac{\sigma^{(\mathrm{sh})}_{s(v,t)}}{\sigma^{(\mathrm{sh})}_{s(w,t'')}} \cdot (1 + \delta^{(\mathrm{sh})}_{s\bullet}(w,t'')).
\end{aligned}$$

□

5.3 Specialized optimality concepts

We have shown in Chapter 4 that counting all foremost paths between a given pair of vertices is intractable. In this section, we consider subclasses of foremost paths that can be counted efficiently.

As an additional motivation, consider the example given in Figure 5.3: the temporal graph consists of a large clique C that is connected to a source s at time 1 and to a target z at time T. Clearly, every path from s to z is a foremost path since any ingoing time edge in z is only present at time T. This leads to an extremely high number of foremost paths, including all the Hamiltonian paths through the clique C. If every path is foremost, we may consider this property rather meaningless, and ask for relevant subclasses of foremost paths. More specifically, we consider *shortest foremost paths* and *prefix-foremost paths*.

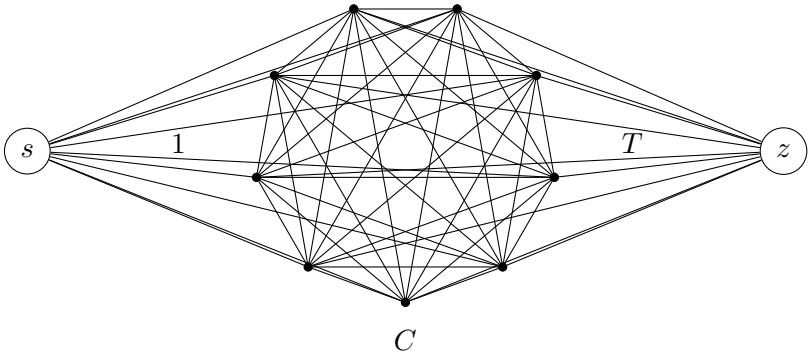

Figure 5.3: Assume C is a (large) clique throughout the lifetime of the temporal graph, s is connected to all $v \in C$ at time step 1 and z is connected to all $v \in C$ at time step T. Then every path from s to z is a foremost path.

5.3.1 Shortest foremost paths

A shortest foremost path is a foremost path that is not longer than any other foremost path. It may not be a shortest path overall, however, as even shorter paths with higher arrival times may exist. Shortest foremost paths can be regarded as paths that prioritize a low arrival time and use a low edge count as a tie-breaker.

Definition 5.10. Let $s, z \in V$. A path P from s to z is a shortest foremost path if P is a foremost path from s to z and for all foremost paths P' from s to z it holds: $|P| \leq |P'|$.

We observe that a shortest foremost path is a t-shortest path for the earliest possible arrival time. Hence, our findings in the previous section translate very easily to shortest foremost paths. In fact, the algorithm for shortest paths discussed in Chapter 6 computes the shortest foremost paths as well.

5.3.2 Prefix-foremost paths

In the example given in Figure 5.3, every path from s to z is foremost because z can only be reached very late. On an intuitive level, this means that a path may waste a lot of time early on without any impact on the final arrival time because there is a bottleneck immediately before the target. In this section, we consider the paths that do not show this behavior. More specifically, we consider the class of foremost paths for which every prefix path is foremost as well. That is, every vertex that is visited by such a path is visited as soon as possible. We call this class of paths *prefix-foremost paths* and consider a variation of betweenness centrality based on prefix-foremost paths.

Definition 5.11. Let $s, z \in V$. A path $P = (e_1, \ldots, e_k)$ from s to z is a *prefix-foremost* path if P is a foremost path and every prefix $P' = (e_1, \ldots, e_{k'}), k' \leq k$ is a foremost path as well.

Wu et al. [Wu+16] show that there is always at least one prefix-foremost path between any pair of vertices s and z unless z is not reachable from s at all.

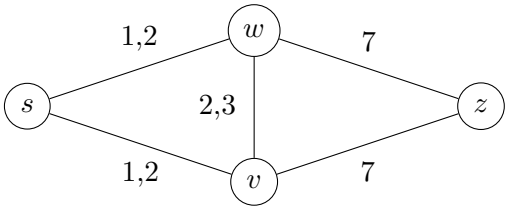

Figure 5.4: There are 10 foremost paths from s to z but only 2 prefix-foremost paths.

Lemma 5.12 (Existence of prefix-foremost paths [Wu+16])**.** *Let $s, z \in V$. If there is a foremost path from s to z, then there is a prefix-foremost path from s to z.*

In Figure 5.4, there are 10 strict foremost paths from s to z, but only 2 of them are prefix-foremost ($(s \xrightarrow{1} v \xrightarrow{7} z)$ and $(s \xrightarrow{1} w \xrightarrow{7} z)$). A strict prefix-foremost path cannot visit both v and w because the latter would be reached too late. In contrast to that, non-strict paths can visit both v and w and still be prefix-foremost. In fact, non-strict prefix-foremost paths still show the undesirable properties of general foremost paths.

Proposition 5.13. NON-STRICT PREFIX-FOREMOST PATHS *is #P-hard.*

Proof. The proposition follows with the same reductionas as used in the proof of Corollary 4.4. □

For strict prefix-foremost paths, we show that the counting problem and the calculation of the corresponding betweenness are solvable in polynomial time by adapting Brandes' algorithm [Bra01].

Let $\sigma_{sz}^{(\mathrm{p})}$ denote the number of strict prefix-foremost paths from s to z.

Lemma 5.14 (Strict prefix-foremost dependency accumulation)**.** *Fix a source $s \in V$. For any vertex $v \in V, v \neq s$ it holds:*

$$\delta_{s\bullet}^{(\mathrm{p})}(v) = \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot (1 + \delta_{s\bullet}^{(\mathrm{p})}(w)).$$

Proof. Let $\sigma_{sz}^{(\mathrm{p})}(v, w, t)$ be the number of prefix-foremost paths from vertex s to z that use the transition (v, w, t).

$$\delta_{s\bullet}^{(\mathrm{p})}(v) = \sum_{z \in V} \delta_{sz}^{(\mathrm{p})}(v) = \sum_{z \in V} \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \delta_{sz}^{(\mathrm{p})}(v, w, t)$$

The first step is just using Definition 5.1. The second is true because the paths going through v are exactly the paths using the transition (v, w, t) for some w. Note that for each w there is exactly one t here since we only count the prefix-foremost paths.

Let v and w be vertices such that v is predecessor of w on a prefix-foremost path from s to w. Let $t + 1$ be the foremost arrival time in w.

If $w = z$, then $\delta_{sz}^{(\mathrm{p})}(v, w, t) = \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sz}^{(\mathrm{p})}} = \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}}$.

Otherwise, the dependency on (v, w, t) is the fraction of paths going from v to w multiplied by the fraction of paths going through w at all.
This number is given by $\frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot \delta_{sz}^{(\mathrm{p})}(w)$.

Inserting the two cases into the equations above yields the recursion we stated:

$$\begin{aligned}
\sum_{z \in V} \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \delta_{sz}^{(\mathrm{p})}(v, w, t) &= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \sum_{z \in V} \delta_{sz}^{(\mathrm{p})}(v, w, t) \\
&= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \left(\delta_{sz}^{(\mathrm{p})}(v, z, t) + \sum_{z \in V \setminus \{ w \}} \delta_{sz}^{(\mathrm{p})}(v, w, t) \right) \\
&= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \left(\frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} + \sum_{z \in V \setminus \{ w \}} \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot \delta_{sz}^{(\mathrm{p})}(w) \right) \\
&= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \left(\frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} + \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot \sum_{z \in V \setminus \{ w \}} \delta_{sz}^{(\mathrm{p})}(w) \right) \\
&= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot (1 + \sum_{z \in V \setminus \{ w \}} \delta_{sz}^{(\mathrm{p})}(w)) \\
&= \sum_{w : v \in P_s^{(\mathrm{p})}(w)} \frac{\sigma_{sv}^{(\mathrm{p})}}{\sigma_{sw}^{(\mathrm{p})}} \cdot (1 + \delta_{s\bullet}^{(\mathrm{p})}(w))
\end{aligned}$$

□

Chapter 6

Algorithms for temporal betweenness

We have shown that shortest, shortest foremost, and strict prefix-foremost paths have sufficiently convenient properties to allow dynamic counting and dependency accumulation in a similar way as done by Brandes' algorithm [Bra01] for the case of static graphs. We use our findings to give algorithms for the three betweenness variants discussed in Chapter 5, starting with Shortest Temporal Betweenness.

In the unweighted case, Brandes' algorithm [Bra01] is based on breadth-first search. For each vertex, the Single-source shortest paths problem is solved once. At the end of each iteration, the dependency of the respective s on all other vertices v is added to their betweenness scores.

Definition 6.1 (Temporal neighborhood)**.** For a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ and a vertex $v \in V(\mathcal{G})$, we call $N_t^{\mathcal{G}}(v) \coloneqq \{\, u \in V(\mathcal{G}) \mid \{\, u, v \,\} \in E_t(\mathcal{G}) \,\}$ the *temporal neighborhood* of v at time step $t \in [T]$.

Our algorithms for temporal betweenness will follow roughly the same structure. Instead of breadth-first search, the appropriate algorithm for the respective path class is used as the base for the algorithm.

6.1 Shortest temporal betweenness

In Section 5.2, we have shown that the temporal dependencies for shortest paths follow a similar recursion as for static graphs. Recall that we have to consider vertex appearances as opposed to vertices, because the necessary conditions of subpath-optimality and acyclicity are not guaranteed on the vertex level.

Therefore, Algorithm 1 uses a $|V| \times T$-table to store the number of shortest paths to all vertex appearances instead of vertices. The overall structure of the algorithm is similar to Brandes' algorithm [Bra01]—a single-source-all-shortest-paths traversal from each vertex to all vertex appearances is performed and the count of shortest paths is stored in the aforementioned table. At the end of each iteration, we add the dependencies found for each vertex appearance to the betweenness score of the respective vertex. Note that there may be t-shortest paths to vertex appearances which are no all-time-shortest

paths; hence, the algorithm needs to check whether these paths are actually relevant for the betweenness score.

In order to count shortest foremost paths instead of (all-time) shortest paths, only the dependency accumulation needs to be changed: instead of checking whether the paths have minimal length, the algorithm checks for minimal arrival time. In fact, Algorithm 1 can be modified such that it computes both variants at the same time, without increasing the (asymptotical) running time.

Proposition 6.2. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ with $n = |V|$ and $M = |\mathcal{E}|$, Algorithm 1 computes the shortest-path betweenness and the shortest-foremost-path betweenness in $\mathcal{O}(n \cdot M)$ time.*

Proof. The correctness follows from Lemma 5.7 and Lemma 5.9 since our algorithm dynamically computes the values given by the recursive formulas we developed in Chapter 5.

Each iteration consists of two major parts: the single-source traversal of the temporal graph, and the accumulation of the dependencies found for the respective source. The first part is dominated by the loop over the queue of vertex appearances. Vertex appearances are only added to the queue if there is an incident temporal edge. Since each vertex appearance is only added and removed once, and only if it is incident to at least one time edge, this yields an upper bound for the running time of $\mathcal{O}(M)$ for this part. The dependency accumulation has the same asymptotical running time since it just iterates over each vertex appearance visited during the first part. Since both parts are done for each vertex once, the overall running time is in $\mathcal{O}(n \cdot M)$. □

Algorithm 1 Shortest betweenness in temporal graphs

Input: Temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$
Output: Betweenness $C_B^{(\mathrm{sh})}$ and $C_B^{(\mathrm{fm})}$ of all vertices $v \in V(\mathcal{G})$

for $s \in V$ **do**
  for $v \in V$ **do** ▷ Initialization
    $P[v] \leftarrow \emptyset$
    $\mathrm{dist}[v] \leftarrow -1$
    $\sigma[v] \leftarrow 0$
    $\delta[v] \leftarrow 0$
  end for
  $\mathrm{dist} \leftarrow \mathrm{BFS}(s)$ ▷ Compute minimal distances to all vertices
  $S \leftarrow$ empty stack
  $Q \leftarrow$ empty queue
  $Q \leftarrow \mathrm{enqueue}(s, 0)$
  while Q not empty **do**
    $(v, t) \leftarrow \mathrm{dequeue}(Q)$
    for $(w, t') \in N(v, t)$ with $t < t'$ **do** ▷ $t \leq t'$ for non-strict
      if $\mathrm{dist}[w, t'] = -1$ **then** ▷ First arrival in w at time t'
        $\mathrm{dist}[w, t'] \leftarrow \mathrm{dist}[v, t] + 1$
        if $\mathrm{dist}[w, t'] > \mathrm{dist}[w]$ **then** ▷ No shortest path to w
          continue
        end if
        $S \leftarrow \mathrm{push}(w, t')$
        $Q.\mathrm{enqueue}(w, t')$
      end if
    end for
    if $\mathrm{dist}[w, t'] = dist[v, t] + 1$ **then**
      $\sigma[w, t'] \leftarrow \sigma[w, t'] + \sigma[v, t]$
      $P[w, t'] \leftarrow P[w, t'] \cup \{ (v, t) \}$
    end if
  end while
  while $(w, t') \leftarrow \mathrm{pop}(S)$ **do**
    for $(v, t) \in P[w, t']$ **do**
      if $(\mathrm{dist}[v, t] = \mathrm{dist}[v])$ **then** ▷ separate t-shortest from all-time-shortest
        $\delta[v, t] \leftarrow \delta[v, t] + \frac{\sigma[v,t]}{\sigma[w,t']} \cdot (1 + \delta[w, t'])$
      end if
      if $(t = t_{\min}[v])$ **then** ▷ only count shortest foremost here
        $\delta^{(\mathrm{sh\,fm})}[v, t] \leftarrow \delta[v, t] + \frac{\sigma[v,t]}{\sigma[w,t']} \cdot (1 + \delta[w, t'])$
      end if
    end for
    $C_B^{(\mathrm{sh})}[v] \leftarrow C_B^{(\mathrm{sh})}[v] + \delta[v, t]$
    $C_B^{(\mathrm{sh\,fm})}[v] \leftarrow C_B^{(\mathrm{sh\,fm})}[v] + \delta^{(\mathrm{sh\,fm})}[v, t]$
  end while
end for
return $C_B^{(\mathrm{sh})}$, $C_B^{(\mathrm{sh\,fm})}$

6.2 Strict prefix-foremost betweenness

Algorithm 2 modifies Brandes' algorithm to count strict prefix-foremost paths instead of shortest (static paths). The overall structure of the algorithm remains the same. Instead of iterating over all neighbors, however, we add add all temporal edges of a vertex into a priority queue (prioritizing early time labels). This allows us to traverse the graph in a time-respecting manner and find the prefix-foremost paths. The dependency-accumulation in lines 23—28 is analogous to Brandes.

Proposition 6.3. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ with $n = |V|$ and $M = |\mathcal{E}|$, Algorithm 2 computes the prefix-foremost path betweenness of all $v \in V$ in $\mathcal{O}(n \cdot M \cdot \log M)$ time.*

Proof. The correctness follows from Lemma 5.7 and Lemma 5.14 since our algorithm dynamically computes the values given by the recursive formulas we developed in Chapter 5.

The outer loop (line 1) iterates over n vertices. The body of the loop is dominated by the iteration over the transitions in the priority queue (line 11). Since each temporal edge is added and removed at most once, this yields a runnning time of $\mathcal{O}(M \cdot \log M)$, under the assumption that adding and removing elements from a priority takes logarithmical time. The overall running time of the algorithm is then in $\mathcal{O}(n \cdot M \cdot \log M)$. □

Algorithm 2 Strict prefix-foremost betweenness

Input: Temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$
Output: Betweenness $C_B^{(\mathrm{p})}$ of all vertices $v \in V(\mathcal{G})$

for $s \in V$ **do**
  for $v \in V$ **do** ▷ Initialization
    $P[v] \leftarrow \emptyset$
    $t_{\min}[v] \leftarrow -1$
    $\sigma[v] \leftarrow 0$
    $\delta[v] \leftarrow 0$
  end for
  $S \leftarrow$ empty stack
  $Q \leftarrow$ empty priority queue ▷ Transitions prioritized by time label
  $Q \leftarrow \mathrm{enqueueAll}(\{\, s \xrightarrow{t} v \mid s \overset{t}{—} v \in \mathcal{E} \,\})$
  while Q not empty **do**
    $v \xrightarrow{t} w \leftarrow \mathrm{dequeue}(Q)$
    if $t_{\min}[w] = -1$ **then** ▷ First and foremost arrival in w
      $t_{\min}[w] \leftarrow t$
      $S \leftarrow \mathrm{push}(w)$
      $Q.\,\mathrm{enqueueAll}(\{\, w \xrightarrow{t'} x \mid w \overset{t'}{—} x \in \mathcal{E}, t < t' \,\})$
    end if
    if $t_{\min}[w] = t'$ **then**
      $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
      $P[w] \leftarrow P[w] \cup \{\, v \,\}$
    end if
  end while
  while $w \leftarrow \mathrm{pop}(S)$ **do**
    for $v \in P[w]$ **do**
      $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$
    end for
    $C_B^{(\mathrm{p})}[v] \leftarrow C_B^{(\mathrm{p})}[v] + \delta[v]$
  end while
end for
return $C_B^{(\mathrm{p})}$

Chapter 7

Conclusion

We have investigated several variants of temporal betweenness centrality based on the various optimization criteria for temporal paths. We have shown a surprising discrepancy in their computational complexity: while some variations are #P-hard, others can be computed in polynomial time. More specifically, we found that counting foremost, and thus fastest paths, is #P-hard, and in turn, the computation of the corresponding betweenness centrality scores is #P-hard as well. In contrast to that, counting the shortest temporal paths is possible to do in polynomial time both for strict and non-strict paths. The same is true for the shortest foremost paths. In the case of prefix-foremost paths, we found a polynomial-time algorithm for the strict version, whereas the non-strict version is again #P-hard.

For the temporal betweenness variants which we found tractable, we have given algorithms. Following the main idea of Brandes' algorithm, we devised algorithms that count shortest, shortest foremost, and prefix-foremost paths, respectively, and calculate the betweenness scores of all vertices based on the respective paths. We expect the algorithm for *prefix-foremost* paths to be fast enough to be practical on real-world graphs. While the algorithm for *shortest foremost* paths has a good asymptotical running time, the space complexity of the table for all vertex appearances will presumably be too demanding for large networks with both many vertices and a high number of time steps. In the case of sparse temporal graphs, however, most of the vertex appearances are unreachable and the entry in the table is not needed. Hence, using better data structures instead of a fixed-size table could greatly reduce the amount of required memory.

An important practical question will be how well our betweenness variants capture the properties of real-world networks, that is, how reliable a high temporal betweenness score will correspond to nodes that are actually of particular interest in their network. Experimentation with real-world networks is required to measure and compare the quality of the betweenness variants we discussed.

Another relevant research question might be the parameterized complexity of the variants which we have shown to be hard. If graph parameters or graph classes can be found for which even the hard variants can be computed fast, this could relativize the theoretical hardness of the general case. Furthermore, it could be investigated whether the intractable variants can be approximated in practical running time. For example, counting the foremost paths is #P-hard, but we have shown that counting the strict

prefix-foremost paths can be done in polynomial time. Further research could be done to determine how large a difference there is between these two metrics in real-world graphs and whether the number of prefix-foremost paths could serve as an approximation or heuristic for the generally intractable number of all foremost paths.

In addition to that, even more betweenness variants can be defined and studied. In particular, all of our temporal betweenness variants are based on temporal *paths* as opposed to the more general *walks*, and we did not consider any restriction of the waiting time in a vertex. Himmel et al. [Him+19] mention that in specific settings, the computation of a temporal *path* is NP-hard, whereas temporal *walks* can still be computed in polynomial time. Hence, variations of temporal betweenness based on walks instead of paths may be more suitable in those cases, and a research question could again ask for the discrepancy between the different variations.

Literature

[Bad+07] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. "Approximating Betweenness Centrality". In: *Algorithms and Models for the Web-Graph*. International Workshop on Algorithms and Models for the Web-Graph. Ed. by A. Bonato and F. R. K. Chung. San Diego: Springer Berlin Heidelberg, 2007, pp. 124–137 (cit. on p. 10).

[Bav50] A. Bavelas. "Communication Patterns in Task-Oriented Groups". In: *The Journal of the Acoustical Society of America* 22.6 (Nov. 1, 1950), pp. 725–730 (cit. on p. 18).

[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. 10).

[Bra01] U. Brandes. "A faster algorithm for betweenness centrality*". In: *The Journal of Mathematical Sociology* 25.2 (June 2001), pp. 163–177 (cit. on pp. 9–11, 14, 16, 17, 19–21, 27, 29, 30, 33, 35, 37).

[DH07] E. M. Daly and M. Haahr. "Social Network Analysis for Routing in Disconnected Delay-tolerant MANETs". In: *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc '07. event-place: Montreal, Quebec, Canada. New York, NY, USA: ACM, 2007, pp. 32–40 (cit. on p. 9).

[Fre77] L. C. Freeman. "A Set of Measures of Centrality Based on Betweenness". In: *Sociometry* 40.1 (1977), pp. 35–41 (cit. on pp. 10, 15).

[GCV91] F. Göbel, J. O. Cerdeira, and H. J. Veldman. "Label-connected graphs and the gossip problem". In: *Discrete Mathematics* 87.1 (Jan. 19, 1991), pp. 29–40 (cit. on p. 10).

[Heu+10] M. P. van den Heuvel, R. C. W. Mandl, C. J. Stam, R. S. Kahn, and H. E. Hulshoff Pol. "Aberrant Frontal and Temporal Complex Network Structure in Schizophrenia: A Graph Theoretical Analysis". In: *Journal of Neuroscience* 30.47 (Nov. 24, 2010), pp. 15915–15926 (cit. on p. 9).

[Him+19] A.-S. Himmel, M. Bentert, A. Nichterlein, and R. Niedermeier. "Efficient Computation of Optimal Temporal Walks under Waiting-Time Constraints". In: *arXiv:1909.01152 [cs]* (Aug. 30, 2019). arXiv: 1909.01152 (cit. on pp. 10, 44).

[KA12] H. Kim and R. Anderson. "Temporal node centrality in complex networks". In: *Physical Review E* 85.2 (Feb. 13, 2012) (cit. on pp. 11, 17).

[KKK02] D. Kempe, J. Kleinberg, and A. Kumar. "Connectivity and Inference Problems for Temporal Networks". In: *Journal of Computer and System Sciences* 64.4 (June 1, 2002), pp. 820–842 (cit. on p. 10).

[Ley07] L. Leydesdorff. "Betweenness centrality as an indicator of the interdisciplinarity of scientific journals". In: *Journal of the American Society for Information Science and Technology* 58.9 (July 2007), pp. 1303–1319 (cit. on p. 9).

[Nic+13] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. "Graph Metrics for Temporal Networks". In: *Temporal Networks*. Ed. by P. Holme and J. Saramäki. Understanding Complex Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 15–40 (cit. on p. 11).

[ŞB09] Ö. Şimşek and A. G. Barto. "Skill Characterization Based on Betweenness". In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 1497–1504 (cit. on p. 9).

[Tan+09] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. "Temporal Distance Metrics for Social Network Analysis". In: *Proceedings of the 2nd ACM Workshop on Online Social Networks*. WOSN '09. New York, NY, USA: ACM, 2009, pp. 31–36 (cit. on p. 9).

[Tan+10] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia. "Analysing Information Flows and Key Mediators Through Temporal Centrality Metrics". In: *Proceedings of the 3rd Workshop on Social Network Systems*. SNS '10. event-place: Paris, France. New York, NY, USA: ACM, 2010, 3:1–3:6 (cit. on p. 11).

[Tan+11] J. Tang, C. Mascolo, M. Musolesi, and V. Latora. "Exploiting temporal complex network metrics in mobile malware containment". In: *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks. June 2011, pp. 1–9 (cit. on p. 11).

[Val79] L. G. Valiant. "The Complexity of Enumeration and Reliability Problems". In: *SIAM Journal on Computing* 8.3 (Aug. 1979), pp. 410–421 (cit. on pp. 23–25).

[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 (Nov. 2016), pp. 2927–2942 (cit. on pp. 10, 19, 20, 34, 35).

[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* 14.2 (Apr. 1, 2003), pp. 267–285 (cit. on p. 10).