RESEARCH ARTICLE

Algorithmic aspects of temporal betweenness[†]

Sebastian Buß[1], Hendrik Molter[1,2], Rolf Niedermeier[1] and Maciej Rymar[1]

[1]Algorithmics and Computational Complexity, Technical University of Berlin, Berlin, Germany and [2]Department of Computer Science, Ben-Gurion University of the Negev, Be'er Sheva, Israel
Corresponding author: Hendrik Molter; Email: molterh@post.bgu.ac.il

Abstract

The *betweenness centrality* of a graph vertex measures how often this vertex is visited on shortest paths between other vertices of the graph. In the analysis of many real-world graphs or networks, the betweenness centrality of a vertex is used as an indicator for its relative importance in the network. In particular, it is among the most popular tools in social network analysis. In recent years, a growing number of real-world networks have been modeled as *temporal graphs* instead of conventional (static) graphs. In a temporal graph, we have a fixed set of vertices and there is a finite discrete set of time steps, and every edge might be present only at some time steps. While shortest paths are straightforward to define in static graphs, temporal paths can be considered "optimal" with respect to many different criteria, including length, arrival time, and overall travel time (shortest, foremost, and fastest paths). This leads to different concepts of *temporal betweenness centrality*, posing new challenges on the algorithmic side. We provide a systematic study of temporal betweenness variants based on various concepts of optimal temporal paths.

Computing the betweenness centrality for vertices in a graph is closely related to counting the number of optimal paths between vertex pairs. While in static graphs computing the number of shortest paths is easily doable in polynomial time, we show that counting foremost and fastest paths is computationally intractable (#P-hard), and hence, the computation of the corresponding temporal betweenness values is intractable as well. For shortest paths and two selected special cases of foremost paths, we devise polynomial-time algorithms for temporal betweenness computation. Moreover, we also explore the distinction between strict (ascending time labels) and non-strict (non-descending time labels) time labels in temporal paths. In our experiments with established real-world temporal networks, we demonstrate the practical effectiveness of our algorithms, compare the various betweenness concepts, and derive recommendations on their practical use.

1. Introduction

Graph metrics such as betweenness centrality are studied and applied in many application areas, including social and technological network analysis (Leydesdorff, 2007; Tang et al., 2009), wireless routing (Daly and Haahr, 2007), machine learning (Şimşek and Barto, 2009), and neuroscience (van den Heuvel et al., 2010). The *betweenness centrality* of a vertex in a graph measures how often this vertex is visited by a shortest (or optimal) path. High betweenness centrality scores are usually associated with vertices that can be seen as more important for the network. In static

---

[†]An extended abstract of this paper appeared in the proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20) (Buß et al., 2020). This version contains full proof details and a description of an alternative algorithmic approach of computing betweenness values on a suitable static expansion. It further contains an empirical comparison of our algorithms (in terms of running time) to the alternative approach.

graphs, betweenness centrality is a well-studied concept. It is well-known that Brandes' algorithm (Brandes, 2001)[1] computes the betweenness centrality of all vertices of a given (unweighted) static graph with n vertices and m edges in $O(n \cdot m)$ time and $O(n + m)$ space.

In temporal graphs, that is, graphs with fixed vertex set and edge sets varying over discrete time steps, the notion of betweenness centrality can be defined in a similar fashion. However, 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") (Bui-Xuan et al., 2003). These three types of optimality are widely studied in the temporal setting (Bui-Xuan et al., 2003; Casteigts et al., 2015; Bentert et al., 2020); however, many other natural optimality concepts may exist. From a motivational standpoints, shortest temporal paths may be considered optimal in cases where edge model connections in a transportation network and changing connections are accompanied by some significant overhead, such as in the case of changing flights, or in the case of the transportation of goods on container ships. In the case where no such overhead exists, finding fastest connections is well-motivated, corresponding to fastest temporal paths. In the scenario of disease spreading, one is usually interested in transmission routes with the earliest arrival, corresponding to foremost temporal paths. The corresponding temporal betweenness score then quantifies the "importance" of a station (in the case of transportation) or an individual (in the case of disease spreading).

For each of these three path types, that is, *shortest*, *foremost*, and *fastest*, we can define and study a variant of temporal betweenness centrality. In addition, combinations of optimality criteria such as *shortest foremost* temporal paths can be considered. Furthermore, we will distinguish between temporal paths with strictly or non-strictly ascending time labels on the edges.

In this work, we investigate algorithmic aspects of temporal betweenness based on shortest, foremost, and fastest paths. In addition, we also consider two subtypes of foremost paths, namely shortest foremost and so-called *prefix-foremost* paths. We analyze the computational complexity of computing the temporal betweenness with each of the mentioned optimality concepts both in the strict and the non-strict setting.

Related work. There is an enormous amount of work on the concept of betweenness centrality in static graphs, as already indicated by the huge citation numbers concerning Brandes's path-breaking algorithm (Brandes, 2001). Betweenness centrality was defined in 1977 by Freeman (1977). We refrain from further discussing the static case which is already treated in many textbooks.

The theory of temporal graphs is comparatively young (Holme, 2015; Holme and Saramäki, 2013, 2019; Michail, 2016; Latapy et al., 2018) but strongly growing in many directions. We focus our discussion of related work on temporal walks, paths, and the computation of temporal betweenness centrality.

Bui-Xuan et al. (2003) did an early work on algorithms that find optimal temporal paths (called "journeys" there). In particular, they presented algorithms for shortest, fastest, and foremost temporal paths. Afterward, Wu et al. (2016) provided state-of-the-art algorithms for optimal temporal paths. Based on breadth-first search which finds shortest paths in static graphs, Wu et al. (2016) showed that shortest, foremost, fastest, and reverse-foremost (strict) temporal paths can be found in a similar fashion. Bentert et al. (2020) and Casteigts et al. (2021) expanded on the work of Wu et al. (2016) and studied a more complex variation of temporal paths and walks with constraints on the waiting time in each vertex. Bentert et al. (2020) contributed efficient algorithms to find optimal temporal *walks* but Casteigts et al. (2021) showed that finding optimal temporal *paths* is NP-hard 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. (2010) argued that temporal graphs are more suitable to represent the dynamics of social and technical networks and introduced temporal variants of centrality metrics such as closeness and betweenness centrality based on foremost temporal paths. Building on this, Tang et al. (2011) used their notion of temporal closeness to

analyze the containment of malware in mobile phone networks. Nicosia et al. (2013) also discussed temporal variants of betweenness and closeness centralities, as well as other temporal graph metrics. They mostly give an overview on different definitions. Kim and Anderson (2012) defined the temporal betweenness centrality of a vertex based on shortest paths in the so-called *static expansion*, which is a (static) directed graph that models the connectivity properties of the corresponding temporal graph. They give a polynomial-time algorithm for computing the temporal betweenness values. Afrasiabi Rad et al. (2017) studied foremost walk temporal betweenness and observed #P-hardness, presented an exponential-time algorithm, and conducted corresponding experiments. Tsalouchidou et al. (2020) consider an arbitrary linear combination of a path's length and duration as an optimality criterion and compute the temporal betweenness centrality with respect to such paths and the help of static expansions. Recently, Rymar et al. (2021) defined a quite general property for optimality concepts of temporal paths which they call *prefix compatible*. It is a property that insures that the corresponding temporal betweenness can be computed in polynomial time. Enright et al. (2023) study the computational complexity of counting temporal paths. The problem is #P-hard and closely related to computing the temporal betweenness based on foremost or fastest temporal paths.

We finish by pointing to several temporal graph surveys (Latapy et al., 2018; Holme, 2015; Holme and Saramäki, 2013, 2019) that already provide some definitions of temporal betweenness centrality. We point out that most works on temporal betweenness (Tsalouchidou et al., 2020; Alsayed and Higham, 2015; Habiba et al., 2007; Kim and Anderson, 2012; Rad et al., 2017; Tang et al., 2010) employ static expansions and do not directly work on the temporal graphs. Also, they usually do not make the point of distinguishing between strict and non-strict paths.

Our contributions. Our main research question is as follows: How hard—theoretically and practically—is the computation of the different variants of temporal betweenness centrality?

We investigate the computational complexity of various canonical temporal betweenness variants and show remarkable differences in their computational complexity: while some of them can be computed in polynomial time, others are computationally hard. Note that computing betweenness values is closely related to counting optimal paths since, informally speaking, the betweenness value of a vertex quantifies how many optimal paths go through this vertex. While it is straightforward to see that computing temporal betweenness values can be reduced to counting optimal temporal paths, we show that the converse also holds: computing the temporal betweenness based on foremost or fastest temporal paths is computationally as hard as counting foremost or fastest temporal paths.

We show that counting foremost and fastest temporal paths is #P-hard, implying the same hardness result for the computation of corresponding betweenness concepts. Since temporal betweenness based on foremost temporal paths is arguably the best-motivated variant in many application areas (Tang et al., 2010; Rad et al., 2017) and we obtain intractability, we investigate two modifications, namely shortest foremost and (strict) prefix-foremost temporal paths. For temporal betweenness based on these modified versions of foremost temporal paths as well as shortest temporal paths, we obtain tractability results.

For the polynomial-time-solvable variants of temporal betweenness, we aim to provide algorithms inspired by Brandes' algorithm (Brandes, 2001), which is based on computing so-called *dependencies* and accumulate them with a BFS-like computation. We lift these concepts to the temporal setting and thus provide a theoretical basis for developing algorithms that directly work on the temporal graph rather than on its static expansion.

Adapting algorithms from the static directly to the temporal setting has proven successful also in other contexts, such as for example clique enumeration (Himmel et al., 2017; Bentert et al., 2019; Molter et al., 2020).

Remarkably, for the case of strict prefix-foremost temporal paths, we show that the corresponding temporal betweenness can be computed significantly more efficient than for the other tractable cases.

Table 1. An overview of the computational complexity of the temporal betweenness variants we consider, both for the strict and the non-strict cases. Here, *n* refers to the number of vertices, *M* refers to the total number of time edges, and *T* refers to the total number of time steps

	Strict	Non-strict
Shortest	$O(n \cdot M \cdot T)$	$O(n \cdot M \cdot T)$
Foremost	#P-hard	#P-hard
Fastest	#P-hard	#P-hard
Prefix-foremost	$O(n \cdot M \cdot \log M)$	#P-hard
Shortest foremost	$O(n \cdot M \cdot T)$	$O(n \cdot M \cdot T)$

In Table 1, we give an overview of our theoretical findings. We present formal (worst-case) computational hardness proofs (Section 3). In the cases where we state polynomial-time computability, we provide algorithms that compute the temporal betweenness scores of all vertices in a given temporal graph (Sections 4 and 5). We provide a thorough formal study of the computational complexity landscape of temporal betweenness centrality, altogether obtaining a fairly complete picture concerning the computation of temporal betweenness.

To complement our theoretical findings, we implement and compare several of our algorithms (in terms of running time, distribution of the betweenness values, and vertex rankings induced by the betweenness values). This aims to provide an empirical exhibition on the differences between the betweenness concepts we study. We also compare our algorithms (in terms of running time) to the alternative (naïve) approach of computing betweenness values on a suitable static expansion.

Organization of the paper. In Section 2, we provide the basic notation used in this paper as well as some preliminary observations. In Section 3, we prove computational hardness for some variants of temporal betweenness (see Table 1) and in Sections 4 and 5, we provide theoretical foundations and algorithms for the remaining temporal betweenness concepts. In Section 6, we present our experimental evaluation of the algorithms described in Section 5. We conclude in Section 7.

2. Preliminaries & basic observations

In this section, we introduce the most important mathematical definitions and terminology used in our work. We further present some basic observations.

2.1 Temporal graphs and paths

We use the following formal definition and notation for temporal graphs.

Definition 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* $T \in \mathbb{N}$*, where* $[T] = \{1, \ldots, T\}$ *is a 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., $E_t(\mathcal{G}) := \{\{u, v\} \mid (\{u, v\}, t) \in E(\mathcal{G})\}$. For a time edge e, we use $t(e)$ to denote the time label of e. We call $V(\mathcal{G}) \times [T]$ the set of *vertex appearances*.

We only consider undirected temporal graphs. However, temporal paths and walks are implicitly directed because of the ascending time labels. Hence, we need a notion for directed *transitions* on a temporal path or walk which indicates not only which time an edge is used but also in which
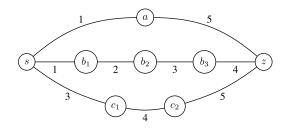
Figure 1. This temporal graph features a *shortest*, a *foremost*, and a *fastest* temporal path from s to z.

direction. 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.

Using this, we can now define temporal walks and temporal paths.

Definition 2 (Temporal Walk). *A temporal walk W on a temporal graph $\mathcal{G}$ from vertex s to vertex 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 $t(e_i) \le t(e_{i+1})$ 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 3 (Temporal Path). *A temporal path $P = (e_i)_{i \in [k]}$ is a temporal walk such that every vertex $v \in V(\mathcal{G})$ is starting point of at most one transition e_i and endpoint of at most one transition $e_{i'}$ for some $i, i' \in [k]$.*

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 "middle" temporal path from s to z in Fig. 1 by:

$$P = \left(s \xrightarrow{1} b_1 \xrightarrow{2} b_2 \xrightarrow{3} b_3 \xrightarrow{4} z \right).$$

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

2.2 Optimality of temporal walks and paths

In static graphs, shortest paths are a central concept. In temporal graphs, there are different concepts of optimal paths. Fig. 1 illustrates three of the most common optimization criteria: *shortest*, *foremost*, and *fastest* (Bui-Xuan et al., 2003).

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

More formally, we use the following definitions:

Definition 4. *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 an earlier 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.*

A temporal *path* is shortest, foremost, or fastest if it is a shortest, foremost, or fastest temporal walk, respectively. Note that for the optimality criteria, "foremost" and "fastest" optimal temporal walks are not necessarily temporal paths. Imagine a temporal graph where all time edges incident to s have the same time label and also all time edges incident to z have the same time label. Then, every walk from s to z (if it exists) is foremost and fastest, since all walks leave s at the same time and arrive at z at the same time.

Together with the distinction between strict and non-strict, this gives us *six* different types of optimal temporal paths. In the following, we use the term "$\star$-optimal" temporal path, where $\star$ denotes the type.

Next, we introduce terminology and notation for counting optimal temporal paths.

Definition 5. *Let $\mathcal{G}$ be a temporal graph. For any $s, z \in V(\mathcal{G})$, $\sigma_{sz}^{(\star)}$ is the number of $\star$-optimal temporal paths from s to z.*

We set $\sigma_{vv}^{(\star)} := 1$. We further introduce terminology and notation for counting optimal temporal paths that visit a certain vertex or a certain vertex appearance.

Definition 6. *Let $v \in V$ be any vertex and let $t \in [T]$ be a time step. 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 arrive at v exactly at time step t, that is, the paths that contain the transition $u \xrightarrow{t} v$ for some $u \in V$.*

We set $\sigma_{sz}^{(\star)}(s) := \sigma_{sz}^{(\star)}$ and $\sigma_{sz}^{(\star)}(z) := \sigma_{sz}^{(\star)}$. We define $\sigma_{sz}^{(\star)}(s, 0) := \sigma_{sz}^{(\star)}$, $\sigma_{sz}^{(\star)}(s, t) := 0$ for all $t \neq 0$, and $\sigma_{sz}^{(\star)}(v, 0) := 0$ for all $v \neq s$. Defining these corner cases as above allows us to keep certain proofs simpler by avoiding to discuss the corner cases explicitly. Intuitively, we assume that we have a dummy vertex appearance $(s, 0)$ for every temporal path starting at s that we consider to be the vertex appearance that the path arrives at time step zero.

2.3 Temporal 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 (1977) defines the *betweenness centrality* $C_B(v)$ of a vertex v (in a connected graph) as

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

As we have seen above, there are different notions of optimal paths (for example, fastest, shortest, foremost) in temporal graphs. Thus, there are several options how to define *temporal betweenness centrality* based on any of these notions. Moreover, we do not want to assume that there is a temporal path from any vertex to any other vertex in the graph. That is, we assume that there are vertex pairs s, z with $\sigma_{sz} = 0$ which we want to leave out when summing over all vertex pairs. To formalize this, we use a *connectivity matrix A* of the temporal graph: let A be a $|V| \times |V|$

matrix, where for every $v, w \in V$ we have that $A_{v,w} = 1$ if there is a temporal path from v to w, and $A_{v,w} = 0$ otherwise. Note that $A_{s,z} = 1$ implies that $\sigma_{sz} \neq 0$.

Formally, temporal betweenness based on these different concepts of path optimality is defined as follows.

Definition 7 (Temporal Betweenness). *The temporal betweenness of any vertex $v \in V$ is given by:*

$$C_B^{(\star)}(v) := \sum_{s \neq v \neq z \text{ and } A_{s,z} = 1} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}}.$$

For our work, we use a slightly different version of temporal betweenness which is defined for vertex appearances and show that the temporal betweenness as defined above can be easily computed from the modified version. This allows us to simplify some of our proofs. We drop the condition that, when summing over all vertex pairs, these vertices have to be different from the vertex of which we want to know the betweenness. Formally, we define

$$\hat{C}_B^{(\star)}(v, t) := \sum_{s,z \in V \text{ and } A_{s,z} = 1} \frac{\sigma_{sz}^{(\star)}(v, t)}{\sigma_{sz}^{(\star)}}.$$

We can observe that using the connectivity matrix A for the temporal graph, we can compute the temporal betweenness $C_B^{(\star)}(v)$ from the modified temporal betweenness values of the vertex appearances (v, t).

Lemma 1. *For any vertex $v \in V$ it holds*

$$C_B^{(\star)}(v) = \sum_{t \in [T] \cup \{0\}} \hat{C}_B^{(\star)}(v, t) - \sum_{w \in V} (A_{v,w} + A_{w,v}) + 1.$$

Proof. We show the claim as follows.

$$C_B^{(\star)}(v) = \sum_{s \neq v \neq z \text{ and } A_{s,z} = 1} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}}$$

$$= \sum_{s,z \in V \text{ and } A_{s,z} = 1} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} - \sum_{s \in V \text{ and } A_{s,v} = 1} \frac{\sigma_{sv}^{(\star)}(v)}{\sigma_{sv}^{(\star)}}$$

$$- \sum_{z \in V \text{ and } A_{v,z} = 1} \frac{\sigma_{vz}^{(\star)}(v)}{\sigma_{vz}^{(\star)}} + \frac{\sigma_{vv}^{(\star)}(v)}{\sigma_{vv}^{(\star)}}$$

$$= \sum_{s,z \in V \text{ and } A_{s,z} = 1} \sum_{t \in [T] \cup \{0\}} \frac{\sigma_{sz}^{(\star)}(v, t)}{\sigma_{sz}^{(\star)}} - \sum_{s \in V} A_{s,v} - \sum_{z \in V} A_{v,z} + 1$$

$$= \sum_{t \in [T] \cup \{0\}} \hat{C}_B^{(\star)}(v, t) - \sum_{w \in V} (A_{v,w} + A_{w,v}) + 1 \qquad \square$$

We remark that in contrast to the static setting, temporal *walks* (visiting vertices multiple times) can also be optimal for certain canonical optimality criteria. In our definition of temporal betweenness centrality, we use the number of optimal temporal paths as opposed to the number of optimal temporal walks. This is natural for static graphs because static shortest walks are always paths. In the temporal case, this is not always true—it is possible that there is a non-path temporal walk (visiting vertices multiple times) that arrives at the same time as the foremost temporal path, so the number of foremost temporal paths and foremost temporal walks between two vertices can be different. Consider the graph shown in Fig. 2. Clearly, every walk from the left half to the right
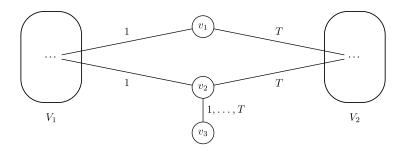
Figure 2. Assume V_1 and V_2 are sets of vertices that are adjacent to v_1 and v_2 at time steps 1 and T, respectively. Then, v_1 and v_2 are vertices with high temporal betweenness based on temporal paths, whereas v_3 has very low temporal betweenness based on temporal paths. However for, example, foremost temporal walks, v_3 would also have a high temporal betweenness.

passes either v_1 or v_2 and since the edges on the right are only present at exactly 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 for defining temporal betweenness centrality.

3. Computationally hard temporal betweenness variants

In this section, we present counting problems closely related to temporal betweenness, including the computation of temporal betweenness itself, and show that they are #P-hard. In particular, we will show that 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 (1979) showed #P-completeness:[2]

PATHS
Input: A static graph $G = (V, E)$, two vertices $s, z \in V$.
Task: Count the number of different paths from s to z in G.

IMPERFECT MATCHINGS
Input: A bipartite static graph.
Task: Count the number of different matchings (of any size) in G.

We use polynomial-time counting reductions[3] from the two problems above to prove that counting problems related to temporal betweenness are #P-hard. More specifically, we show the #P-hardness of the following problems:

(STRICT) TEMPORAL PATHS
Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, two vertices $s, z \in V$.
Task: Count the number of (strict) temporal paths from s to z in $\mathcal{G}$.

FOREMOST (STRICT) PATHS
Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, two vertices $s, z \in V$.
Task: Count the number of foremost (strict) temporal paths from s to z in $\mathcal{G}$.

FASTEST (STRICT) PATHS
Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, two vertices $s, z \in V$.
Task: Count the number of fastest (strict) temporal paths from s to z in $\mathcal{G}$.
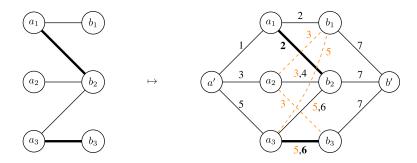
Figure 3. Given a static bipartite graph G, we construct a temporal graph G such that the number of matchings in G equals the number of strict paths from a' to b' in G. In this example, the matching $\{\{a_1, b_2\}, \{a_3, b_3\}\}$ (highlighted in bold) translates to the path $\left(a' \xrightarrow{1} a_1 \xrightarrow{2} b_2 \xrightarrow{5} a_3 \xrightarrow{6} b_3 \xrightarrow{7} b'\right)$.

(Foremost/Fastest) (Strict) Temporal Betweenness
Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, a vertex $v \in V$.
Task: Compute the betweenness centrality of v in $\mathcal{G}$.

As also observed by Afrasiabi Rad et al. (2017), 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 $T = 1$. Furthermore, each temporal path in this instance is also foremost and fastest. Hence, we have the following:

Proposition 2. Temporal Paths, Foremost Paths, *and* Fastest Paths *are #P-hard.*

The counting problem Strict Temporal Paths is #P-hard as well, but the proof is more demanding 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 a polynomial-time counting reduction from Imperfect Matchings.

Theorem 3. Strict Temporal Paths *is #P-hard.*

Proof. We reduce from the #P-complete problem Imperfect Matchings (Valiant, 1979). Given a bipartite graph $G = (A \uplus B, E)$, we construct a temporal graph $\mathcal{G} = (A \cup B \cup \{a', b'\}, \mathcal{E}, T)$ such that the number of *non-empty* matchings in G is equal to the number of strict temporal paths from a' to b' in $\mathcal{G}$. An example for the transformation is shown in Fig. 3 and it is easy to check that the transformation can be computed in polynomial time.

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 drawn dashed in orange in Fig. 3 and we will refer to them as *back-edges*.

We justify the terms *forward-edge* and *back-edge* by showing that for any temporal 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 from a $b \in B$ to an $a \in A$ (*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 $2 \cdot i$. Hence, any temporal 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 analogous argument, back-edges cannot be used forwardly because it is impossible to arrive in a_i before time $2 \cdot i - 1$.

As a consequence, on any temporal path from a' to b', 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
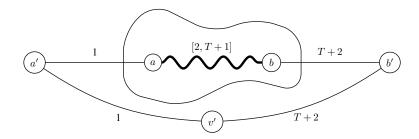
Figure 4. Given a temporal graph *G*, we construct a temporal graph *G'* such that we can compute the number of *a-b*-paths in *G* from the betweenness of *v* in *G'*.

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 temporal path from a' to b' containing exactly the forward edges corresponding to M, and conversely, for each temporal path P from a' to b' 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 temporal paths from a' to b' in $\mathcal{G}$ which implies that we have a polynomial-time counting reduction. □

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. 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 section's main result.

Theorem 5. TEMPORAL BETWEENNESS *based on foremost or fastest, strict or non-strict paths is #P-hard.*

Proof. We prove the #P-hardness by a polynomial-time counting reduction from (STRICT) TEMPORAL PATHS. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph with vertices a and b. Let p be the number of (strict) 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 $\mathcal{E}' = \left\{ u \xrightarrow{t+1} v \mid u \xrightarrow{t} v \in \mathcal{E} \right\} \cup \left\{ a' \xrightarrow{1} a, a' \xrightarrow{1} v', v' \xrightarrow{T+2} b', b \xrightarrow{T+2} b' \right\}$. The construction is illustrated in Fig. 4 and can clearly be computed in polynomial time.

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

1. $\left(a' \xrightarrow{1} v' \xrightarrow{T+2} b' \right)$,

2. $\left(a \xrightarrow{1} a' \xrightarrow{1} v' \xrightarrow{T+2} b' \right)$,

3. $\left(a \xrightarrow{1} a' \xrightarrow{1} v' \xrightarrow{T+2} b' \xrightarrow{T+2} b \right)$, and

4. $\left(a' \xrightarrow{1} v' \xrightarrow{3} b' \xrightarrow{T+2} b \right)$.

We observe that for each of these four pairs, there are $p + 1$ foremost (and fastest) (strict) temporal paths, and for any other pair of vertices, no temporal path goes through v at all. This allows us to compute p from the temporal betweenness centrality based on foremost (fm) or fastest (fa) temporal paths of v in the following way. We define

$$\delta_{sz}^{(\mathrm{fm})}(v) := \begin{cases} 0, & \text{if } \sigma_{sz}^{(\mathrm{fm})} = 0 \\ \dfrac{\sigma_{sz}^{(\mathrm{fm})}(v)}{\sigma_{sz}^{(\mathrm{fm})}}, & \text{otherwise.} \end{cases}$$

Then, we have:

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

$$\Rightarrow p = \frac{4}{C_B^{(\mathrm{fm})}(v)} - 1.$$

This implies that we have a polynomial-time counting reduction. $\qquad\square$

4. Counting temporal paths and temporal dependencies

In the previous section, we have shown that computing the temporal betweenness centrality based on fastest or foremost paths is #P-hard. Now we investigate ways to extend the approach of Brandes' algorithm (Brandes, 2001) for static graphs to variants of temporal betweenness based on types of optimal temporal paths for which the counting problem is not intractable. In this section, we provide the theoretical foundations for designing efficient algorithms to compute temporal betweenness centrality.

4.1 Counting shortest temporal paths

We first show how we can count shortest temporal paths efficiently. Our results from Section 3 indicate that this is not possible for the case of foremost and fastest temporal paths.

As it turns out, what we need to count is a slightly different version of shortest temporal paths, which we call *t-shortest* temporal paths. A temporal path P from s to z is a *t-shortest* temporal path if it arrives at time t and if there is no temporal path P' from s to z which arrives at time t and is shorter than P. We show that similarly to the static case, *t*-shortest temporal paths P have the property that every prefix is a t'-shortest temporal path, where t' is the time when the prefix arrives at its last vertex. Formally, we show the following.

Lemma 6. *Let* $P = \left(s \xrightarrow{t_1} \ldots \xrightarrow{t'} v \xrightarrow{t''} s' \xrightarrow{t'''} \ldots \xrightarrow{t} z\right)$ *be a t-shortest temporal path from s to z. Then* $P' = \left(s \xrightarrow{t_1} \ldots \xrightarrow{t'} v\right)$ *is a t'-shortest temporal path from s to v.*

Proof. Assume that $P = \left(s \xrightarrow{t_1} \ldots \xrightarrow{t} z\right)$ is a *t*-shortest temporal path but $P' = \left(s \xrightarrow{t_1} \ldots \xrightarrow{t'} v\right)$ is not a t'-shortest temporal path. Then there is a shorter temporal path $P'' = \left(s \xrightarrow{t_1'} \ldots \xrightarrow{t'} v\right)$ which implies that $\left(P'' \xrightarrow{t''} s' \xrightarrow{t'''} \ldots \xrightarrow{t} z\right)$ is shorter than P, a contradiction. $\qquad\square$

Lemma 6 allows us to formulate a recursive relation of the number of shortest temporal paths. To do this, we need to define the predecessor of a vertex appearance (v, t) on a temporal path.

Definition 8. *Let P be a temporal starting at s that contains the transitions* $w' \xrightarrow{t'} w \xrightarrow{t} v$. *Then* (w, t') *is the predecessor of (v, t) on P. Let P also contain the transition* $s \xrightarrow{t''} v'$. *Then* $(s, 0)$ *is the predecessor of* (v', t'').

Let $s \in V$ be any vertex and $(v, t) \in V \times [T]$ any vertex appearance. Then $P_s^{(\star)}(v, t)$ denotes the set of predecessors of (v, t) on $\star$-optimal temporal paths starting at s.

If we want to use the predecessor relation to formulate recursive function, then we need that the relation is acyclic; otherwise, we are not guaranteed to reach a base case. To show this formally, we define the directed *predecessor graph* for $\star$-optimal temporal paths starting at some vertex $s \in V$ as follows. 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 a $\star$-optimal temporal path starting at s 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 $(v, t) \in P_s^{(\star)}(w, t')$.

Definition 9 (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*

$$G_{\mathrm{pre}}^{(\star)}(s) := (V \times ([T] \cup \{0\}), E),$$

where

$$E := \left\{ ((v, t), (w, t')) \mid (v, t) \in P_s^{(\star)}(w, t') \right\}.$$

It is easy to observe that the predecessor graph is acyclic for all strict optimal temporal path versions, since the time stamps of the vertex appearances are strictly increasing along the arcs of the graph. In the case of t-shortest temporal paths, we show next that this also holds in the non-strict case.

Lemma 7. *Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. The predecessor graph $G_{\mathrm{pre}}^{(\mathrm{n}\text{-}t\text{-}\mathrm{sh})}(s)$ for non-strict t-shortest (n-t-sh) temporal paths is acyclic for all $s \in V$ and all $t \in T$.*

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 returns the length of the shortest path in the predecessor graph $G_{\mathrm{pre}}^{(\mathrm{n}\text{-}t\text{-}\mathrm{sh})}(s)$ from $(s, 0)$ to any vertex appearance. Note that for any vertex appearance (v, t') we have that $\mathrm{dist}(v, t')$ is also the length of a t'-shortest temporal path from s to v. Hence, for any vertex appearances (v, t') and (w, t''), if the predecessor graph contains the arc $((v, t'), (w, t''))$, then $\mathrm{dist}(v, t') < \mathrm{dist}(w, t'')$. Assume that $G_{\mathrm{pre}}^{(\mathrm{n}\text{-}t\text{-}\mathrm{sh})}(s)$ contains a cycle $C = ((v, t'), \ldots, (v, t'))$. Then $\mathrm{dist}(v, t') < \mathrm{dist}(v, t')$, a contradiction. □

With the help of this notation and Lemma 6 (and Lemma 7 for the non-strict case), we can formulate the following recursion for the number of t-shortest temporal paths.

Corollary 8 (Counting of t-Shortest Temporal Paths). *Let s be a source and let z be a vertex with $s \neq z$. The number of t-shortest temporal paths (t-sh) from s to z is given by:*

$$\sigma_{sz}^{(t-\mathrm{sh})} = \sum_{(v,t') \in P_s^{t-\mathrm{sh}}(z,t)} \sigma_{sv}^{(t'-\mathrm{sh})}.$$

By an analogous argument as in the proof of Lemma 7, we get the following, which we need in the next subsection.

Lemma 9. *Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. The predecessor graph $G_{\mathrm{pre}}^{(\mathrm{n}\text{-}\mathrm{sh})}(s)$ for non-strict shortest temporal paths is acyclic for all $s \in V$.*

4.2 Temporal dependencies

Similar to Brandes (2001), we introduce so-called *dependencies* to obtain an alternative formulation for the betweenness which we then use for the actual computation. Recall the definition of (static) betweenness centrality

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

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 (2001) calls the latter the *pair-dependency* of s and z on v. Brandes' central observation is 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 *dependency* of s on v. He showed that the dependencies obey a recursive relation which can be exploited algorithmically.

Both concepts, dependencies and pair-dependencies, naturally generalize to optimal paths in temporal graphs. 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 *temporal dependencies* of vertex appearances:

Definition 10 (Temporal Dependency). *Let s be a source and let z be a target vertex. 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:*

$$\delta_{sz}^{(\star)}(v, t) := \begin{cases} 0, & \text{if } \sigma_{sz}^{(\star)} = 0 \\ \dfrac{\sigma_{sz}^{(\star)}(v, t)}{\sigma_{sz}^{(\star)}}, & \text{otherwise,} \end{cases}$$

$$\delta_{s\bullet}^{(\star)}(v, t) := \sum_{z \in V} \delta_{sz}^{(\star)}(v, t).$$

From our definition of the modified temporal betweenness of a vertex appearance, it follows that we can use the temporal dependencies to compute it.

Observation 10. For any vertex appearance $(v, t) \in V \times [T]$ it holds:

$$\hat{C}_B^{(\star)}(v, t) = \sum_{s \in V} \delta_{s\bullet}^{(\star)}(v, t).$$

Now we state our central result allowing us to design algorithms to compute the temporal betweenness centrality based on shortest temporal paths.

Lemma 11 (Temporal Dependency Accumulation). *Fix a source $s \in V$. For any vertex appearance $(v, t) \in V \times [T]$ it holds:*

$$\delta_{s\bullet}^{(sh)}(v, t) = \delta_{sv}^{(sh)}(v, t) + \sum_{(w,t'):(v,t) \in P_s^{sh}(w,t')} \frac{\sigma_{sv}^{(t-sh)}}{\sigma_{sw}^{(t'-sh)}} \cdot \delta_{s\bullet}^{(sh)}(w, t').$$

Proof. We start by rewriting the pair-dependency of a vertex appearance as the summed fractions of shortest temporal paths using a specific transition from v to some vertex w at some time step:

$$\delta_{s\bullet}^{(sh)}(v, t) = \sum_{z \in V} \delta_{sz}^{(sh)}(v, t)$$

$$= \delta_{sv}^{(sh)}(v, t) + \sum_{z \in V} \sum_{(w,t'):(v,t) \in P_s^{sh}(w,t')} \delta_{sz}^{(sh)}((v, t), (\{v, w\}, t')),$$

where $\delta_{sz}^{(\mathrm{sh})}((v,t),(\{v,w\},t'))$ is the fraction of shortest temporal paths from vertex s to z that use the transitions $v' \xrightarrow{t} v \xrightarrow{t'} w$ for some v'. Note that for the case that $z = v$, there is no vertex appearance (w,t') such that $(v,t) \in P_s^{\mathrm{sh}}(w,t')$; hence, we need $\delta_{sv}^{(\mathrm{sh})}(v,t)$ as an additional summand. Furthermore, we have that

$$\delta_{sz}^{(\mathrm{sh})}((v,t),(\{v,w\},t')) = \frac{\sigma_{sv}^{(t-\mathrm{sh})}}{\sigma_{sw}^{(t'-\mathrm{sh})}} \cdot \frac{\sigma_{sz}^{(\mathrm{sh})}(w,t')}{\sigma_{sz}^{(\mathrm{sh})}}.$$

Inserting these cases into the double sum above yields the following:

$$\sum_{z \in V} \sum_{(w,t'):(v,t) \in P_s^{\mathrm{sh}}(w,t')} \delta_{sz}^{(\mathrm{sh})}((v,t),(\{v,w\},t'))$$

$$= \sum_{(w,t'):(v,t) \in P_s^{\mathrm{sh}}(w,t')} \sum_{z \in V} \delta_{sz}^{(\mathrm{sh})}((v,t),(\{v,w\},t'))$$

$$= \sum_{(w,t'):(v,t) \in P_s^{\mathrm{sh}}(w,t')} \left(\frac{\sigma_{sv}^{(t-\mathrm{sh})}}{\sigma_{sw}^{(t'-\mathrm{sh})}} \cdot \sum_{z \in V} \frac{\sigma_{sz}^{(\mathrm{sh})}(w,t')}{\sigma_{sz}^{(\mathrm{sh})}} \right)$$

$$= \sum_{(w,t'):(v,t) \in P_s^{\mathrm{sh}}(w,t')} \frac{\sigma_{sv}^{(t-\mathrm{sh})}}{\sigma_{sw}^{(t'-\mathrm{sh})}} \cdot \delta_{s\bullet}^{(\mathrm{sh})}(w,t').$$

The lemma statement now follows immediately. □

Lemma 1 together with Observation 10 show that we can use temporal dependencies to compute temporal betweenness centrality. Lemma 11 shows that the temporal dependencies obey a recursive relation, and Lemma 9 shows that it is acyclic. This together with Corollary 8 allows us to design a polynomial-time algorithm to compute temporal betweenness centrality based on shortest temporal paths. Before we give the specific algorithm description and running time analysis, we describe some other temporal betweenness concepts that can be computed in a similar fashion.

4.3 Specialized optimality concepts

So far, our results allow us to efficiently compute the temporal betweenness for strict and non-strict *shortest* temporal paths. For all other of the six canonical variants, we have shown computational hardness results in Section 3. In the following, we show that for some specialized optimality concepts for temporal paths, we can also efficiently compute the corresponding temporal betweenness. Specifically, we consider *shortest foremost* temporal paths and *prefix-foremost* temporal paths.

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

Definition 11. *Let $s, z \in V$. A temporal path P from s to z is a shortest foremost temporal path if P is a foremost temporal path from s to z and for all foremost temporal paths P' from s to z it holds: $|P| \leq |P'|$.*

We observe that a shortest foremost temporal path is a *t*-shortest temporal path for the earliest possible arrival time. Hence, our findings in the previous subsection translate very easily to shortest foremost temporal paths.

Prefix-foremost temporal paths. Now we consider the class of foremost temporal paths for which every prefix of the temporal path is foremost as well. That is, every vertex that is visited by such a temporal path is visited as soon as possible. These temporal paths are called *prefix-foremost paths*.

Definition 12. *Let* $s, z \in V$. *A path* $P = (e_1, \ldots, e_k)$ *from s to z is a prefix-foremost temporal path if P is a foremost path and every prefix* $P' = (e_1, \ldots, e_{k'})$, $k' \leq k$ *is a foremost temporal path as well.*

Wu et al. (2016) showed 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. We can observe that in the straightforward reduction from Paths (Valiant, 1979) that yields Proposition 2 we also get that counting non-strict prefix-foremost temporal paths is computationally hard.

Proposition 12. PREFIX-FOREMOST PATHS *is #P-hard.*

The main difference between strict and *non-strict* prefix-foremost paths is that in the former case, the predecessor relation is acyclic, and in the latter it is not. Intuitively, this is the reason why we get hardness in the non-strict case and tractability for the strict version.

For strict prefix-foremost paths, one can show that the counting problem and the computation of the corresponding betweenness are solvable in polynomial time by very similar arguments as for shortest temporal paths. The main difference is that the time stamp of a predecessor is unique; hence, we can define the set of predecessors as a set of vertices as opposed to a set of vertex appearances; otherwise, the arguments are analogous to the ones used in the proof of Lemma 11. This allows us to simplify the recursive function for the temporal dependency for prefix-foremost temporal paths which also results in a faster computation.

Lemma 13 (Strict Prefix-Foremost Dependency Accumulation). *Fix a source* $s \in V$. *For any vertex* $v \in V$, $v \neq s$ *it holds:*

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

Proof. We start by rewriting the pair-dependency of a vertex appearance as the summed fractions of prefix-foremost temporal paths using a specific transition from *v* to some vertex *w* at some time step:

$$\delta_{s\bullet}^{(\mathrm{pfm})}(v) = \sum_{z \in V} \delta_{sz}^{(\mathrm{pfm})}(v)$$

$$= \delta_{sv}^{(\mathrm{pfm})}(v) + \sum_{z \in V} \sum_{w : v \in P_s^{(\mathrm{pfm})}(w)} \delta_{sz}^{(\mathrm{pfm})}(v, \{v, w\}),$$

where $\delta_{sz}^{(\mathrm{pfm})}(v, \{v, w\})$ is the fraction of prefix-foremost temporal paths from vertex *s* to *z* that use the transitions $v \xrightarrow{t} w$ for some *t*. Note that for the case that $z = v$ there is no vertex *w* such that $v \in P_s^{(\mathrm{pfm})}(w)$, hence we need $\delta_{sv}^{(\mathrm{pfm})}(v)$ as an additional summand. Furthermore, we have that

$$\delta_{sz}^{(\mathrm{pfm})}(v, \{v, w\}) = \frac{\sigma_{sv}^{(\mathrm{pfm})}}{\sigma_{sw}^{(\mathrm{pfm})}} \cdot \frac{\sigma_{sz}^{(\mathrm{pfm})}(w)}{\sigma_{sz}^{(\mathrm{pfm})}}.$$

Inserting these cases into the double sum above yields the following:

$$\sum_{z \in V} \sum_{w:v \in P_s^{(\text{pfm})}(w)} \delta_{sz}^{(\text{pfm})}(v, \{v, w\})$$

$$= \sum_{w:v \in P_s^{(\text{pfm})}(w)} \sum_{z \in V} \delta_{sz}^{(\text{pfm})}(v, \{v, w\})$$

$$= \sum_{w:v \in P_s^{(\text{pfm})}(w)} \left(\frac{\sigma_{sv}^{(\text{pfm})}}{\sigma_{sw}^{(\text{pfm})}} \cdot \sum_{z \in V} \frac{\sigma_{sz}^{(\text{pfm})}(w)}{\sigma_{sz}^{(\text{pfm})}} \right)$$

$$= \sum_{w:v \in P_s^{(\text{pfm})}(w)} \frac{\sigma_{sv}^{(\text{pfm})}}{\sigma_{sw}^{(\text{pfm})}} \cdot \delta_{s\bullet}^{(\text{pfm})}(w).$$

The lemma statement now follows immediately. □

5. Algorithms and running time analysis

In the following, we give detailed descriptions of the algorithms we use to compute the temporal betweenness variants discussed in Section 4. We first describe how to adapt Brandes' algorithm (Brandes, 2001) to the temporal setting and then describe the competing approach of computing temporal betweenness using so-called *static expansions*.

5.1 Algorithms based on adaptations of Brandes' algorithm to the temporal setting

Brandes' algorithm (Brandes, 2001) (for static graphs without edge weights) 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 source s on all other vertices v is added to the betweenness score of s.

Our algorithms for temporal betweenness will follow roughly the same structure. Instead of breadth-first search, the appropriate algorithm for the respective temporal path class is used as the base for the algorithm. To describe our algorithm formally, we need the notion of temporal neighborhoods.

Definition 13 (Temporal Neighborhood). *For a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ and a vertex $v \in V(\mathcal{G})$, we call $N(v) := \{(u, t) \in V(\mathcal{G}) \times [T] \mid \{u, v\} \in E_t(\mathcal{G})\}$ the temporal neighborhood of v.*

Shortest (foremost) temporal betweenness. We have shown that the temporal dependencies for shortest (foremost) paths follow a similar recursion as for static graphs.

Algorithm 1 uses a $|V| \times T$-table to store the number of shortest temporal paths to all vertex appearances. The overall structure of the algorithm is similar to Brandes' algorithm (Brandes, 2001)—a single-source-all-shortest-paths traversal from each vertex to all vertex appearances is performed and the count of shortest temporal 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.

In order to count shortest foremost temporal paths instead of shortest temporal 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.

Proposition 14. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, Algorithm 1 computes the shortest temporal betweenness and the shortest foremost temporal betweenness of all $v \in V$ in $O(|V| \cdot |\mathcal{E}| \cdot T)$ time and $O(|V| \cdot T + |\mathcal{E}|)$ space.*

Algorithm 1 Shortest (foremost) betweenness in temporal graphs

---

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

1:    **for** $s \in V$ **do**
2:      **for** $v \in V$ **do**   ▷ Initialization
3:        $\text{dist}[v] \leftarrow -1; \sigma[v] \leftarrow 0; t_{\min}[v] \leftarrow -1$
4:      **for** $(\{u, v\}, t) \in \mathcal{E}$ **do**
5:        $\delta^{\text{sh}}[v, t] \leftarrow 0; \delta^{\text{fm}}[v, t] \leftarrow 0$
6:        $\sigma[v, t] \leftarrow 0; P[v, t] \leftarrow \emptyset; \text{dist}[v, t] \leftarrow -1$
7:      $\text{dist}[s] \leftarrow 0; \text{dist}[s, 0] \leftarrow 0; t_{\min}[s] \leftarrow 0$
8:      $\sigma[s] \leftarrow 1; \sigma[s, 0] \leftarrow 1$
9:      $S \leftarrow$ empty stack; $Q \leftarrow$ empty queue; $Q \leftarrow$ enqueue$(s, 0)$
10:     **while** Q not empty **do**
11:       $(v, t) \leftarrow$ dequeue(Q)
12:       **for** $(w, t') \in N(v)$ with $t < t'$ **do**   ▷ $(t \leq t')$ for n-str
13:         **if** $\text{dist}[w, t'] = -1$ **then**   ▷ First arrival at (w, t')
14:           $\text{dist}[w, t'] \leftarrow \text{dist}[v, t] + 1$
15:           **if** $\text{dist}[w] = -1$ **then**   ▷ Shortest path to w
16:             $\text{dist}[w] \leftarrow \text{dist}[v, t] + 1$
17:           $S \leftarrow$ push(w, t'); Q.enqueue(w, t')
18:         **if** $\text{dist}[w, t'] = \text{dist}[v, t] + 1$ **then**   ▷ Shortest path to (w, t') via (v, t)
19:           $\sigma[w, t'] \leftarrow \sigma[w, t'] + \sigma[v, t]$
20:           $P[w, t'] \leftarrow P[w, t'] \cup \{(v, t)\}$
21:           **if** $\text{dist}[w, t'] = \text{dist}[w]$ **then**   ▷ Shortest path to w via (v, t)
22:             $\sigma[w] \leftarrow \sigma[w] + \sigma[v, t]$
23:           **if** $t_{\min}[w] = -1$ **or** $t' < t_{\min}[w]$ **then**   ▷ Foremost shortest path to w
24:             $t_{\min}[w] \leftarrow t'$
25:      $C_B^{(\text{sh})}[s] \leftarrow C_B^{(\text{sh})}[s] - |\{i \mid \text{dist}[i] \geq 0\}| + 1$
26:      $C_B^{(\text{fm})}[s] \leftarrow C_B^{(\text{fm})}[s] - |\{i \mid \text{dist}[i] \geq 0\}| + 1$
27:      **while** $(w, t') \leftarrow$ pop(S) **do**   ▷ Vertex appearances in order of
                                                           non-increasing distance from s
28:        **if** $\text{dist}[w, t'] = \text{dist}[w]$ **then**   ▷ Shortest path to w
29:          $\delta^{\text{sh}}[w, t'] \leftarrow \delta^{\text{sh}}[w, t'] + \frac{\sigma[w, t']}{\sigma[w]}$
30:        **if** $t' = t_{\min}[w]$ **then**   ▷ Foremost shortest path to w
31:          $\delta^{\text{fm}}[w, t'] \leftarrow \delta^{\text{fm}}[w, t'] + 1$
32:        **for** $(v, t) \in P[w, t']$ **do**
33:          $\delta^{\text{sh}}[v, t] \leftarrow \delta^{\text{sh}}[v, t] + \frac{\sigma[v, t]}{\sigma[w, t']} \cdot \delta^{\text{sh}}[w, t']$
34:          $C_B^{(\text{sh})}[v] \leftarrow C_B^{(\text{sh})}[v] + \frac{\sigma[v, t]}{\sigma[w, t']} \cdot \delta^{\text{sh}}[w, t']$
35:          $\delta^{\text{fm}}[v, t] \leftarrow \delta^{\text{fm}}[v, t] + \frac{\sigma[v, t]}{\sigma[w, t']} \cdot \delta^{\text{fm}}[w, t']$
36:          $C_B^{(\text{fm})}[v] \leftarrow C_B^{(\text{fm})}[v] + \frac{\sigma[v, t]}{\sigma[w, t']} \cdot \delta^{\text{fm}}[w, t']$
37:    **return** $C_B^{(\text{sh})}, C_B^{(\text{fm})}$

Proof. The correctness of Algorithm 1 follows from Lemma 1 together with Observation 10, which show that we can use temporal dependencies to compute the temporal betweenness, and from Corollary 8 and Lemma 11 (with Lemma 9) since our algorithm dynamically computes the values given by the recursive formulas stated in the lemmas.

More precisely, the algorithm follows the idea of a classical breadth-first-search algorithm. Starting at a source vertex s (Line 12), we queue vertex appearances that are reachable by traversing one additional time edge from a vertex appearance that is already reachable (loop in Line 10, new vertex appearances are queued in Line 17). Whenever a new t'-shortest temporal path from s to a vertex appearance (w, t') is discovered (Line 18), we update the t'-shortest temporal path count in Line 19 according to Corollary 8 and update the set of predecessors in Line 20. If the discovered temporal path is a shortest path from s to w, we update shortest temporal path count in Line 22. Furthermore, we save all vertex appearances in non-increasing distance from s in a stack S.

Next, we compute the temporal dependencies according to Lemma 11 in the loop starting in Line 27. By Lemma 9 we know that the predecessor relation is acyclic and hence we have that the temporal dependencies of the predecessors of a vertex appearance are already correctly computed once that vertex appearance is popped off the stack S. Using the temporal dependencies, we compute the modified temporal betweenness according to Observation 10.

Finally, we upper-bound the running time of Algorithm 1. The number of executions of the outer for-loop in Line 1 is clearly in $O(|V|)$. Now we look at the while-loop in Line 10 and the for-loop in Line 12 that is inside the while-loop.

Consider the for-loop in Line 12: Note that for every time edge $(\{v, w\}, t')$ in $\mathcal{E}$ there are $O(T)$ vertex appearances (v, t) such that $(w, t') \in N(v)$. Furthermore, for every time edge in $\mathcal{E}$ at most two elements are added to Q (see while-loop in Line 10). It follows that every time edge in $\mathcal{E}$ is touched $O(T)$ times during the execution of the while-loop in Line 10, hence the overall running time is in $O(|V| \cdot |\mathcal{E}| \cdot T)$.

The space consumption of Algorithm 1 is in $O(|V| \cdot T + |\mathcal{E}|)$, since we need to store matrices of size $|V| \cdot T$ for the temporal path counts and the temporal dependencies. $\square$

Strict prefix-foremost temporal betweenness. Algorithm 2 modifies Brandes' algorithm (Brandes, 2001) to count strict prefix-foremost temporal paths. The overall structure of the algorithm remains the same. Instead of iterating over all neighbors, however, we 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 temporal paths.

Notably, since the time labels of predecessors are unique in prefix-foremost temporal paths, we do not have to consider vertex appearances in this case. Intuitively, this is the main reason why we can achieve both a faster running time a lower space consumption.

Proposition 15. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, Algorithm 2 computes the prefix-foremost temporal betweenness of all $v \in V$ in $O(|V| \cdot |\mathcal{E}| \cdot \log|\mathcal{E}|)$ time and $O(|V| + |\mathcal{E}|)$ space.*

Proof. The proof is essentially analogous to the correctness proof for Algorithm 1 but uses Lemma 13 instead of Lemma 11.

It is easy to check that the running time of Algorithm 2 is upper-bounded by $|V|$ (the outer loop in Line 1) times the size of Q (the loop from Line 9) times the time necessary to dequeue (Line 10). The size of Q is in $O(|\mathcal{E}|)$ and the time to dequeue is in $O(\log|\mathcal{E}|)$. Here, we assume that $|V| \le |\mathcal{E}|$. The claimed running time follows.

The space consumption of Algorithm 2 is in $O(|V| + |\mathcal{E}|)$ the arrays for the temporal path counts and the temporal dependencies used in the algorithm are all of size $|V|$. $\square$

Algorithm 2 Strict prefix-foremost betweenness

Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$.

Output: Betweenness $C_B^{(\text{pfm})}$ of all vertices $v \in V(\mathcal{G})$

1: **for** $s \in V$ **do**
2:      **for** $v \in V$ **do**   ▷ Initialization
3:          $P[v] \leftarrow \emptyset; t_{\min}[v] \leftarrow -1$
4:          $\sigma[v] \leftarrow 0; \delta[v] \leftarrow 1$
5:      $t_{\min}[s] \leftarrow 0; \sigma[s] \leftarrow 1$
6:      $S \leftarrow$ empty stack
7:      $Q \leftarrow$ empty priority queue   ▷ Prioritized by time label
8:      $Q \leftarrow \text{enqueueAll}(\{s \xrightarrow{t} v \mid s \underline{t} v \in \mathcal{E}\})$
9:      **while** Q not empty **do**
10:         $v \xrightarrow{t} w \leftarrow \text{dequeue}(Q)$
11:         **if** $t_{\min}[w] = -1$ **then**   ▷ First and foremost arrival in w
12:             $t_{\min}[w] \leftarrow t$
13:             $S \leftarrow \text{push}(w)$
14:             $Q.\text{enqueueAll}\big(\{w \xrightarrow{t'} x \mid w \underline{t'} x \in \mathcal{E}, t < t'\}\big)$
15:         **if** $t_{\min}[w] = t$ **then**   ▷ Prefix-foremost path to w via v
16:             $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
17:             $P[w] \leftarrow P[w] \cup \{v\}$
18:     $C_B^{(\text{pfm})}[s] \leftarrow C_B^{(\text{pfm})}[s] - |\{i \mid \text{dist}[i] \geq 0\}| + 1$
19:     **while** $w \leftarrow \text{pop}(S)$ **do**   ▷ Vertices in order of non-increasing
                                                              distance from s
20:         **for** $v \in P[w]$ **do**
21:             $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot \delta[w]$
22:             $C_B^{(\text{pfm})}[v] \leftarrow C_B^{(\text{pfm})}[v] + \frac{\sigma[v]}{\sigma[w]} \cdot \delta[w]$
23: **return** $C_B^{(\text{pfm})}$

5.2 Algorithms based on the static expansion

In this section, we discuss the presumably more straightforward and "out-of-the-box" way of computing temporal betweenness, namely through the use of so-called *static expansions* or time-expanded temporal graphs, which are a key tool in temporal graph algorithmics (Zschoche et al., 2020; Akrida et al., 2019; Kempe et al., 2002; Mertzios et al., 2019; Wu et al., 2016). Static expansions are directed graphs that have a vertex for every vertex *appearance* of the corresponding

temporal graph and they preserve the connectivity between vertex appearances of the temporal graph.

The main idea is to (naïvely) use Brandes' algorithm (Brandes, 2001) for (directed) static graphs on the static expansion. Of course, we need to make sure that each shortest path in the static expansion corresponds to exactly one optimal temporal path in the temporal graph for the optimality criterion that we are interested in. To do this, it is necessary to exclude some paths from the betweenness computation. To this end, we first give a generalized version of static betweenness. Then we present the details on how to construct static expansions for the optimality concepts used in our work, which are slightly tailored versions of the static expansions used in the literature (Zschoche et al., 2020; Akrida et al., 2019; Kempe et al., 2002; Mertzios et al., 2019; Wu et al., 2016). Notably, we do not present a static expansion for the strict prefix-foremost case, since for this case our algorithm is significantly faster than for the other optimality concepts.

Let $S \subseteq V$ be a set of "source" vertices and $Z: S \to 2^V$ be a function that, for any $s \in S$, returns the set of "terminal" vertices for that source. Now we define the *S-Z-Betweenness* of a vertex $v \in V$ to be

$$C_B^{(SZ)}(v) = \sum_{s \in S \setminus \{v\}} \sum_{z \in Z(s) \setminus \{v\}} \delta_{sz}(v).$$

The problem of computing the *S-Z*-betweenness for each vertex in an arbitrary (weighted) directed graph can be solved with a straightforwardly generalized version Brandes' algorithm (Brandes, 2001). Informally speaking, only vertices in S are considered as sources in the outermost loop of the algorithm. Furthermore, when the dependencies are accumulated, only shortest paths from the current source to vertices in Z are considered. Formally, we have the following.

Observation 16. *Given a directed unweighted graph $G = (V, E)$ with a set of sources $S \subseteq V$ and a terminal function $Z: S \to 2^V$, the S-Z-betweenness of all vertices in V can be computed in $O(|S| \cdot (|V| + |E|))$ time and $O(|V| + |E|)$ space*[4].

Static expansion for shortest temporal betweenness. We now describe how to construct a static expansion of a temporal graph in order to compute (strict) shortest temporal betweenness. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. We then define the static expansion graph for the (strict) shortest betweenness as the directed unweighted graph $G = (\tilde{V}, \tilde{E})$, with the following sets of vertices and edges:

For each vertex $v \in V$ of the original graph, we define $T + 2$ vertices in the static expansion—one for each appearance and two special vertices: a "source" and a "sink." Formally,

$$\tilde{V} := \bigcup_{v \in V} \{v_t \mid t \in [T+1] \cup \{0\}\}.$$

For the edges, we have three cases. The vertices v_{T+1} are used as sentinel "terminal" vertices that will signify the end of a path in our betweenness computation. Hence, they do not have any outgoing edges. Now, for any temporal edge $(\{v, w\}, t) \in \mathcal{E}$ and every $v_{t'}$ with $t' \leq t$, we either add the directed edge $(v_{t'}, w_t)$, or the edge $(v_{t'}, w_{t+1})$—the former in the non-strict case, the latter in the strict case. Moreover, we also add an edge to the corresponding terminal vertex, that is, we add the edge $(v_{t'}, w_{T+1})$. Naturally, we also go through the same process with w as we did with v. Formally, we have in the non-strict case:

$$\tilde{E} := \bigcup_{(\{v,w\},t) \in \mathcal{E}} \bigcup_{0 \leq t' \leq t} \{(v_{t'}, w_t), (v_{t'}, w_{T+1}), (w_{t'}, v_t), (w_{t'}, v_{T+1})\},$$

and in the strict case:

$$\tilde{E} := \bigcup_{(\{v,w\},t) \in \mathcal{E}} \bigcup_{0 \leq t' \leq t} \{(v_{t'}, w_{t+1}), (v_{t'}, w_{T+1}), (w_{t'}, v_{t+1}), (w_{t'}, v_{T+1})\}.$$

Now, to use the static expansion to compute the betweenness values in the temporal graph, we first compute the *S-Z*-betweenness on the static expansion with

$$S := \{v_0 \mid v \in V\},$$
$$Z(s) := \{v_{T+1} \mid v \in V \setminus \{s\}\}.$$

We can now compute the temporal betweenness values using the following (where $\star$ is the corresponding optimality concept):

$$C_B^{(\star)}(v) = \sum_{t \in [T]} C_B^{(SZ)}(v_t).$$

Static expansion for shortest foremost temporal betweenness. This variant of static expansion will be similar to the static expansion from the previous paragraph. Indeed, for a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ we define the static expansion for (strict) shortest foremost betweenness to be the directed weighted graph $G = (\tilde{V}, \tilde{E}, \omega)$, with $\tilde{V}$ and $\tilde{E}$ defined exactly as in the static expansion for (strict) shortest betweenness (see previous paragraph). Additionally, we use the edge weight function ω to "simulate" the foremost aspect of the paths. We let all "internal" edges be of equal weight, but let the "terminal" edges have a weight related to their timestamps. We set the weight in such a way as to make it so that any path to an "appearance" v_t is better than any path to an "appearance" $v_{t'}$ for $t < t'$. Formally, we have

$$\omega(v_t, w_{t'}) = \begin{cases} 1, & \text{if } t' \leq T, \\ (n+1) \cdot (t'+1), & \text{otherwise.} \end{cases}$$

Note that this weight function works in both the strict and the non-strict case.

The further procedure, that is, the computation of *S-Z*-betweenness and then the temporal betweenness values can be done in exactly the same way as in for shortest temporal betweenness (see the previous paragraph).

We can observe that $|\tilde{V}| \in O(|V| \cdot T)$, $|\tilde{E}| \in O(|\mathcal{E}| \cdot T)$, and $|S| \in O(|V|)$. Using Observation 16, we obtain the following running time and space bound.

Corollary 17. *The static-expansion-based algorithm to compute the shortest (foremost) temporal betweenness runs in $O(T \cdot |V| \cdot (|V| + |\mathcal{E}|))$ time and $O(T \cdot (|V| + |\mathcal{E}|))$ space.*

Comparing the theoretical running time bounds of Corollary 17 to the ones from Proposition 15, we can see that using static-expansion-based algorithms required essentially the same running time but more space. However, we remark that the main purpose is to compare the approach of working directly on the temporal graph data to an "naïve" or "out-of-the-box" use of Brandes' algorithm (Brandes, 2001). The static-expansion-based algorithm might be improved in terms of space consumption, which we leave for future research. In the next section, we investigate empirically how the two approaches compare in practice.

6. Experimental evaluation

In this section, we analyze the running time of our implementations of Algorithm 1 (computing strict and non-strict shortest and shortest foremost temporal betweenness) and Algorithm 2 (computing strict prefix-foremost temporal betweenness) as well as implementations of algorithms based on static expansions (computing strict and non-strict shortest and shortest foremost temporal betweenness, same as Algorithm 1) on several real-world temporal graphs and we investigate the differences between the five different betweenness concepts our algorithms can compute.

Table 2. Statistics for the data sets used in our experiments. The lifetime $T^\star$ of a graph is the difference between the largest and smallest time stamp on an edge in the graph. The resolution r indicates how often edges were measured. Note that the number of times steps is $T = T^\star/r$

Data set	# Vertices n	# Edges M	Res. r	Lifetime $T^\star$
highschool-2011	126	28,560	20	272,330
highschool-2012	180	45,047	20	729,500
highschool-2013	327	188,508	20	363,560
primary school	242	125,773	20	116,900
hospital-ward	75	32,424	20	347,500
infectious	10,972	415,912	20	6,946,340
hypertext	113	20,818	20	212,340
karlsruhe	1,870	461,661	1	123,837,267
Facebook-like	1,899	59,835	1	16,736,181

6.1 Setup and statistics

We implemented[5] Algorithms 1 and 2 and algorithms using the static expansions (see Subsection 5.2) in C++ and we compiled our source code with GCC 7.5.0. We carried out experiments on an Intel Xeon W-2125 computer with four cores clocked at 4.0 GHz and with 256 GB RAM, running Linux 4.15. We did not utilize the parallel-processing capabilities.

We used the following freely available data sets for temporal graphs: Physical-proximity networks[6] between high school students ("highschool-2011," "highschool-2012," "highschool-2013" Gemmetto et al., 2014; Stehlé et al., 2011; Fournet and Barrat, 2014), children and teachers in a primary school ("primary school" Stehlé et al., 2011), patients and health-care workers ("hospital-ward" Vanhems et al., 2013), attendees of the Infectious SocioPatterns event ("infectious" Isella et al., 2011), and conference attendees of ACM Hypertext 2009 ("hypertext" Isella et al., 2011), and an email communication networks of the Karlsruhe Institute of Technology (KIT) ("karlsruhe" Goerke, 2011),[7] and one social communication network ("Facebook-like" Opsahl and Panzarasa, 2009; Panzarasa et al., 2009).[8] We summarize some important statistics about the different data sets in Table 2.

6.2 Experimental results

We now discuss the results of our experiments. We analyzed the influence of the temporal betweenness type on the running time, the distribution of the temporal betweenness values, and the ranking of the ten vertices with the highest temporal betweenness values.

Running time. As indicated by our theoretical running time bounds (see Table 1), Algorithm 1 (computing shortest and foremost shortest betweenness) is several orders of magnitudes slower than Algorithm 2 (computing prefix-foremost betweenness). Algorithm 1 solved all instances except for (which was not solved within the timeout of five hours) within 45 minutes (keep in mind that Algorithm 1 computes shortest and foremost shortest temporal betweenness simultaneously). Algorithm 2 solved all instances except "karlsruhe" within 30 s. We show the specific running times in Table 3. It is noticeable that in small instances, the algorithms based on static expansions are roughly a factor of two faster. However, on large instances, the approach directly working on the temporal graph seems to be better, see for example "Facebook-like," where we are

Table 3. Running times in seconds of our implementation, where the last two columns correspond to the algorithms based on static expansions. From left to right: non-strict shortest and shortest foremost betweenness, strict shortest and shortest foremost betweenness, strict prefix-foremost betweenness, non-strict shortest and shortest foremost betweenness computed with static-expansion-based algorithms, strict shortest and shortest foremost betweenness computed with static-expansion-based algorithms. A -1 indicates that the instance was not solved within five hours

Data set	N-Str. Sh (Fm)	Str. Sh (Fm)	Str. P Fm	Ex N-Str. Sh (Fm)	Ex Str. Sh (Fm)
highschool-2011	6.59×10^1	6.56×10^1	7.93×10^{-1}	3.63×10^1	3.47×10^1
highschool-2012	1.97×10^2	1.97×10^2	2.11×10^0	1.24×10^2	1.19×10^2
highschool-2013	2.61×10^3	2.59×10^3	2.23×10^1	2.06×10^3	1.91×10^3
primary school	8.98×10^2	8.95×10^2	8.81×10^0	5.13×10^2	5.08×10^2
hospital-ward	9.75×10^1	9.72×10^1	4.33×10^{-1}	4.89×10^1	4.67×10^1
infectious	9.95×10^2	9.42×10^2	6.94×10^0	-1	-1
hypertext	3.71×10^1	3.69×10^1	4.62×10^{-1}	2.14×10^1	2.04×10^1
karlsruhe	-1	-1	$311, 727$	-1	-1
Facebook-like	1.3×10^3	1.3×10^3	2.49×10^1	8.25×10^3	8.22×10^3

faster by a factor of roughly six, and "infectious," where the algorithm based on static expansions did not finish within five hours. We believe that the static-expansion-based algorithm has higher memory requirements and more inefficient cache usage and hence becomes significantly slower for larger instances.

Impact of temporal betweenness type on the temporal betweenness distribution. In Fig. 5 we exemplarily show histograms of the temporal betweenness values of the "highschool-2013" data set, the "Facebook-like" data set, and the "primary school" data set. We can observe that the temporal betweenness centrality has a power-law-like distribution and the temporal betweenness type does not have a strong influence on the distribution in the "highschool-2013" data set and the "Facebook-like" data set (which is the case in most data sets). In the "primary school" data set, however, the distributions for non-strict shortest and strict shortest temporal betweenness are very similar but the other ones are quite different from each other.

Impact of temporal betweenness type on the vertex ranking. In Table 4, we present Kendall's tau correlation measure[9] (Kendall, 1938; Knight, 1966) for each pair of vertex rankings induced by the different temporal betweenness versions. For the temporal betweenness variants "shortest" and "shortest foremost," we can observe that their respective non-strict and strict variants produce very similar rankings. We can further see that the betweenness variants "non-strict shortest foremost," "strict shortest foremost," and "strict prefix foremost" are pairwise similar. Recall that strict prefix-foremost temporal betweenness was computed significantly faster by our algorithms, hence we can conclude that if foremost temporal paths are of interest, then the prefix-foremost temporal betweenness is much easier to compute and yields very similar results.

These findings are supported by the pairwise comparisons of the sets of the ten vertices with the largest temporal betweenness values, which are presented in Table 5. Notably, the "primary school" has quite different top ten vertex sets for all pairs of betweenness variants that do not fall into the above-mentioned pairings. We can also observe that "non-strict shortest" and "strict prefix foremost" produce the most dissimilar vertex rankings and top ten sets.
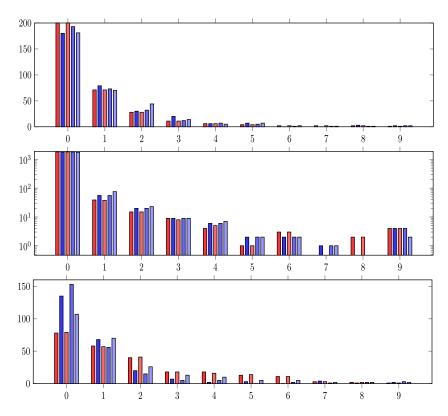
Figure 5. Top: temporal betweenness histogram of "highschool-2013" data set. Vertices are collected in 10 evenly distributed buckets between 0 and the highest temporal betweenness value. The *y*-axis corresponds to the number of vertices. Middle: temporal betweenness histogram of "Facebook-like" data set. The *y*-axis is on a log-scale. Bottom: temporal betweenness histogram of "primary school" data set. The temporal betweenness types from left to right are as follows: non-strict shortest, non-strict shortest foremost, strict shortest, strict shortest foremost, strict prefix foremost. Variants of shortest betweenness are colored in shades of red; foremost betweenness is colored in shades of blue.

7. Conclusion

We 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, one can count shortest and shortest foremost temporal paths in polynomial time both for strict and non-strict paths. In the case of prefix-foremost paths, however, we found a polynomial-time algorithm for the strict version, whereas the non-strict version is again #P-hard. An intuitive explanation for this behavior might be that our algorithms strongly rely on a recursive formulation for the so-called temporal dependencies, which in turn requires that the predecessor relation of optimal temporal paths is acyclic and that prefixes of optimal temporal paths are also optimal. For all optimal temporal path types for which we show computational hardness of the corresponding counting problem, one of the two mentioned requirements is not given.

As to challenges for future research, one direction is to attack the temporal betweenness centrality variants which we have shown to be computationally hard by means of approximation algorithms or the development of fixed-parameter algorithms (e.g., for some structural graph

Table 4. Kendall's tau correlation measure for each pair of vertex rankings induced by the different temporal betweenness versions. Values close to 1 indicate strong agreement, and values close to -1 indicate strong disagreement between the rankings

Data set	non-str. sh vs. non-str. sh fm	non-str. sh vs. str. sh	non-str. sh vs. str. sh fm	non-str. sh vs. str. p fm	non-str. sh fm vs. str. sh	non-str. sh fm vs. str. sh fm	non-str. sh fm vs. str. p fm	str. sh vs. str. sh fm	str. sh vs. str. p fm	str. sh fm vs. str. p fm
highschool-2011	0.51	1	0.54	0.48	0.51	0.87	0.81	0.54	0.48	0.84
highschool-2012	0.49	0.99	0.49	0.43	0.49	0.93	0.81	0.49	0.43	0.82
highschool-2013	0.43	1	0.43	0.39	0.43	0.89	0.75	0.43	0.39	0.78
primary school	0.26	0.99	0.29	0.18	0.26	0.76	0.58	0.28	0.17	0.63
hospital-ward	0.64	1	0.63	0.55	0.64	0.97	0.82	0.63	0.55	0.82
hypertext	0.54	1	0.55	0.51	0.54	0.89	0.78	0.55	0.51	0.85
Facebook-like	0.84	0.97	0.84	0.75	0.84	0.97	0.83	0.84	0.75	0.83

Table 5. Size of the intersection of each pair of sets containing the ten vertices with highest temporal betweenness values

Data set	non-str. sh vs. non-str. sh fm	non-str. sh vs. str. sh	non-str. sh vs. str. sh fm	non-str. sh vs. str. p fm	non-str. sh fm vs. str. sh	non-str. sh fm vs. str. sh fm	non-str. sh fm vs. str. p fm	str. sh vs. str. sh fm	str. sh vs. str. p fm	str. sh fm vs. str. p fm
highschool-2011	4	10	5	4	4	8	8	5	4	9
highschool-2012	4	10	4	3	4	9	7	4	3	8
highschool-2013	4	10	4	3	4	8	5	4	3	7
primary school	0	9	0	0	0	9	7	0	0	8
hospital-ward	7	10	7	7	7	9	8	7	7	9
hypertext	4	10	4	4	4	10	9	4	4	9
Facebook-like	9	10	9	6	9	10	7	9	6	7

parameters of the underlying graph). Roughly in the same direction would be to undertake a closer study of the special structures of real-world networks that might be algorithmically exploitable. A line of research directly motivated by our experiments would be to explore how to further decrease the memory consumption of our algorithms—the practical limitations seem to mainly come from the high space consumption, at some point leading our algorithms to do a lot of paging (for the static-expansion-based algorithm, this point seems to be reached much earlier). Specifically, our algorithm for shortest and shortest foremost temporal betweenness stores dependency and path count values for every vertex-time step combination. This is a major difference to the static algorithm of Brandes (Brandes, 2001) which is less vulnerable in this respect.

On the experimental side, we compared the running times of our approach to static-expansion-based temporal betweenness algorithms. We found that for small instances, the static-expansion-based temporal betweenness algorithms are faster but on large instances or algorithm performs better. We conjecture that this is the case because our algorithm has a better memory usage, hence the specific threshold of the input size where our algorithm is faster probably depends heavily on the machine that is used for the computation.

We can further observe that the temporal betweenness type does not have a strong impact on the distribution of the betweenness values. When comparing the vertex rankings produced by the betweenness values we can observe that there is little difference between strict and the respective non-strict variants. Furthermore, "shortest foremost" and "prefix foremost" yield similar results in terms of the betweenness values of the vertices, while the strict prefix-foremost temporal betweenness values can be computed significantly faster.

It would be very enlightening to compare the betweenness values of the two tractable variants of temporal betweenness based on foremost paths (shortest foremost and prefix foremost) to the betweenness values for temporal betweenness based on foremost temporal paths (which is intractable). However, that requires a way to compute or approximate the temporal betweenness values based on foremost temporal paths. Hence, developing efficient (parameterized exponential-time) algorithms or approximation algorithms for the temporal betweenness based on foremost temporal paths is also an interesting future work direction.

Data availability statement. The code of our implementation is available under GNU General Public license version 3 at http://fpt.akt.tu-berlin.de/software/temporal_betweenness/. The data sets used in the experimental evaluation are available at:

- http://www.sociopatterns.org/datasets/
- https://i11www.iti.kit.edu/en/projects/spp1307/emaildata
- https://snap.stanford.edu/data/CollegeMsg.html

Notes

1 According to Google Scholar, accessed April 2022, the paper is cited more than 5000 times.
2 Intuitively speaking, #P is the counting analogue of NP for decision problems.

3 A *polynomial-time counting reduction* transforms instances I_A of a counting problem A to instances I_B of a counting problem B such that number of solutions of I_A can be computed in polynomial time from the number of solutions of I_B.

4 *Here, we assume for each source we can check in constant time whether a given vertex is a terminal for this source.*

5 The code of our implementation is available under GNU General Public license version 3 at http://fpt.akt.tu-berlin.de/software/temporal_betweenness/.

6 Available at http://www.sociopatterns.org/datasets/.

7 Available at http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata.

8 Available at https://snap.stanford.edu/data/CollegeMsg.html.

9 The *Kendall's tau correlation measure* is a measure to compare rankings. It is defined as follows: (# concordant pairs – # discordant pairs)/# pairs.

References

Akrida EC, Czyzowicz J, Gasieniec L, Kuszner L and Spirakis PG (2019) Temporal flows in temporal networks. *Journal of Computer and System Sciences 103*, 46–60.

Alsayed A, & Higham DJ (2015) Betweenness in time dependent networks. *Chaos, Solitons & Fractals 72*, 35–48.

Bentert M, Himmel A-S, Molter H, Morik M, Niedermeier R and Saitenmacher R (2019) Listing all maximal *k*-plexes in temporal graphs. *ACM Journal of Experimental Algorithmics 24*(1), 13:1–13:27.

Bentert M, Himmel A-S, Nichterlein A and Niedermeier R (2020) Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science 5*(1), 73.

Brandes U (2001) A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology 25*(2), 163–177.

Bui-Xuan B-M, Ferreira A and Jarry A (2003) Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science 14*(02), 267–285.

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

Casteigts A, Flocchini P, Mans B and Santoro N (2015) Shortest, fastest, and foremost broadcast in dynamic networks. *International Journal of Foundations of Computer Science 26*(4), 499–522.

Casteigts A, Himmel A, Molter H and Zschoche P (2021) Finding temporal paths under waiting time constraints. *Algorithmica 83*(9), 2754–2802.

Daly EM and Haahr M (2007) 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*. ACM, pp. 32–40.

Enright JA, Meeks K and Molter H (2023) Counting temporal paths. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS'23)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, vol. *254*, pp. 30:1–30:19.

Fournet J and Barrat A (2014) Contact patterns among high school students. *PlOS ONE 9*(9), e107878.

Freeman LC (1977) A set of measures of centrality based on betweenness. *Sociometry 40*(1), 35–41.

Gemmetto V, Barrat A and Cattuto C (2014) Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases 14*(1), 695.

Goerke R (2011) Email network of KIT informatics. http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata.

Habiba CT and Berger-Wolf TY (2007) Betweenness centrality measure in dynamic networks. Technical Report 19, Department of Computer Science, University of Illinois at Chicago, Chicago. DIMACS Technical Report.

Himmel A-S, Molter H, Niedermeier R and Sorge M (2017) Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining 7*(1), 35:1–35:16.

Holme, P. (2015) Modern temporal network theory: a colloquium. *The European Physical Journal B 88*(9), 234.

Holme P and Saramäki J (2013) *Temporal Networks*. Springer.

Holme P and Saramäki J (2019) *Temporal Network Theory*. Springer.

Isella L, Stehlé J, Barrat A, Cattuto C, Pinton J-F and Van den Broeck W (2011) What's in a crowd? Analysis of face-to-face behavioral networks. *Journal of Theoretical Biology 271*(1), 166–180.

Kempe D, Kleinberg J and Kumar A (2002) Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences 64*(4), 820–842.

Kendall MG (1938) A new measure of rank correlation. *Biometrika 30*(1/2), 81–93.

Kim H and Anderson R (2012) Temporal node centrality in complex networks. *Physical Review E 85*(2), 026107.

Knight WR (1966) A computer method for calculating Kendall's tau with ungrouped data. *Journal of the American Statistical Association 61*(314), 436–439.

Latapy M, Viard T and Magnien C (2018) Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining 8*(1), 61.

Leydesdorff L (2007) Betweenness centrality as an indicator of the interdisciplinarity of scientific journals. *Journal of the American Society for Information Science and Technology 58*(9), 1303–1319.

Mertzios GB, Michail O and Spirakis PG (2019) Temporal network optimization subject to connectivity constraints. *Algorithmica 81*(4), 1416–1449.

Michail O (2016) An introduction to temporal graphs: an algorithmic perspective. *Internet Mathematics 12*(4), 239–280.

Molter H, Niedermeier R and Renken M (2020) Isolation concepts applied to temporal clique enumeration. *Network Science 9*(S1), 1–23.

Nicosia V, Tang J, Mascolo C, Musolesi M, Russo G and Latora V (2013) Graph metrics for temporal networks. In *Temporal Networks*, pp. 15–40. Springer.

Opsahl T and Panzarasa P (2009) Clustering in weighted networks. *Social Networks 31*(2), 155–163.

Panzarasa P, Opsahl T and Carley KM (2009) Patterns and dynamics of users' behavior and interaction: network analysis of an online community. *Journal of the American Society for Information Science and Technology 60*(5), 911–932.

Rad AA, Flocchini P and Gaudet J (2017) Computation and analysis of temporal betweenness in a knowledge mobilization network. *Computational Social Networks 4*(1), 5.

Rymar M, Molter H, Nicherlein A and Niedermeier R (2021) Towards classifying the polynomial-time solvability of temporal betweenness centrality. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'21)*, of *Lecture Notes in Computer Science*, vol. 12911. Springer, pp. 219–231.

Şimşek Ö and Barto AG (2009) Skill characterization based on betweenness. In *Proceedings of Advances in Neural Information Processing Systems 21*. Curran Associates, Inc, pp. 1497–1504.

Stehlé J, Voirin N, Barrat A, Cattuto C, Isella L, Pinton J-F, Quaggiotto M, Van den Broeck W, RégisC, Lina B, Vanhems P and Viboud C (2011) High-resolution measurements of face-to-face contact patterns in a primary school. *PlOS ONE 6*(8), e23176.

Tang J, Mascolo C, Musolesi M and Latora V (2011) Exploiting temporal complex network metrics in mobile malware containment. In *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1–9.

Tang J, Musolesi M, Mascolo C and Latora V (2009) Temporal distance metrics for social network analysis. In *Proceedings of the 2nd ACM Workshop on Online Social Networks*. ACM, pp. 31–36.

Tang J, Musolesi M, Mascolo C, Latora V and Nicosia V (2010) Analysing information flows and key mediators through temporal centrality metrics. In *Proceedings of the 3rd ACM Workshop on Social Network Systems*. ACM, pp. 3:1–3:6.

Tsalouchidou I, Baeza-Yates R, Bonchi F, Liao K and Sellis T (2020) Temporal betweenness centrality in dynamic graphs. *International Journal of Data Science and Analytics 9*(3), 257–272.

Valiant LG (1979) The complexity of enumeration and reliability problems. *SIAM Journal on Computing 8*(3), 410–421.

van den Heuvel MP, Mandl RC, Stam CJ, Kahn RS an dPol HEH (2010) Aberrant frontal and temporal complex network structure in schizophrenia: a graph theoretical analysis. *Journal of Neuroscience 30*(47), 15915–15926.

Vanhems P, Barrat A, Cattuto C, Pinton J-F, Khanafer N, Régis C and Voirin N (2013) Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PlOS ONE 8*(9), e73970.

Wu H, Cheng J, Ke Y, Huang S, Huang Y and Wu H (2016) Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering 28*(11), 2927–2942.

Zschoche P, Fluschnik T, Molter H and Niedermeier R (2020) The complexity of finding separators in temporal graphs. *Journal of Computer and System Sciences 107*, 72–92.