Computing Betweenness Centrality in Link Streams

Frédéric Simard [1] *Clémence Magnien* [1] *Matthieu Latapy* [1]

[1]Sorbonne University, CNRS, LIP6, F-75005 Paris, France

Abstract. Betweeness centrality is one of the most important concepts in graph analysis. It was recently extended to link streams, a graph generalization where links arrive over time. However, its computation raises non-trivial issues, due in particular to the fact that time is considered as continuous. We provide here the first algorithms to compute this generalized betweenness centrality, as well as several companion algorithms that have their own interest. They work in polynomial time and space, we illustrate them on typical examples, and we provide an implementation.

1 Introduction

Betweenness centrality, or betweenness for short, is one of the most classical and important concepts defined over graphs and used in the field of complex networks and social network analysis [42, 36, 23, 22, 10]. Given a graph $G = (V, E)$, it measures how frequently each node $v \in V$ is involved in shortest paths: $B(v) = \sum_{u \in V, w \in V} \frac{\sigma(u,w,v)}{\sigma(u,w)}$ where $\frac{\sigma(u,w,v)}{\sigma(u,w)}$ is the fraction of all shortest paths from u to w that involve v if there is a path from u to w, 0 otherwise. Reference algorithms compute the betweenness of all nodes in a graph in time $O(n \cdot m)$, where n and m are the number of nodes and links in the graph [4].

Betweenness was extended recently to link streams [21], a family of formal objects that model sequences of interactions over time in a way similar to the modeling of relations by graphs. They are equivalent to other objects like time-varying graphs (TVG) [8, 2], relational event models (REM) [7, 30], or temporal networks [24, 16], with an emphasis on the streaming nature of link sequences. Various temporal extensions of beweenness were introduced in these contexts, see Section 7.

We first introduce key concepts and notations in Section 2. Betweenness in link streams has some unique features that make it quite different from other temporal extensions of betweenness in graphs. In particular, it considers continuous time and links with or without durations: nodes may be linked at specific time instants, as well as during continuous periods of time. As a consequence,

---

E-mail addresses: Frederic.Simard@lip6.fr (Frédéric Simard) Clemence.Magnien@lip6.fr (Clémence Magnien)
Matthieu.Latapy@lip6.fr (Matthieu Latapy)

a set of paths from a temporal node to another one is in general uncountable, which requires specific concepts to estimate its importance; we define them in Section 3. Also, betweenness in link streams considers paths from any node at any time instant to any node at any time instant, which induces an uncountable amount of temporal nodes. Therefore, not all paths that involve a given temporal node necessarily have the same impact on its betweenness centrality: this impact depends on the amount of temporal nodes that are sources and destinations for each path. This raises specific algorithmic challenges, that we address in Sections 4 and 5. We finally obtain the first algorithm for computing betweenness centrality in link streams. We present this polynomial time and space algorithm, and show results on non-trivial toy examples in Section 6. We provide an open Python implementation of these algorithms [19].

2   Preliminaries

A link stream L is a triplet (T, V, E) where $T = [\alpha, \omega]$ is an interval of $\mathbb{R}$ representing time, V is a finite set of nodes, and $E \subseteq T \times V \otimes V$ is the set of links [1]. Then, $(t, uv) \in E$ means that u and v are linked together at time t. For any u and v in V, $T_{uv} = \{t, (t, uv) \in E\}$ denotes the set of time instants at which u and v are linked together. See Figure 1 for an illustration and [21] for a full presentation of the formalism.

We assume here that T_{uv} is the union of a finite number of disjoint closed intervals (possibly singletons) of T. We denote by $\mathcal{T}$ the set of bounds of maximal intervals in T_{uv} for any u and v, that we call event times. We denote by $\overline{m}_{uv}$ the number of maximal intervals in T_{uv}, and by $\overline{m} = \sum_{u,v \in V} \overline{m}_{uv}$ their sum, *i.e.* the number of maximal intervals in E. In the case of Figure 1, we obtain $\mathcal{T} = \{1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 18, 19, 22, 23, 24, 25, 27, 28, 29, 30, 31\}$, $\overline{m}_{ab} = 3$, $\overline{m}_{ac} = 1$, $\overline{m}_{bc} = 4$, $\overline{m}_{bd} = 1$, $\overline{m}_{cd} = 3$, $\overline{m}_{de} = 4$, and so $\overline{m} = 16$.

Given a link stream $L = (T, V, E)$ and a time t, we define the graph $G_t = (V, E_t)$ with $E_t = \{uv, (t, uv) \in E\}$. We denote by $N_t(v)$ the set of neighbors of v in G_t. We denote by $\sigma_t(u, v)$ the (finite) number of paths from u to v in G_t, and by $d_t(u, v)$ the distance from u to v in this graph.
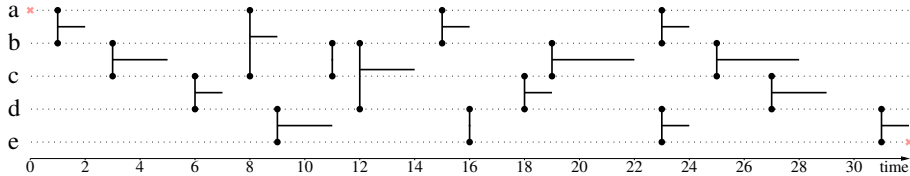

Figure 1: **An example of link stream** $L = (T, V, E)$ with $T = [\alpha, \omega] = [0, 32]$, $V = \{a, b, c, d, e\}$, and E defined by $T_{ab} = [1, 2] \cup [15, 16] \cup [23, 24]$, $T_{ac} = [8, 9]$, $T_{bc} = [3, 5] \cup \{11\} \cup [19, 22] \cup [25, 28]$, $T_{bd} = [12, 14]$, $T_{cd} = [6, 7] \cup [18, 19] \cup [27, 29]$, and $T_{de} = [9, 11] \cup \{16\} \cup [23, 24] \cup [31, 32]$.

In $L = (T, V, E)$, a path P from $(x, u) \in T \times V$ to $(y, v) \in T \times V$ is a sequence $v_0, t_1, v_1, t_2, v_2, \ldots t_k, v_k$ with $v_0 = u$, $v_k = v$, $x \le t_1 \le t_2 \le \cdots \le t_k \le y$, and $(t_i, v_{i-1}v_i) \in E$ for all i. If such a path exists, then (y, v) is reachable from (x, u), which we denote by $(x, u) \longrightarrow (y, v)$. The path P involves (t_1, u), (t_k, v), and (t, v_i) for all $t \in [t_i, t_{i+1}]$ and all i. It starts at t_1, arrives at t_k, has length k and duration $t_k - t_1$. A path with duration 0 is called an instantaneous path.

---

[1] We make the distinction between the set $X \times Y$ of ordered pairs of elements of X and Y, that we denote by (x, y) with $x \in X$ and $y \in Y$, and the set $X \otimes Y$ of unordered pairs of distinct elements of X and Y, that we denote by xy with $x \in X$, $y \in Y$ and $x \ne y$; $(x, y) \ne (y, x)$ while $xy = yx$.

For instance, in the case of Figure 1, the sequences $a, 1, b, 4, c, 6, d, 9, e$ and $a, 9, c, 18, d, 27, c, 28,$ $d, 31, e$ are two paths from $(0, a)$ to $(32, e)$ with length 4 and duration 8, and length 5 and duration 22, respectively.

The path P is a shortest path from (x, u) to (y, v) if it has minimal length, called the distance from (x, u) to (y, v) and denoted by $d((x, u), (y, v))$. The path P is a fastest path from (x, u) to (y, v) if it has minimal duration, called the latency from (x, u) to (y, v) and denoted by $\ell((x, u), (y, v))$. The path P is a shortest fastest path from (x, u) to (y, v) if it is a path of minimum length among those of minimal duration from (x, u) to (y, v).

For instance, in the case of Figure 1, the path $a, 2, b, 4, c, 6, d, 9, e$ is a fastest path from $(0, a)$ to $(32, e)$, but $a, 1, b, 4, c, 6, d, 9, e$ is not (it has duration 8). The path $a, 2, b, 4, c, 6, d, 9, e$ has length 4 and duration 7, and no path from $(0, a)$ to $(32, e)$ with lower duration exists. It is not a shortest path since $a, 9, c, 18, d, 23, e$ also is a path from $(0, a)$ to $(32, e)$ which has length 3 and duration 14. This last path is a shortest path, since no path with lower length exists, but not a fastest one. The distance from $(0, a)$ to $(32, e)$ therefore is 3 and the latency is 7. Among the fastest paths from $(0, a)$ to $(32, e)$, *i.e.* the paths of duration 7, the shortest have length 4. Therefore, $a, 2, b, 4, c, 6, d, 9, e$ is a shortest fastest path between them, as well as $a, 2, b, 5, c, 6, d, 9, e$, for instance.

Finally, the betweenness of a node $v \in V$ at a time instant $t \in T$ measures how frequently (t, v) is involved in shortest fastest paths in L, see [21]:

$$B(t, v) = \sum_{u \in V, w \in V} \int_{i \in T, j \in T} \frac{\sigma((i, u), (j, w), (t, v))}{\sigma((i, u), (j, w))} \, \mathrm{d}i \, \mathrm{d}j$$

where $\frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))}$ is the fraction of all shortest fastest paths from u at time i to w at time j that involve v at time t if there is a path from (i, u) to (j, w), 0 otherwise.

It is important to notice that path contributions depend on the amount of temporal node pairs for which they are shortest fastest paths. Consider for instance the shortest fastest paths from (i, a) to (j, c) in both link streams in Figure 2, for any i and j. There is only one such path in each case, namely $a, 1, b, 2, c$ for the first stream and $a, 5, b, 6, c$ for the second stream. But in the first link stream the path is a shortest fastest path for $i \in [0, 1]$ and $j \in [2, 10]$, while in the second the path is a shortest fastest path for $i \in [0, 5]$ and $j \in [6, 10]$. This has drastic influence on betweenness computation. Consider for instance the temporal nodes $(1.5, b)$ and $(5.5, b)$ in the first and second link stream respectively, highlighted with crosses in the figure. Although the paths have similar features regarding these two temporal nodes, they have significantly different contributions to their betweenness: $\int_{i \in T, j \in T} \frac{\sigma((i,u),(j,w),(1.5,b))}{\sigma((i,u),(j,w))} \, \mathrm{d}i \, \mathrm{d}j = (1 - 0) * (10 - 2) = 8$ and $\int_{i \in T, j \in T} \frac{\sigma((i,u),(j,w),(5.5,b))}{\sigma((i,u),(j,w))} \, \mathrm{d}i \, \mathrm{d}j = (5 - 0) * (10 - 6) = 20$, respectively. This reflect the fact that $(5.5, b)$ in the second link stream is involved in shortest fastest paths between more temporal node pairs than $(1.5, b)$ in the first stream.

In the original definition given above, the quantity $\frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))}$ is only loosely defined as a fraction of shortest fastest paths; the function σ itself, as well as the ratio between its values, are not explicitly defined. We will see in next section that this fraction involves uncountable sets of shortest fastest paths that have finite volumes with a size and a dimension. We will also introduce the appropriate arithmetic operators needed to deal with them, and an algorithm to compute these volumes.

Figure 2: Two link streams illustrating the fact the impact of a shortest fastest path depends on the amount of temporal node pairs for which it is a shortest fastest path. Left: $L = (T, V, E)$ with $T = [0, 10]$, $V = \{a, b, c\}$, and $E = \{(1, ab), (2, bc)\}$. The temporal node $(1.5, b)$ is marked with a cross. Right: $L = (T, V, E)$ with $T = [0, 10]$, $V = \{a, b, c\}$, and $E = \{(5, ab), (6, bc)\}$. The temporal node $(5.5, b)$ is marked with a cross.

3 Volumes of shortest paths

Let us consider a link stream $L = (T, V, E)$, and a sequence I_1, I_2, $\cdots$, I_k of intervals of T. Let us denote by b_i and e_i the bounds of interval I_i, with $b_i \leq e_i$. If $e_i = b_i$ then I_i is a singleton ($I_i = \{b_i\} = \{e_i\}$). The intervals may be closed ($I_i = [b_i, e_i]$), half-open ($I_i =]b_i, e_i]$ or $I_i = [b_i, e_i[$), or open ($I_i =]b_i, e_i[$).

We say that the sequence I_1, I_2, $\cdots$, I_k is a **sliding sequence** if for all i, there exists no element in I_{i+1} strictly smaller than all elements of I_i ($\nexists y \in I_{i+1}, \forall x \in I_i, y < x$), and no element of I_i strictly larger than all elements of I_{i+1} ($\nexists x \in I_i, \forall y \in I_{i+1}, x > y$).

In such a sequence, the intervals may overlap ($I_i \cap I_j \neq \emptyset$, $i \neq j$), may be included in each other ($I_i \subseteq I_j$, $i \neq j$), or may even be equal ($I_i = I_j$, $i \neq j$).

Given a sliding sequence I_1, I_2, $\cdots$, I_k, we denote by $v_0, I_1, v_1, I_2, v_2, \cdots I_k, v_k$ the set S of all sequences $v_0, t_1, v_1, t_2, v_2, \cdots, t_k, v_k$ such that $v_i \in V$, $t_i \in I_i$ and $t_{i+1} \geq t_i$ for all i. We say that S is a **sliding set**. If the intervals are disjoint then $S = \{v_0\} \times I_1 \times \{v_1\} \times I_2 \times \{v_2\} \times \cdots \times I_k \times \{v_k\}$, but this is not true in general.

In the case of Figure 1, for instance, $[23, 24],]25, 28], [27, 29], \{31\}$ is a sliding sequence and $a, [23, 24], b,]25, 28], c, [27, 29], d, \{31\}, e$ is a sliding set. The elements of this set are the paths $a, t_1, b, t_2, c, t_3, d, t_4, e$ with $23 \leq t_1 \leq 24$, $25 < t_2 \leq 28$, $\max(27, t_2) \leq t_3 \leq 29$, and $t_4 = 30$.

More generally, all paths in any link stream are elements of sliding sets. In the case of Figure 1, for instance, all shortest paths from $(0, a)$ to $(14, e)$ go from a to b between times 1 and 2, from b to c between times 3 and 5, from c to d between times 6 and 7, and finally from d to e between times 9 and 11. Therefore, they are elements of $a, [1, 2], b, [3, 5], c, [6, 7], d, [9, 11], e$.

In addition, if we consider any two elements (i, u) and (j, v) of $T \times V$, then we have the following result.

Proposition 1 *The set $SP((i, u), (j, v))$ of all shortest paths from (i, u) and (j, v) is the disjoint union of a finite number of sliding sets.*

Proof: Let us consider all sliding sequences I_1, I_2, $\cdots$, I_k with $k = d((i, u), (j, v))$ and I_i is either an open interval $]t, t'[$ such that t and t' are two consecutive event times, or I_i is a singleton $\{t\}$ such that t is an event time. Since there is a finite number of event times, there is a finite number of such sequences, and they induce a finite number of sliding sets which are all disjoint.

Any path in $SP((i, u), (j, v))$ is in one of these sliding sets, and then all the elements of this sliding set are shortest paths from (i, u) to (j, v). Therefore $SP((i, u), (j, v))$ is the union of such sliding sets. □

For instance, let us consider the following sliding sets:

$A = a, [1, 2], b, [3, 5], c, [6, 7], d, [9, 11], e$; $B = a, [8, 9], c, \{11\}, b, [12, 14], d, \{16\}, e$;

$C = a, [15, 16], b, \{19\}, c, \{19\}, d, [23, 24], e$; $D = a, [23, 24], b, [25, 28], c, [27, 29], d, [31, 32], e$;

$E = a, [1, 2], b, [12, 14], d, \{16\}, e$; $F = a, [1, 2], b, [12, 14], d, \{23\}, e$;

$G = a, [1, 2], b, [12, 14], d, [23, 24], e$; $H = a, [1, 2], b, [12, 14], d, [31, 32], e$;

$I = a, [8, 9], c, [18, 19], d, \{23\}, e$; $J = a, [8, 9], c, [18, 19], d, [23, 24], e$.

$K = a, [8, 9], c, [18, 19], d, [31, 32], e$; and $L = a, [8, 9], c, [27, 29], d, [31, 32], e$.

Then, consider the link stream of Figure 1. There are simple cases where each set of shortest paths corresponds to a unique sliding set, like for instance $SP((0, a), (14, e)) = A$, $SP((4, a), (17, e)) = B$, $SP((12, a), (26, e)) = C$, $SP((20, a), (32, e)) = D$, or $SP((0, a), (18, e)) = E$. In most cases, however, the set of shortest paths are disjoint unions (denoted by $\sqcup$) of several sliding sets, like for instance $SP((0, a), (23, e)) = E \sqcup F \sqcup I$, $SP((0, a), (26, e)) = E \sqcup G \sqcup J$, or $SP((0, a), (32, e)) = E \sqcup G \sqcup H \sqcup J \sqcup K \sqcup L$.

Definition 1 (volumes) *The volume of a sliding set $S = v_0, I_1, v_1, I_2, v_2, \cdots I_k, v_k$, denoted by $|S|$, is defined by its* **size** *and* **dimension** *as follows:*

- *If I_i is a singleton for all i, then S contains only one sequence. It has size 1 and dimension 0.*

- *Otherwise, let $I'_1, I'_2, \cdots, I'_l$ be the subsequence of $I_1, I_2, \cdots, I_k$ composed of all its intervals that are not singletons, and let b'_i and e'_i, $b'_i < e'_i$, denote the bounds of I'_i, for all i. Then, $size(S) = \int_{t_1=b'_1}^{e'_1} \int_{t_2=max(t_1,b'_2)}^{e'_2} \cdots \int_{t_l=max(t_{l-1},b'_l)}^{e'_l} 1 \, dt_l \ldots dt_2 \, dt_1$ and $dim(S) = l$.*

In both cases, the volume of S, $|S|$, is defined as the pair $(size(S), dim(S))$ giving its size and dimension.

For instance, the sliding sets above have the following volumes: $|A| = (4, 4)$, $|B| = (2, 2)$, $|C| = (1, 2)$, $|D| = (5.5, 4)$, $|E| = (2, 2)$, $|F| = (2, 2)$, $|G| = (2, 3)$, $|H| = (2, 3)$, $|I| = (1, 2)$, $|J| = (1, 3)$, $|K| = (1, 3)$, and $|L| = (2, 3)$. The case of D is different from the others, as it involves two non-trivially overlapping intervals, namely $[25, 28]$ and $[27, 29]$. Therefore, D may be written as $D = a, [23, 24], b, [25, 27], c, [27, 29], d, [31, 32], e \cup a, [23, 24], b, [27, 28], c, [27, 28], d, [31, 32], e \cup a, [23, 24], b, [27, 28], c, [28, 29], d, [31, 32], e$. The volume of D is then the sum of volumes of these three sliding sets. The first and last ones have volumes $(4, 4)$ and $(1, 4)$, respectively. The middle one has volume $(0.5, 4)$, since it is the set of all sequences of the form $a, t_1, b, t_2, c, t_3, d, t_4$ with t_1 in $[23, 24]$, both t_2 and t_3 in $[27, 28]$, and t_4 in $[30, 31]$, with the constraint that $t_2 \leq t_3$.

More generally, we have the following definitions for volume operations.

Definition 2 (addition, $\boxplus$) *Given two disjoint sliding sets S and S' of volume $|S| = (s, d)$ and $|S'| = (s', d')$, the volume of their union $S \sqcup S'$ is the sum of their two volumes, which we denote by $|S| \boxplus |S'|$. In such a sum, volumes in lower dimensions are negligible, and the sizes of volumes with maximal dimension just add up, so we obtain $|S \sqcup S'| = |S| \boxplus |S'| = (s + s', d)$ if $d = d'$, (s, d) if $d > d'$, and (s', d') if $d' > d$. By extension, any disjoint union of a finite number of sliding sets $S_1, S_2, \cdots, S_k$ has dimension equal to the largest dimension of these sets, and size equal to the sum of the size of all these sets of maximal dimension; we denote its volume by $\boxplus_{i=1}^{k} |S_i| = |S_1| \boxplus |S_2| \boxplus \cdots \boxplus |S_k|$.*

Definition 3 (product, $\boxdot$) *Consider three nodes u, v and w in V, and two sets S and S' such that all elements of S are of the form $u, t_1, v_1, t_2, \cdots, t_k, v$ and the ones of S' are of the form*

$v, t'_1, v'_1, t'_2, \cdots, t'_l, w$, with $t_k \le t'_1$. We denote by $S \cdot S'$ the set of all sequences $u, t_1, v_1, t_2, \cdots, t_k, v, t'_1,$ $v'_1, t'_2, \cdots, t'_l, w$ such that the sequence from u to v is in S and the one from v to w is in S'. If S and S' are disjoint unions of a finite number of sliding sets with $|S| = (s, d)$ and $|S'| = (s', d')$, then $S \cdot S'$ also is the disjoint union of a finite number of sliding sets, and its volume is $|S \cdot S'| =$ $|S| \boxdot |S'| = (s \cdot s', d + d')$.

Definition 4 (quotient, ⬚ and $\doteq$) *Consider S and S' two disjoint unions of sliding sets with $|S| = (s, d)$ and $|S'| = (s', d')$, and such that $S' \subseteq S$. Then necessarily $d' \le d$ and the fraction of elements of S that are also in S', which we denote by $|S'| \boxslash |S|$ or $\frac{|S'|}{|S|}$, is equal to 0 if $d > d'$, and to s'/s if $d = d'$.*

These notations and operations make it easy to describe the set $SP((i, u), (j, v))$ and compute its volume, which is the goal of this section.

In the non-trivial cases above, for instance, $|SP((0, a), (23, e))| = |E \sqcup F \sqcup I| = |E| \boxplus |F| \boxplus |I| = (2, 2) \boxplus (2, 2) \boxplus (1, 2) = (5, 2)$, $|SP((0, a), (26, e))| = |E \sqcup G \sqcup J| = |E| \boxplus |G| \boxplus |J| = (2, 2) \boxplus (2, 3) \boxplus (1, 3) = (3, 3)$, and $|SP((0, a), (32, e))| = |E \sqcup G \sqcup H \sqcup J \sqcup K \sqcup L| = (2, 2) \boxplus (2, 3) \boxplus (2, 3) \boxplus (1, 3) \boxplus (1, 3) \boxplus (2, 3) = (8, 3)$.

We will now prove two lemmas needed to compute the volume of shortest paths from a given temporal node (i, u) in $T \times V$ to another one (j, v) in $T \times V$. Lemma 1 shows how to compute the volume of shortest paths between two consecutive event times. Lemma 2 shows how to decompose the set of shortest paths from a temporal node to another one into a disjoint union of smaller sets of shortest paths.

In all the following, we consider two consecutive event times t and t'. For all x and y in $]t, t'[$, the graphs G_x and G_y are identical. We denote by G_t^+ (or $G_{t'}^-$) this graph, and by $\sigma_t^+(u, v)$ and $d_t^+(u, v)$ (or $\sigma_{t'}^-(u, v)$ and $d_{t'}^-(u, v)$) the (finite) number of shortest paths and the distance from u to v in this graph.

Lemma 1 *Given two nodes x and w, the volume of the set of shortest paths from x to w that start and arrive during $]t, t'[$ is equal to*

$$\left(\sigma_t^+(x, w) \cdot \frac{(t' - t)^{d_t^+(x, w)}}{d_t^+(x, w)!}, \quad d_t^+(x, w) \right).$$

Proof: First notice that if $v_0, t_1, v_1, t_2, v_2, \ldots, t_k, v_k$ is a shortest path from (t, x) to (t', w) in L with $t_i \in]t, t'[$ for all i, then necessarily $v_0, v_1, v_2, \ldots, v_k$ is a shortest path from x to w in G_t^+. Conversely, if $v_0, v_1, v_2, \ldots, v_k$ is a shortest path from x to w in G_t^+ then each sequence $v_0, t_1, v_1, t_2, v_2, \ldots, t_k, v_k$ with $t_i \in]t, t'[$ and $t_i \le t_{i+1}$ for all i is a shortest path from (t, x) to (t', w) in L.

Therefore, the set of shortest paths from x to w that start and arrive during $]t, t'[$ is the disjoint union of $v_0,]t, t'[, v_1,]t, t'[, \cdots,]t, t'[, v_k$ for all shortest path $v_0, v_1, v_2, \ldots, v_k$ from x to w in G_t^+, where $v_0 = x$, $v_k = w$, and $k = d_t^+(x, w)$. It is easy to show by induction that the size of each such sliding set is $\int_{t_1=t}^{t'} \int_{t_2=t_1}^{t'} \cdots \int_{t_k=t_{k-1}}^{t'} 1 \, dt_k \ldots dt_2 \, dt_1 = \frac{(t'-t)^k}{k!}$, and its dimension is k. The volume of $SP((t, x), (t', w))$ is the sum of the volumes of all these sliding sets, and there are $\sigma_t^+(x, w)$ such sliding sets, which completes the proof. □

Lemma 2 *Given (i, u) in $T \times V$ and w in V, we define the two sets $X = \{x \in V, d((i, u), (t, x)) + d_t^+(x, w) = d((i, u), (t', w))\}$ and $Y = \{y \in N_{t'}(w), d((i, u), (t', y)) + 1 = d((i, u), (t', w))\}$. Then, the volume of $SP((i, u), (t', w))$ is the sum of the two following volumes:*

$$\boxplus_{x \in X} \left(|SP((i, u), (t, x))| \boxdot \left(\sigma_t^+(x, w) \cdot \frac{(t' - t)^{d_t^+(x, w)}}{d_t^+(x, w)!}, d_t^+(x, w) \right) \right)$$

and

$$\boxplus_{y \in Y} |SP((i, u), (t', y))|.$$

Proof: Let us denote by A the set $A = \sqcup_{x \in X} SP((i, u), (t, x)) \cdot SP^+((t, x), (t', w))$, where $SP^+((t, x), (t', w))$ is the set of shortest paths from (t, x) to (t', w) that start and arrive during $]t, t'[$. Let us denote by B the set $B = \sqcup_{y \in Y} SP((i, u), (t', y)) \cdot \{(y, t', w)\}$, which means that B is the set obtained when one concatenates any sequence in $SP((i, u), (t', y))$ with $y \in Y$ to the sequence y, t', w. By definition of X and Y, elements of A and B are shortest paths from (i, u) to (t', w), and A and B are disjoint.

Let us now consider a shortest path $P = v_0, t_1, v_1, t_2, v_2, \cdots, t_k, v_k$ in $SP((i, u), (t', w))$, hence $v_0 = u$ and $v_k = w$. We show that P is in A or B. Indeed, if $t_k = t'$ then v_{k-1} is in Y, and $v_0, t_1, v_1, t_2, v_2, \cdots, v_{k-1}$ is in $SP((i, u), (t', v_{k-1}))$, which implies that P is in B. If instead $t_k < t'$, let l be the largest value such that $t_l \leq t$. Then, all t_j with $l < j \leq k$ are in $]t, t'[$ and v_l necessarily is in X. Therefore, $v_l, t_{l+1}, \cdots, t_k, v_k$ necessarily is a shortest path from (t, v_l) to (t', w) that starts and arrives in $]t, t'[$. In addition, $v_0, t_1, v_1, \cdots, t_l, v_l$ is a shortest path from (i, u) to (t, v_l), with $v_l \in X$. Therefore, P is in A.

Finally, $SP((i, u), (t', w))$ is exactly $A \sqcup B$, which, together with Lemma 1 and Definitions 2 and 3 on volume operations, proves the claim. $\square$

These lemmas have crucial consequences: it is in general non-trivial to incrementally compute the volumes of sets of shortest paths, even in quite simple cases. Consider for instance the following two sets of paths in a given link stream, $X = a, [1, 2], b, [3, 4], c, [3, 4], d$ and $Y = a, [1, 2], b, [3, 4], c, [5, 6], d$. These two sets clearly do not have the same volume. Yet, they are both obtained as the set of all paths in $Z = a, [1, 2], b, [3, 4], c$ with an additional jump that may occur within a time interval of duration 1. Knowing the volume of Z and this feature of the additional jump is not sufficient to infer the volume of X or Y: it is crucial to know in addition whether the last time interval of the additional jump is the same as the one of the previous jump. Even worse, knowing the number of such equal intervals is required to compute the expression $\frac{(t'-t)^{d_t^+(x,w)}}{d_t^+(x,w)!}$ in Lemma 1 (this number is nothing but $d_t^+(x, w)$).

This leads to Algorithm 1, that computes the volume of shortest paths from a given temporal node (i, u) to another one (j, v). It starts by computing the volume of shortest paths from (i, u) to (i, w) for any w. Then, in a temporal BFS-like manner, it uses volumes from (i, u) to (t, x) to compute volumes from (i, u) to (t', w), for increasing pairs of consecutive event times t and t'. Indeed, as illustrated in Figure 3, the volumes at t' can be derived from the ones at t. The temporal BFS also uses two queues, named Q and X, to compute the distances that are also needed to compute volumes of shortest paths. It stops when it reaches time j.

Theorem 1 *Given two temporal nodes (i, u) and (j, v) in $T \times V$, Algorithm 1 computes the volume of shortest paths from (i, u) to (j, v).*

Algorithm 1: Volume of shortest paths between two temporal nodes.

---

1 Function *VSP*:

    Input: a link stream $L = (T, V, E)$, $(i, u) \in T \times V$, and $(j, v) \in T \times V$

    Output: volume of shortest paths from (i, u) to (j, v)

2 $Dist \leftarrow$ Dictionary initialized to ∞ for any key

3 $vol \leftarrow$ Dictionary intialized to $(0, 0)$ for any key

4   **for** *each w reachable from u in G_i* **do**

5 $Dist[(i, w)] \leftarrow d_i(u, w)$ and $vol[(i, w)] \leftarrow (\sigma_i(u, w), 0)$

6   **for** *each t, t' consecutive times in $\{i, j\} \cup (\mathcal{T} \cap [i, j])$ in increasing order* **do**

7 $Q \leftarrow$ empty queue

8   set all nodes as unmarked

9 $X \leftarrow$ list of all $(w, Dist[(t, w)])$ in increasing order of $Dist[(t, w)]$

10   **while** *Q or X is not empty* **do**

11 $(w, d) \leftarrow$ get and remove the first element of Q or X with minimal d

12   **if** *w is unmarked* **then** $Dist[(t', w)] \leftarrow d$ and mark w

13   **for** *all unmarked node y in $N_{t'}(w)$* **do** add $(y, d + 1)$ to Q

14   **for** *all marked node w in increasing order of $Dist[(t', w)]$* **do**

15   **for** *all marked node x such that $Dist[(t, x)] + d_t^+(x, w) = Dist[(t', w)]$* **do**

16 $vol[(t', w)] \leftarrow vol[(t', w)] \boxplus vol[(t, x)] \boxdot \left(\sigma_t^+(x, w) \cdot \frac{(t' - t)^{d_t^+(x, w)}}{d_t^+(x, w)!}, d_t^+(x, w) \right)$

17   **for** *all marked node y in $N_{t'}(w)$ such that $Dist[(t', w)] = Dist[(t', y)] + 1$* **do**

18 $vol[(t', w)] \leftarrow vol[(t', w)] \boxplus vol[(t', y)]$
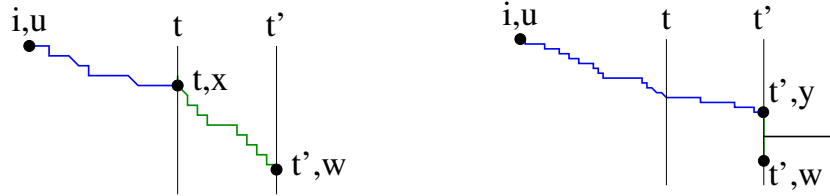
19   **return** $vol[(j, v)]$

Figure 3: If t and t' are two consecutive event times, then a shortest path from (i, u) to (t', w) is the concatenation of either (left) a blue path from (i, u) to a given (t, x) and a green path from this (t, x) to (t', w) in G_t^+; or (right) a blue path from (i, u) to a specific (t', y) and then a jump from y to w at time t' using $(t', yw) \in E_{t'}$.

Proof: Let us consider any time $t < j$ in $\{i, j\} \cup (\mathcal{T} \cap [i, j])$ and let t' be the next time in this set. We show below that, if $Dist[(t, w)] = d((i, u), (t, w))$ and $vol[(t, w)] = |SP((i, u), (t, w))|$ for all w when one enters the main loop at line 6, then at the end of the loop we have $Dist[(t', w)] = d((i, u), (t', w))$ and $vol[(t', w)] = |SP((i, u), (t', w))|$. This is sufficient to prove that the algorithm returns $|SP((i, u), (j, v))|$, since the loop at line 4 initializes $Dist$ and vol correctly.

    Lines 7 to 13 deal with the computation of $d((i, u), (t', w))$ from the distances at time t, for all w. It is similar to a BFS on the graph $G_{t'}$, except that distances at t' are bounded by the

ones at t: $d((i, u), (t', w)) \leq d((i, u), (t, w))$. The loop therefore uses two queues: a list X of nodes in increasing distance at time t, and a queue Q for the exploration of $G_{t'}$. At each round, we consider a node w with minimal distance in these queues: Line 11 takes the first element of X or Q, depending on which has the minimal second field d. This is its actual distance $d((i, u), (t', w))$ (line 12). Then we add its neighbors to Q, together with the information that their distance from (i, u) cannot be larger than $d((i, u), (t', w)) + 1$ (line 13). The loop ends when both X and Q are empty, *i.e.* the distances to all reachable nodes are found.

Then, Lines 14 to 18 deal with the computation of $|SP((i, u), (t', w))|$ from the volumes at time t, for all reachable w. They are a straightforward application of Lemma 2. □

4 Latency pairs

Let us consider a link stream $L = (T, V, E)$, and two nodes u and w in V. The previous section shows how to compute the volume of shortest paths from u to w between two given time instants i and j. However, betweenness computations rely on volumes of shortest *fastest* paths from u to w. These paths are the shortest paths from (s, u) to (a, w) if the latency from (s, u) to (a, w) is equal to $a - s$. We then say that (s, a) in $T \times T$ is a **latency pair** from u to w (in L). This section is devoted to the computation of such latency pairs.

In the case of Figure 1, for instance, $(2, 9)$ is a latency pair from a to e, because the fastest paths from $(2, a)$ to $(9, e)$ start at 2 and end at 9. Similarly, $(9, 16)$, $(16, 23)$ and $(24, 31)$ are the other latency pairs from a to e. Instead, $(3, 8)$ is not a latency pair from a to e since there is no path from $(3, a)$ to $(8, e)$, and $(1, 9)$ is not a latency pair from a to e either because the fastest paths from $(1, a)$ to $(9, e)$ start at time 2.

For any t in T, the pair (t, t) is a latency pair from u to w exactly if there is an instantaneous path between (t, u) and (t, w), *i.e.* there is a path between u and w in G_t. The latency between (t, u) and (t, w) is then equal to 0, and we call (t, t) an **instantaneous latency pair**. In the case of Figure 1, such latency pairs occur from b to d at all times from 12 to 14, at time 19, and at all times from 27 to 28.

Notice that there may exist an infinite amount of instantaneous latency pairs from a node to another one, like in this last example, but there is only a finite number of non-instantaneous latency pairs. Indeed, if (s, a) is a latency pair with $a - s \neq 0$, then s and a necessarily are event times, and as said in Section 2 all link streams considered here have a finite number of event times.

Notice also that if (s, a) is a latency pair from u to w, then there cannot be any latency pair (s', a') from u to w with $[s', a'] \subsetneq [s, a]$. Indeed, this would imply that the latency from (s, u) to (a, w) is equal to $s' - a' < s - a$, which contradicts the fact that (s, a) is a latency pair. This also implies that, if (s, a) is a latency pair with $s \neq a$, then necessarily s and a are event times: otherwise, there is a pair (s', a') such that $[s', a'] \subsetneq [s, a]$, with $G_{s'} = G_s$ and $G_{a'} = G_a$, which would imply that (s', a') also is a latency pair, which contradicts our previous remark.

As a consequence, latency pairs are componentwise ordered: if (s, a) and (s', a') are two distinct latency pairs, then $[s', a'] \not\subseteq [s, a]$ and $[s, a] \not\subseteq [s', a']$. Therefore, either $s < s'$ and $a < a'$, or $s' < s$ and $a' < a$. However, two latency pairs may correspond to overlapping time intervals: $s < s' \leq a < a'$ is possible.

In this section, we compute the **latency list** from u to w, defined as the (finite) componentwise ordered list of all latency pairs (s, a) such that s and a are event times. For instance, in the case of Figure 1, the latency list from a to e is $(2, 9), (9, 16), (16, 23), (24, 31)$, and the latency list from b to d is $(5, 6), (12, 12), (14, 14), (19, 19), (27, 27), (28, 28)$.

Our algorithm considers all event times in increasing order. It maintains the latency lists from a given node to all others before the current event time. It then updates these latency lists for the current time by computing the connected components of the graph at this time. For each of these components, it considers the latest starting time from which a node in this component can be reached, which is given by the previously computed latency lists. This time is the beginning of latency pairs for its nodes, that ends at current time, and so the algorithms updates the lists accordingly.

Algorithm 2: Computation of all latency lists from a given node.

---

1 **Function** *Latency-lists*:
    Input: a link stream $L = (T, V, E)$ and $u \in V$
    Output: for each $w \in V$, the latency list from u to w
2   create $LL \leftarrow$ empty dictionary and $LL[w] \leftarrow$ empty list for all w
3   **for** t *in* $\mathcal{T}$ **do**
4   append (t, t) to $LL[u]$
5   **for** *each connected component C of G_t* **do**
6             $s \leftarrow$ *None* and $X \leftarrow \emptyset$
7   **for** $w \in C$ *with non-empty $LL[w]$* **do**
8                 $(s', a') \leftarrow$ last element of $LL[w]$
9   **if** $s = $ *None* or $s' > s$ **then** $s \leftarrow s'$ and $X \leftarrow \{w\}$
10   **else if** $s' = s$ **then** add w to X
11   **if** X *is non-empty* **then**
12   **for** $w \in C \setminus X$ **do** append (s, t) to $LL[w]$
13   **return** LL

Theorem 2 *Given a link stream $L = (T, V, E)$ and a node $u \in V$, Algorithm 2 computes the ordered latency lists from u to any node $w \in V$.*

Proof: We claim that, at the end of each iteration of the main loop, for all w in V, $LL[w]$ is the list of all latency pairs (s, a) from u to w such that s and a are event times with $a \leq t$.

Assume this is true for all iterations before a given event time t. When it reaches this event time, the loop starts by adding (t, t) to $LL[u]$, which makes the claim true for $w = u$. Consider any connected component C of G_t; the nodes $w \in C$, with non-empty $LL[w]$ are the nodes reachable from u with an arrival time before t or at t. Then, the value of s' computed by the loop at Line 7 is the latest starting time such that one of these nodes is reachable from (s', u) before t or at t, and X is the set of these reachable nodes.

Therefore, if X is non-empty, there exists a path from (s', u) to (t, w) for any $w \in C \setminus X$: for any $x \in X$, the path from (s', u) to (t, x) and then from (t, x) to (t, w) (which exists since x and w are in the same connected component C of G_t) is such a path. As a consequence, (s', t) is a latency pair for any $w \in C \setminus X$. Notice that (s', t) is not a latency pair for any node $x \in X$, $x \neq u$, since they all have a latency pair (s', t_x) with $t_x < t$.

Finally, if the claim is true for all event times lower than t, it is true for t too. It is true for the first iteration, *i.e.* when t is the first event time: it sets $LL[w]$ to $\{(t, t)\}$ for all node w in the same connected component of G_t as u, which is the correct value. Therefore, for all w in V, the

returned value of $LL[w]$ is the list of latency pairs (s, a) from u to w such that s and a are event times, and it is ordered by construction.

$\square$

5 Contribution of a node pair

In all this section, we consider a link stream $L = (T, V, E)$ and two nodes u and w in V. In addition, we consider a temporal node (t, v) in $T \times V$.

For any i and j in T, we denote by $C_{tv}^{ij}(u, w)$ the fraction $\frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))}$ of shortest fastest paths from (i, u) to (j, w) that involve (t, v), and we call it the **contribution of** (i, j). If there is no path from (i, u) to (j, w), we consider that $C_{tv}^{ij}(u, w) = 0$. By extension, we call $\int_{i,j \in T} C_{tv}^{ij}(u, w) \, \mathrm{d}i \, \mathrm{d}j$ the **contribution of** (u, w) to the betweenness of (t, v), and we denote it by $C_{tv}(u, w)$. The goal of this section is to compute $C_{tv}(u, w)$.

First notice that the contribution of (i, j) is derived from volumes of paths as follows. Given x, y and z in $T \times V$, we denote by $SFP(x, y)$ the set of all shortest fastest paths from x to y, and by $SFP(x, y, z)$ the set of these paths that involve z. Then, we define $\sigma(x, y)$ and $\sigma(x, y, z)$ as the volumes of $SFP(x, y)$ and $SFP(x, y, z)$, respectively. It follows that $C_{tv}^{ij}(u, w)$ is equal to $\sigma((i, u), (j, w), (t, v)) \boxdiv \sigma((i, u), (j, w))$ if there is a path from (i, u) to (j, w). Otherwise, $C_{tv}^{ij}(u, w)$ is 0.

This gives a rigorous ground to the definition of $C_{tv}^{ij}(u, w)$, which, as discussed at the end of Section 2, was loosely defined as the fraction $\frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))}$ of shortest fastest paths from (i, u) to (j, w) that involve (t, v); it is indeed equal to the ratio between the two volumes $\sigma((i, u), (j, w), (t, v))$ and $\sigma((i, u), (j, w))$ now defined, with volume ratio operation from Definition 4: $C_{tv}^{ij}(u, w) = \frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))}$.

Consider for instance the case of Figure 1 with $u = a$ and $w = e$, and let us consider $i = 0$ and $j = 18$. Then, the shortest fastest paths from $(i, u) = (0, a)$ to $(j, w) = (18, e)$ are the elements of the set $SFP((0, a), (18, e)) = X \sqcup Y$ where X and Y are the sliding sets $a, \{2\}, b, [3, 5], c, [6, 7], d, \{9\}, e$ and $a, \{9\}, c, \{11\}, b, [12, 14], d, \{16\}, e$, respectively. If $(t, v) = (7.5, c)$ or $(t, v) = (10, b)$, for instance, then none of these paths involve (t, v) and so we obtain a 0 contribution. If $(t, v) = (5.5, c)$ or $(t, v) = (8, d)$, for instance, then all paths in X involve (t, v) and no path in Y does, leading to $C_{tv}^{ij}(u, w) = \sigma((0, a), (18, e), (t, v)) \boxdiv \sigma((0, a), (18, e)) = |X| \boxdiv (|X| \boxplus |Y|) = (2, 2) \boxdiv ((2, 2) \boxplus (2, 1)) = (2, 2) \boxdiv (2, 2) = 1$. If $(t, v) = (10, c)$ or $(t, v) = (14, d)$, then $C_{tv}^{ij}(u, w) = |Y| \boxdiv (|X| \boxplus |Y|) = 0$.

Before presenting the algorithm computing these path volumes and associated contributions, we characterize more precisely which pairs (i, j) have non-zero contribution.

Lemma 3 *There is at most one latency pair from u to w with non-zero contribution.*

Proof: Consider two distinct latency pairs (s, a) and (s', a'); we can assume $s < s'$ and $a < a'$, since, as explained in previous section, $[s', a'] \subseteq [s, a]$ is impossible. Suppose both latency pairs have non-zero contribution: there are shortest fastest paths from (s, u) to (a, w) that involve (t, v) and from (s', u) to (a', w) that also involve (t, v). Therefore, there is a path from (s', u) to (t, v) and a path from (t, v) to (a, w), and so a path from (s', u) to (a, w). It has duration $a - s'$ which is strictly lower than both $a - s$ and $a' - s'$, thus contradicting both that (s, a) and (s', a') are latency pairs. $\square$

If all latency pairs from u to w have contribution 0, then the contribution of (u, w) itself is 0. Otherwise, let us denote by (s, a) the unique latency pair with non-zero contribution.

We now introduce **two specific times, S and A**, that we will use to find all time instants with non-zero contribution. We define $]S, A[$ as the largest interval containing $]s, a[$ such that: for all other latency pair (s', a') in this interval, either $a' - s' > a - s$, or $a' - s' = a - s$ and $d((s', u), (a', w)) \geq d((s, u), (a, w))$; and the number of instantaneous paths from (S, u) to (A, w) of length $d((s, u), (a, w))$ is finite. We illustrate this definition in Figure 4.
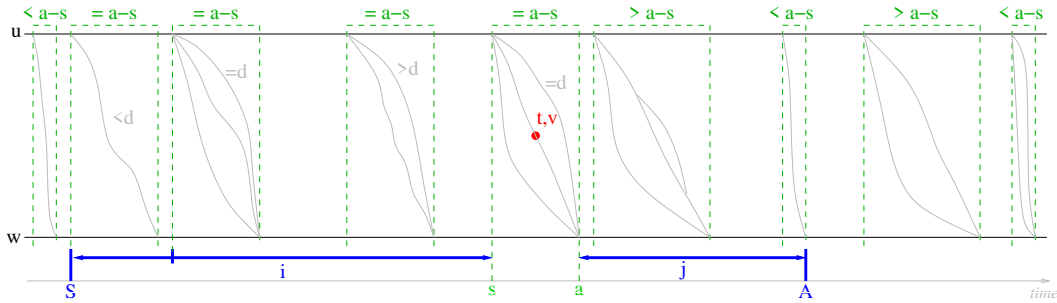

Figure 4: An abstract example of link stream $L = (T, V, E)$ in which we consider a specific (t, v) in $T \times V$ (in red), two nodes u and w in V (in black, horizontal lines), as well as the latency pair (s, a) containing t such that shortest (necessarily fastest) paths from (s, u) to (a, w) have length d and some of them involve (t, v). We display all latency pairs from u to w with two green vertical lines topped by a dotted horizontal line indicating the corresponding latencies ($= a - s$, $< a - s$ or $> a - s$). In addition, we also indicate the length ($= d$, $< d$ or $> d$) of corresponding shortest paths within each latency pair, when this is useful (in grey). We indicate in blue the two specific times S and A defined above, as well as the time periods for i and j such that the contribution of (i, j) may be non-zero (Lemma 4).

We then have the following result.

Lemma 4 *All pairs (i, j) in $T \times T$ that have non-zero contribution are in $[S, s] \times [a, A]$.*

Proof: If a given pair (i, j) has non-zero contribution, then there is a latency pair (s', a') with $s' \geq i$ and $a' \leq j$ that has non-zero contribution. Remind that (s, a) is itself such a latency pair. From Lemma 3, we then have $(s', a') = (s, a)$, and so $i \leq s$ and $j \geq a$.

If $i < S$ and $j \geq a$, or if $i \leq s$ and $j > A$, then by definition of S and A we are in one of the following situations.

There exists a latency pair (s', a') in $[i, j]$ such that: either $a' - s' < a - s$, or $a' - s' = a - s$ and $d((s', u), (a', w)) < d((s, u), (a, w))$. Then, shortest fastest paths from (s, u) to (a, w) are not shortest fastest paths from (i, u) to (j, w). All shortest fastest paths from (i, u) to (j, w) are from (s', u) to (a', w) where (s', a') is a latency pair as described above. Suppose such a shortest fastest path involves (t, v). Then there are paths from (s', u) to (t, v) and from (t, v) to (a', w). As a consequence, $s' \notin [s, t]$, otherwise (s, a) would not be a latency pair. Likewise, $a' \notin [t, a]$. Therefore, $s' < s$ and $a' > a$, but this contradicts the fact that $a' - s' \leq s - a$. This means that shortest fastest paths from (s', u) to (a', w) cannot involve (t, v), and so the contribution of (i, j) is 0.

Or there is an infinite number of instantaneous paths from (i, u) to (j, w) with length $d((s, u), (a, w))$. Only the $\sigma_t(u, w)$ ones starting and arriving at time t involve (t, v). There is a finite number of such paths, as they are paths in the graph G_t. Therefore, the contribution of (i, j) is zero.

In conclusion, $i \leq s$, $j \geq a$, i cannot be smaller than S, and j cannot be larger than A, which proves the claim. □

This lemma says that all pairs (i, j) with non-zero contribution are in $[S, s] \times [a, A]$. Notice however that some pairs (i, j) in $[S, s] \times [a, A]$ may have a contribution equal to 0. This happens whenever the volume of shortest fastest paths from (s, u) to (a, w) has a lower dimension than the one from (i, u) to (j, w).

We now define specific latency pairs that play a special role, as any shortest fastest path from (i, u) to (j, w) must start and arrive within one of these pairs. To do this, we introduce an **ordered list** LP **of latency pairs** centered on (s, a), which means that latency pairs preceding (s, a) have negative indexes in the list and the others have positive indexes. It is the list $LP = (s_{-l}, a_{-l}), (s_{-l+1}, a_{-l+1})), \ldots, (s_0 = s, a_0 = a), \ldots, (s_r, a_r)$ such that, for all k, $[s_k, a_k] \subseteq [S, A]$, $a_k - s_k = a - s$, and $d((s_k, u), (a_k, w)) = d((s, u), (a, w))$. We also define $s_{-l-1} = S$ and $a_{r+1} = A$. Notice that $s_{-l-1} = s_l$ or $a_r = a_{r+1}$ are not forbidden; this happens for instance when $s_{-l} = \alpha$ or $a_r = \omega$. We show now that the latency pairs in LP give precisely the shortest fastest paths from u to w.

Lemma 5 *For any pair (i, j) in $]S, s] \times [a, A[$, the set $SFP((i, u), (j, w))$ is the disjoint union of all sets $SFP((s_k, u), (a_k, w))$ such that (s_k, a_k) in LP and $[s_k, a_k] \subseteq [i, j]$.*

Proof: We first show that for any k such that $[s_k, a_k] \subseteq [i, j]$, $SFP((s_k, u), (a_k, w)) \subseteq SFP((i, u), (j, w))$. Let us consider a path in $SFP((s_k, u), (a_k, w))$. Since $[s_k, a_k] \subseteq [i, j]$, it is a path from (i, u) to (j, w). It has duration $s_k - a_k = s - a$ because (s_k, a_k) is in LP. Moreover, since $i > S$ and $j < A$, there exists no latency pair (s', a') such that $[s', a'] \subseteq [i, j]$ and $a' - s' < a - s$. Therefore, it is a fastest path from (i, u) to (j, w). Similarly, because (s_k, a_k) is in LP, this path has length $d((s, u), (a, w))$ and therefore it is a shortest fastest path from (i, u) to (j, w).

Now consider any shortest fastest path from (i, u) to (j, w), and let us denote by s' and a' its starting and arrival times. Since it is a fastest path, (s', a') is a latency pair, and obviously $[s', a'] \subseteq [i, j]$. In addition, $s' - a' = s - a$: if it was larger then the paths from (s', u) to (a', w) would not be fastest paths from (i, u) to (j, w); and if it was smaller, then the paths from (s, u) to (a, w) would not be fastest paths from (i, u) to (j, w). Similarly, $d((s', u), (a', w)) = d((s, u), (a, w))$: if it was larger then the paths from (s', u) to (a', w) would not be shortest paths from (i, u) to (j, w); and if it was smaller, then the paths from (s, u) to (a, w) would not be shortest paths from (i, u) to (j, w). Therefore, (s', a') is in LP, leading to the fact that $SFP((i, u), (j, w))$ is included in the union of all sets $SFP((s_k, u), (a_k, w))$ such that (s_k, a_k) in LP and $[s_k, a_k] \subseteq [i, j]$.

Finally, notice that the sets $SFP((s_k, u), (a_k, w))$ are disjoint for different values of k, since all the paths they contain start at s_k and arrive at a_k. We therefore obtain the claim. □

Thanks to these results, we obtain an expression giving the contribution of a node pair as a discrete sum.

Lemma 6 *The contribution of (u, w) to the betweenness of (t, v), i.e. the fraction of shortest fastest paths from u to w that involve (t, v), namely $C_{tv}(u, w) = \int_{i,j \in T} \frac{\sigma((i,u),(j,w),(t,v))}{\sigma((i,u),(j,w))} \, \mathrm{d}i \, \mathrm{d}j$, can be written as a discrete sum:*

$$C_{tv}(u, w) = \sum_{k=-l-1}^{-1} \sum_{k'=0}^{r} (s_{k+1} - s_k)(a_{k'+1} - a_{k'}) \frac{\sigma((s, u), (a, w), (t, v))}{\boxplus_{h=k+1}^{k'} \sigma((s_h, u), (a_h, w))}. \tag{1}$$

Algorithm 3: Compute the list of starting times for latency pairs in LP, as well as associated volumes of sets of shortest fastest paths.

---

1 **Function** *PrevList*:

    Input: a link stream $L = (T, V, E)$, $u \in V$, $w \in V$, (s, a) a latency pair from u to w, and the ordered latency list LL from u to w

    Output: the list $((s_{-1}, f_{-1}), (s_{-2}, f_{-2}), \ldots, (s_{-l-1} = S, f_{-l-1}))$ with s_k defined by LP and with $f_k = \sigma((s_{k+1}, u), (a_{-1}, w))$

2   init *Result* to empty list and *vol* to $(0, 0)$

3   **if** $s = a$ and $d_s^-(u, w) = d((s, u), (a, w))$ **then**

4         $\lfloor$ **return** *Result*

5   **foreach** (s', a') *with* $s' < s$ *in* LL *backwards* **do**

6   **if** $a' - s' < a - s$ **then**

7             $\lfloor$ append (s', vol) to *Result* and **return** *Result*

8   **if** $a' - s' = a - s$ **then**

9   **if** $d((s', u), (a', w)) < d((s, u), (a, w))$ **then**

10                $\lfloor$ append (s', vol) to *Result* and **return** *Result*

11   **if** $d((s', u), (a', w)) = d((s, u), (a, w))$ **then**

12   append (s', vol) to *Result*

13   **if** $s' = a'$ and $d_{s'}^-(u, w) = d((s, u), (a, w))$ **then**

14                   $\lfloor$ **return** *Result*

15                $vol \leftarrow vol \boxplus \mathrm{VSP}(L, (s', u), (a', w))$

16   append (α, vol) to *Result* and **return** *Result*

Proof: According to Lemma 4, the contribution of time instants (i, j) is equal to zero whenever $(i, j) \notin [S, s] \times [A, a]$. For $(i, j) \in [S, s] \times [A, a]$, all shortest fastest paths from (i, u) to (j, w) involving (t, v) start at time s and arrive at time a and the contribution of (i, j) is therefore equal to $\sigma((s, u), (a, w), (t, v)) \boxdiv \sigma((i, u), (j, w))$. Therefore, $C_{tv}(u, w) = \int_{[S,s] \times [a,A]} \frac{\sigma((s,u),(a,w),(t,v))}{\sigma((i,u),(j,w))} \, \mathrm{d}i \, \mathrm{d}j = \int_{]S,s] \times [a,A[} \frac{\sigma((s,u),(a,w),(t,v))}{\sigma((i,u),(j,w))} \, \mathrm{d}i \, \mathrm{d}j$.

According to Lemma 5, for any $k < 0$, any $k' \geq 0$, any $i \in]s_k, s_{k+1}]$, and any $j \in [a_{k'}, a_{k'+1}[$, the value of $\sigma((i, u), (j, w))$ is constant and it is equal to $\boxplus_{h=k+1}^{k'} \sigma((s_h, u), (a_h, w))$.

Therefore,

$$\int_{s_k}^{s_{k+1}} \int_{a_{k'}}^{a_{k'+1}} \frac{\sigma((i, u), (j, w), (t, v))}{\sigma((i, u), (j, w))} \, \mathrm{d}i \, \mathrm{d}j = \int_{s_k}^{s_{k+1}} \int_{a_{k'}}^{a_{k'+1}} \frac{\sigma((s, u), (a, w), (t, v))}{\sigma((i, u), (j, w))} \, \mathrm{d}i \, \mathrm{d}j$$

$$= \int_{s_k}^{s_{k+1}} \int_{a_{k'}}^{a_{k'+1}} \frac{\sigma((s, u), (a, w), (t, v))}{\boxplus_{h=k+1}^{k'} \sigma((s_h, u), (a_h, w))} \, \mathrm{d}i \, \mathrm{d}j$$

$$= (s_{k+1} - s_k)(a_{k'+1} - a_{k'}) \frac{\sigma((s, u), (a, w), (t, v))}{\boxplus_{h=k+1}^{k'} \sigma((s_h, u), (a_h, w))}$$

and we obtain the claim. $\qquad\qquad\square$

In order to compute the sum of Lemma 6, we need to iterate over all s_k, $-l-1 \leq k \leq -1$ and all $a_{k'}$, $0 \leq k' \leq r$. For this purpose, we first give an algorithm computing the values of s_k. The algorithm also associates to each s_k a volume of shortest fastest paths that will be useful for computing the denominator of the fraction in the sum.

Lemma 7 *Algorithm 3 computes the list* $(s_{-1}, f_{-1}), (s_{-2}, f_{-2}), \ldots, (s_{-l-1} = S, f_{-l-1})$ *with* s_k *defined by LP and with* $f_k = \sigma((s_{k+1}, u), (a_{-1}, w))$.

Proof: The algorithm builds and returns the (initially empty) list *Result*. The algorithm terminates when S is found. Indeed, a return is triggered in three different cases. If the empty list is returned at Line 4, this means that there exists an $\epsilon > 0$ such that for all $t \in [s - \epsilon, s]$, there is an instantaneous path of length $d((s, u), (a, w))$ from (t, u) to (t, w), which implies that $S = s$. If the return happens after the last value is added to *Result* during the *for* loop, then either $s' < s$, and s' is the largest value such that: $a' - s' < a - s$ (Line 7); or $a' - s' = a - s$ and $d((s', u), (a', w)) < d((s, u), (a, w))$ (Line 10); or there exists $\epsilon > 0$ such that for any $t \in [s' - \epsilon, s']$ there is an instantaneous path of length $d((s, u), (a, w))$ from (t, u) to (t, w) (Line 14). This corresponds exactly to the definition of S. Finally if the function returns at Line 16, then this means that $S = \alpha$ because none of the above conditions is true for any $s' > \alpha$.

The elements (s', vol) added to *Result* correspond to all latency pairs (s', a') such that $s' \in [S, s[$, $a' - s' = a - s$, and $d((s', u), (a', w)) = d((s, u), (a, w))$. These are therefore the (s_i, a_i) in *LP* with $i < 0$.

Let us now show that *vol* contains the desired value when (s', vol) is added to *Result*.

If the empty list is returned at Line 4, then this is true. Otherwise, *vol* is initialized to $(0, 0)$ and this value is not changed before the first time the pair (s', vol) is appended to *Result*. Therefore the first pair appended to *Result* is $(s_{-1}, (0, 0))$, which is correct since $f_{-1} = \sigma((s_0, u), (a_{-1}, w)) = (0, 0)$ (since there are no paths from (s_0, u) to (a_{-1}, w)).

Assume now that the correct value (s_i, f_i) has been added to *Result* at one loop iteration, and that $s_i > S$ (otherwise, as shown above, a return is triggered just after the append and the function returns). We then have $s' = s_i$ and the value $\sigma((s', u), (a', w))$ is then added to *vol*. *vol* is therefore now equal to $\sigma((s', u), (a', w)) \boxplus f_i = \sigma((s_i, u), (a_i, w)) \boxplus \sigma((s_{i+1}, u), (a_{-1}, w)) = f_{i-1}$. Moreover, the loop will skip latency pairs not in *LP* and the next value of s' that will be considered is s_{i-1}. Therefore the next value that is added to *Result* is (s_{i-1}, f_{i-1}) and finally all the correct values are added to *Result*, which completes the proof. $\qquad\square$

We also introduce the function *NextList* by replacing in *PrevList* of Algorithm 3: d_s^- by d_a^+ in Line 3; s by a in Line 4; $s' < s$ in *LL backwards* by $a' > a$ in *LL forwards* in Line 5; all (s', vol) appended to *Result* by (a', vol); $d_{s'}^-$ by $d_{a'}^+$ in Line 13; and (α, vol) by (ω, vol) in the last line.

The obtained function computes the list $(a_1, g_1), (a_2, g_2), \ldots, (a_{r+1} = A, g_{r+1})$ with a_k defined by *LP* and with $g_k = \sigma((s_1, u), (a_{k-1}, w))$.

We finally reach the objective of this section.

Theorem 3 *Given a link stream* $L = (T, V, E)$, *a temporal node* (t, v) *in* $T \times V$, *and two nodes* u *and* w *in* V, *Algorithm 4 computes the contribution of* u *and* w *to the betweenness of* (t, v), *i.e.*

$$C_{tv}(u, w) = \int_{i \in T, j \in T} \frac{\sigma((i, u), (j, w), (t, v))}{\sigma((i, u), (j, w))} \, di \, dj.$$

Proof: If there exists a latency pair (s, a) with non-zero contribution, then, for any $i \leq s$ and any $j \geq a$, we have: $\sigma((i, u), (j, w), (t, v)) = \sigma((s, u), (a, w), (t, v))$. The *for* loop of Line 3 computes

---

Algorithm 4: Contribution of two given nodes to the betweenness of a given temporal node

---

1 Function *Contribution***:**

    Input: a link stream $L = (T, V, E)$, $u \in V$, $w \in V$, $(t, v) \in T \times V$, and the latency list LL from u to w

    Output: the contribution $\int_{T \times T} \sigma((i, u), (j, w), (t, v)) \boxtimes \sigma((i, u), (j, w)) \, \mathrm{d}i \, \mathrm{d}j$

2 $vol\_tv \leftarrow (0, 0)$

3   **for** (x, y) *in* LL **do**

4   **if** $t \in [x, y]$ *and* $(x, u) \longrightarrow (t, v)$ *and* $(t, v) \longrightarrow (y, w)$ **then**

5   **if** $d((x, u), (y, w)) = d((x, u), (t, v)) + d((t, v), (y, w))$ **then**

6 $vol\_tv \leftarrow VSP(L, (x, u), (t, v)) \odot VSP(L, (t, v), (y, w))$

7   set (s, a) to (x, y) and exit the loop

8   **if** $vol\_tv = (0, 0)$ **then**

9   **return** 0

10 $middle \leftarrow \mathrm{VSP}(L, (s, u), (a, w))$

11 $Prev \leftarrow \mathrm{PrevList}(L, u, w, s, a, LL)$

12 $Next \leftarrow \mathrm{NextList}(L, u, w, s, a, LL)$

13 $contrib \leftarrow 0$

14 $s' \leftarrow s$

15   **for** $(s\_left, left)$ *in* $Prev$ **do**

16 $a' \leftarrow a$

17   **for** $(a\_right, right)$ *in* $Next$ **do**

18 $contrib \leftarrow contrib + (s' - s\_left)(a\_right - a') \cdot vol\_tv \boxtimes (left \boxplus right \boxplus middle)$

19 $a' \leftarrow a\_right$

20 $s' \leftarrow s\_left$

21   **return** *contrib*

$\sigma((s, u), (a, w), (t, w))$ and stores it in *vol_tv*. Indeed, since (t, v) is involved in a shortest fastest path from (s, u) to (a, w), necessarily (s, a) is such that $t \in [s, a]$, $(s, u) \longrightarrow (t, v)$, $(t, v) \longrightarrow (a, w)$, and $d((s, u), (a, w)) = d((s, u), (t, v)) + d((t, v), (a, w))$. Since there is at most one such pair satisfying the first three conditions, the algorithm breaks out of the *for* loop if one is found. If no such latency pair is found, *vol_tv* is equal to $(0, 0)$ at the end of the loop and the Algorithm returns 0. Notice that, in the special case where t is not an event time and $(t, u) \longrightarrow (t, w)$, then the arguments above do not apply, but the algorithm still returns the correct value: (t, t) is a latency pair that does not belong to the latency list, and the contribution of (u, w) is 0, which is the returned value.

Remember that $LP = (s_{-l}, a_{-l}), (s_{-l+1}, a_{-l+1})), \ldots, (s_0 = s, a_0 = a), \ldots, (s_r, a_r)$ and that $s_{-l-1} = S$ and $a_{r+1} = A$. *PrevList* (Algorithm 3) then computes, according to Lemma 7, the list $Prev = ((s_{-1}, f_{-1}), (s_{-2}, f_{-2}), \ldots, (s_{-l-1} = S, f_{-l-1})$ with s_k defined by LP and with $f_k = \sigma((s_{k+1}, u), (a_{-1}, w))$; its dual algorithm *NextList* computes the list $Next = (a_1, g_1), (a_2, g_2), \ldots, (a_{r+1} = A, g_{r+1})$ with a_k defined by LP and with $g_k = \sigma((s_1, u), (a_{k-1}, w))$.

According to Lemma 6,

$$C_{tv}(u, w) = \sum_{k=-l-1}^{-1} \sum_{k'=0}^{r} (s_{k+1} - s_k) \cdot (a_{k'+1} - a_{k'}) \cdot \frac{\sigma((s,u),(a,w),(t,v))}{\boxplus_{h=k+1}^{k'} \sigma((s_h, u),(a_h, w))}.$$

Lines 15 to 20 of Algorithm 4 compute this sum. First notice that s' is initialized to $s = s_0$ and s_left loops over values in *Prev*, starting with s_{-1}. At the end of each iteration s' is set to s_left and therefore s' and s_left loop over all consecutive values s_{k+1}, s_k for s_k in *Prev*. The value of $left$ is $f_k = \sigma((s_{k+1}, u),(a_{-1}, w)) = \boxplus_{h=k+1}^{-1} \sigma((s_h, u),(a_h, w))$, as explained in the characterization of *Prev* above.

Similarly, in the inner *for* loop a' and a_right loop over all values $a_{k'}, a_{k'+1}$ for $a_{k'+1}$ in *Next*, and $right = g_{k'+1} = \sigma((s_1, u),(a_{k'}, w)) = \boxplus_{h=1}^{k'} \sigma((s_h, u),(a_h, w))$.

The value of $middle$ has been set to $\sigma((s, u),(a, w))$ (Line 10). Therefore $left \boxplus right \boxplus middle = \boxplus_{h=k+1}^{k'} \sigma((s_h, u),(a_h, w))$. Finally, one iteration of the inner loop adds the value $(s_{k+1} - s_k)(a_{k'+1} - a_{k'}) \sigma((s, u),(a, w)) \boxdiv \boxplus_{h=k+1}^{k'} \sigma((s_h, u),(a_h, w))$, which is exactly one term of the sum in Equation 1, and the loop itself ensures we obtain the whole sum. □

6 Betweenness of a temporal node

We now have all needed building blocks for computing the betweenness of any given temporal node: we just have to sum the contribution of each node pair, see Algorithm 5.

Algorithm 5: Betweenness of a temporal node

1 **Function** *Betweenness*:
    Input: a link stream $L = (T, V, E)$ and $(t, v) \in T \times V$
    Output: the betweenness of (t, v)
2     $B \leftarrow 0$
3   **for** $u \in V$ **do**
4         $LL \leftarrow$ Latency-lists(u)
5   **for** $w \in V$ **do**
6             $B \leftarrow B + $ Contribution$(L, u, w, (t, v), LL[w])$

7   **return** B

The key variables describing the size of our algorithm inputs, for a given link stream $L = (T, V, E)$, are the number of nodes $n = |V|$ and the number of link segments $\overline{m}$, *i.e.* the number of maximal intervals in E. Notice that the number of event times $|\mathcal{T}|$ is at most $2 \cdot \overline{m}$, and so it is in $O(\overline{m})$. Likewise, the number of links $m_t = |\{uv, (t, uv) \in E\}|$ at time t, for any t, as well as the number of links $m = |\{uv, \exists t, (t, uv) \in E\}|$ in the induced graph, also are in $O(\overline{m})$. Then, the complexity of all algorithms presented in this paper is clearly polynomial in n and $\overline{m}$, which makes Algorithm 5 polynomial itself.

We display in Figure 5 the results obtained in the case of Figure 1, where we computed the betweenness of more than 5000 temporal nodes in a few seconds. We provide the implementation at [19].
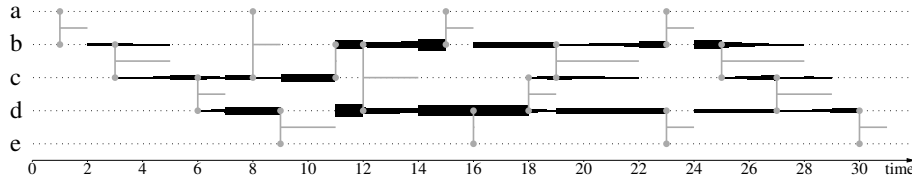
Figure 5: Results of betweenness computations on the example of Figure 1. We computed the betweenness of (t, v) for all v and t equal to $\alpha + i \cdot \frac{\omega - \alpha}{1000}$, for $i = 0..1000$. The obtained value is displayed at (t, v) as a black rectangle of width $\frac{\omega - \alpha}{1000}$ and height propotional to the betweenness of (t, v). Dotted lines represent betweenness values equal to 0.

7   Related work

Betweenness computations are first related to **path computations**. Temporal paths already received much attention, in particular optimal path computations according to several criteria (like length, duration, and/or arrival time), see for instance [5, 39, 40]. However, most of these works are limited to discrete time and instantaneous links; only few consider continuous time and links with duration [37, 41, 29]. Then, the focus is on finding optimal paths or computing distances and latencies [29], not counting them as we do here. The authors of [1] notice that the number of foremost paths (temporal paths with minimal arrival time) may be exponential. The problem that we consider here is quite different because we handle continuous time and links with duration. This leads to the concept of finite volumes of uncountable path sets, that never appeared in previous literature, up to our knowledge.

The graph betweenness itself also has been studied in temporal settings. A first line of study focuses on **updating betweenness values** upon link arrival or departure, see for instance [3, 13]. This is quite different from our work: the considered paths are classical (static) graph paths, and the considered betweenness is the classical one, at each time instant.

Several works consider **temporal betweenness extensions** that rely on various kinds of optimal (fastest, shortests, foremost, etc) paths. Among those, Rymar et al. [28] is the closest to our work. The authors propose an algorithm similar to the one by Brandes [4] for the corresponding centralities.

This work, and the works discussed below, assume discrete time, which implies finite sets of shortest paths. Instead, we consider continuous time, leading to uncountable sets of paths, with finite volume which requires to introduce the notion of *volume* to quantify them. In addition, these works keep a partly node-centric point of view by considering paths between nodes; we push the integration of temporal aspects further by considering paths between temporal nodes. This makes an important difference, since the node-centric view misses locally-optimal paths: they only count paths with a given duration or length between pairs of nodes (for any starting and arrival times), whereas our approach combines a variety of locally shortest fastest paths, with different durations and lengths. because in our case *all* paths that are shortest fastest paths for *some* source and destination temporal nodes are taken into account in the computation, with weights depending on the amount of source and destination nodes they correspond to. The computation of these weights is the focus of Section 5. This raises different algorithmic challenges, like the computation of latency lists and the selection of appropriate contributing latency pairs.

Most temporal betweenness extensions have a node-centric view: they define a value for each node, not for each temporal node, see for instance [15, 32, 24, 38, 17]. Others define a value for each

temporal node, like in our case. For instance, [31] proposes *coverage centrality* of (t, v), defined as the fraction of pairs of (non-temporal) nodes for which there exists a fastest path involving (t, v). Buß et al. [6] deals with betweenness centrality of temporal nodes, in the case of instantaneous links and with various types of optimal temporal paths. The authors of [32, 14] also define a betweenness value for each temporal node, based on foremost paths or other optimal paths. The algorithm in [14] starts by identifying time instants for which foremost path trees are stable, which is related to our latency pairs. In [33], the authors combine the length and duration of paths using a tunable parameter, and focus on instantaneous links.

Another line of works uses **algebraic approaches** to compute path-related features like distances in graphs [12]. In particular, the authors of [34] design a semiring structure in which they express the Brandes algorithm for betweenness [4]. Such approaches are also used in the case of temporal graphs [18, 26, 2, 9, 38]. They generally use a matrix that gives the number and length of shortest paths between each pair of nodes. Then, they define algebraic operators that compute the number of paths corresponding to unions and concatenations of sets of paths. Indeed, shortest paths from a node a to another one c are in general concatenations of shortest paths from a to b and shortest paths from b to c, for various intermediate nodes b. Then, it is in most cases easy to infer the number of shortest paths from a to c from the ones from a to b and from b to c, with b in a well chosen set of nodes. As seen in Section 3, this does not hold for shortest fastest paths in link streams, though: computing the volume corresponding to the concatenation of two sets of such paths requires more information than just the volume of these two sets. In addition, a prefix of a shortest fastest path is not necessarily a shortest fastest path itself, a condition usually required by algebraic approaches (and true for shortest paths in graphs). Extending existing algebraic approaches to betweenness in link streams, although appealing, therefore seems non-trivial. It would at least require to store additional information in order to handle the more complex concatenation operations. For instance, one may store the number of identical time intervals in the end of the sliding sets under concern. Even then, it is unclear what matrix operations would lead to the wanted results, if they exist, and that the corresponding structure would be a semiring. Moreover this would not necessarily be simpler than Algorithm 1. Finally, deepening these questions is a non-trivial perspective that seems to raise significant challenges; we leave them for future work.

In another approach, [1] and [25] obtain a betweenness value for each node for each time slice by considering optimal paths within time slices. Again, they only consider discrete time, and only a limited number of source and target temporal nodes.

Finally, the generalized betweenness that we consider in this paper, by dealing with continuous time, links with or without duration, as well as paths between all pairs of temporal nodes, raises original algorithmic questions that are not present in previous literature.

8 Conclusion

We presented the first algorithms to compute betweenness centrality of temporal nodes in link streams. To obtain these algorithms, we identified and addressed several original challenges, like the definition and computation of volumes of infinite sets of paths, the computation of all latency pairs from any node to all others, or the transformation of continuous-time integrals into discrete sums over finite numbers of time intervals. Each of these building blocks has its own interest, in particular the computation of shortest path volumes from a given temporal node. The complexity of obtained algorithms is polynomial in time and space, and we provide an implementation in python [19].

Our algorithm leaves room for complexity improvement. In particular, it seems promising

to explore extensions to link streams of approaches like Brandes' for betweenness on graphs [4]. Another important direction is to design algorithms to compute the betweeness of *all* temporal nodes rather than just one: iterating our algorithm over many temporal nodes leads to much redundancy. However, keep in mind that there is an infinite number of temporal nodes; one may then try to infer the betweenness of any of them from the betweenness of a finite number of them, for instance each node at each event time. This seems non-trivial, though, and an open question.

Going further, one may try to design approximate algorithms. Indeed, the best known time complexity of betweenness computations in graphs is $O(nm)$ [4] and it cannot be lower in link streams, since graphs are special cases [21]. This is prohibitive in many practical cases, leading to much work on approximate computations, that typically compute shortest paths from some nodes only [27, 35]. Such approaches are very relevant in link streams too, where the contribution of only a few node pairs may give reasonably accurate approximates, at a much lower cost than exact computations. This remains to explore, though.

An even more challenging direction is to embrace the streaming nature of link streams, and design on-line and/or streaming algorithms for betweenness. Such algorithms do not store the data in memory; they compute results on-the-fly and output them as soon as they are available. They would be of high theoretical and practical interest, but they raise many challenges.

Another interesting family of perspectives consists in extending or restricting the considered input. In particular, one may consider stream graphs instead of link streams: in stream graphs, nodes are not always present, leading to more subtle path, distance, and latency concepts [21]. We considered here streams with link (and node) presence times equal to unions of disjoint closed intervals (including singletons); another extension would be to consider more general cases, like for instance unions of disjoint closed or open intervals. Also, weighted and/or directed stream graphs and link streams [20] lead to more complex concepts of shortest fastest paths, and our definitions of volumes may be extended to these cases. Conversely, one may consider more specific situations, like discrete time streams, or link stream with instantaneous links only. Such cases often appear in practice, and it may be possible to design more efficient algorithms for them.

Extending our algorithms to variants of the betweenness concept itself also is an interesting perspective. One may for instance consider betweenness of *links* rather nodes, or consider paths of other kinds than shortest fastest ones, *e.g.* foremost ones [21]

Finally, this paper opens the perspective of practical uses of betweenness in link streams, since until now only the definition was available. It is now possible to explore how betweenness is distributed in (small scale) real-world cases, and gain insight from this. It may also be used to extend important graph algorithms to link streams, like the computation of communities by iteratively removing temporal nodes of highest betweenness, in a way similar to [11] that iteratively removes links of highest betweenness.

Acknowledgements

References

[1] A. Afrasiabi Rad, P. Flocchini, and J. Gaudet. Computation and analysis of temporal betweenness in a knowledge mobilization network. *Computational Social Networks*, 4(1), dec

2017. `doi:10.1186/s40649-017-0041-7`.

[2] V. Batagelj and S. Praprotnik. An algebraic approach to temporal network analysis based on temporal quantities. *Social Netw. Analys. Mining*, 6(1):28:1–28:22, 2016. `doi:10.1007/s13278-016-0330-4`.

[3] E. Bergamini, H. Meyerhenke, M. Ortmann, and A. Slobbe. Faster betweenness centrality updates in evolving networks. In *Proceedings of the Symposium on Experimental Algorithms (SEA)*, 2017. `doi:10.4230/LIPIcs.SEA.2017.23`.

[4] U. Brandes. A Faster Algorithm for Betweenness Centrality. In *Journal of Mathematical Sociology*, volume 25, pages 163–177, 2001. `doi:10.1080/0022250X.2001.9990249`.

[5] B.-M. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, nov 2003. `doi:10.1142/S0129054103001728`.

[6] S. Buß, Hendrik, R. Niedermeier, and M. Rymar. Algorithmic aspects of temporal betweenness. In *Proceedings of the 26th SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2020. `doi:10.1145/3394486.3403259`.

[7] C. T. Butts. A relational event framework for social action. *Sociological Methodology*, 38(1):155–200, 2008. `doi:10.1111/j.1467-9531.2008.00203.x`.

[8] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012. `doi:10.1080/17445760.2012.668546`.

[9] L. Falzon, E. Quintane, J. Dunn, and G. Robins. Embedding time in positions: Temporal measures of centrality for social network analysis. *Social Networks*, 54:168–178, 2018. `doi:10.1016/j.socnet.2018.02.002`.

[10] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977. `doi:10.2307/3033543`.

[11] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002. `doi:10.1073/pnas.122653799`.

[12] M. Gondran and M. Minoux. *Graphs, Dioids And Semirings*. Springer, 2008.

[13] O. Green, R. McColl, and D. A. Bader. A fast algorithm for streaming betweenness centrality. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 11–20, 2012. `doi:10.1109/SocialCom-PASSAT.2012.37`.

[14] V. M. Gunturi, S. Shekhar, K. Joseph, and K. M. Carley. Scalable computational techniques for centrality metrics on temporally detailed social network. *Machine Learning*, 106(8):1133–1169, aug 2017. `doi:10.1007/s10994-016-5583-7`.

[15] Habiba, C. Tantipathananandh, and T. Berger-Wolf. Betweenness Centrality Measure in Dynamic Networks. Technical report, 2011. URL: `https://www.researchgate.net/publication/265324205_Betweenness_Centrality_Measure_in_Dynamic_Networks`.

[16] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97 – 125, 2012. Temporal Networks. `doi:10.1016/j.physrep.2012.03.001`.

[17] H. Kim and R. Anderson. Temporal node centrality in complex networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 85(2):026107, feb 2012. `doi:10.1103/PhysRevE.85.026107`.

[18] N. Kontoleon, L. Falzon, and P. Pattison. Algebraic structures for dynamic networks. *Journal of Mathematical Psychology*, 57(6):310–319, 2013. `doi:10.1016/j.jmp.2013.11.002`.

[19] M. Latapy, C. Magnien, and F. Simard. Code for computing betweenness centrality in link streams. URL: `https://www-complexnetworks.lip6.fr/~latapy/btw-ls/`.

[20] M. Latapy, C. Magnien, and T. Viard. *Weighted, Bipartite, or Directed Stream Graphs for the Modeling of Temporal Networks*, pages 49–64. Springer International Publishing, Cham, 2019. `doi:10.1007/978-3-030-23495-9_3`.

[21] M. Latapy, T. Viard, and C. Magnien. Stream graphs and link streams for the modeling of interactions over time. *Soc. Netw. Anal. Min.*, 8(1):61:1–61:29, 2018. `doi:10.1007/s13278-018-0537-7`.

[22] V. Latora, V. Nicosia, and G. Russo. *Complex Networks: Principles, Methods and Applications*. Cambridge University Press, 2017. `doi:10.1017/9781316216002`.

[23] N. Masuda and R. Lambiotte. *A Guide to Temporal Networks*, volume 4 of *Series on Complexity Science*. World Scientific (Europe), sep 2016. `doi:10.1142/q0033`.

[24] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora. Graph Metrics for Temporal Networks. In P. H. Saramäki and Jari, editors, *Temporal Networks*, pages 15–40. jun 2013. `doi:10.1007/978-3-642-36461-7_2`.

[25] F. S. Pereira, S. de Amo, and J. Gama. Evolving Centralities in Temporal Graphs: A Twitter Network Analysis. In *IEEE International Conference on Mobile Data Management (MDM)*, pages 43–48. Institute of Electrical and Electronics Engineers (IEEE), aug 2016. `doi:10.1109/mdm.2016.88`.

[26] S. Praprotnik and V. Batagelj. Semirings for temporal network analysis, 2018. `arXiv:1603.08261`.

[27] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM'14, pages 413–422, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2556195.2556224`.

[28] M. Rymar, A. Molter, Hendrikand Nichterlein, and R. Niedermeier. Towards classifying the polynomial-time solvability of temporal betweenness centrality. In Ł. Kowalik, M. Pilipczuk, and P. Rzążewski, editors, *Graph-Theoretic Concepts in Computer Science*, pages 219–231, Cham, 2021. Springer International Publishing. `doi:10.1007/978-3-030-86838-3\_17`.

[29] F. Simard. On computing distances and latencies in link streams. In F. Spezzano, W. Chen, and X. Xiao, editors, *ASONAM '19: International Conference on Advances in Social Networks Analysis and Mining, Vancouver, British Columbia, Canada, 27-30 August, 2019*, pages 394–397. ACM, 2019. `doi:10.1145/3341161.3342907`.

[30] C. Stadtfeld and P. Block. Interactions, actors, and time: Dynamic network actor models for relational events. *Sociological Science*, 2017. `doi:10.15195/v4.a14`.

[31] T. Takaguchi, Y. Yano, and Y. Yoshida. Coverage centralities for temporal networks. *Physics of Condensed Matter*, 89(2), 2015. `doi:10.1140/epjb/e2016-60498-7`.

[32] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia. Analysing information flows and key mediators through temporal centrality metrics. In *Proceedings of the 3rd Workshop on Social Network Systems, SNS'10*, pages 1–6, New York, New York, USA, 2010. ACM Press. `doi:10.1145/1852658.1852661`.

[33] I. Tsalouchidou, R. Baeza-Yates, F. Bonchi, K. Liao, and T. Sellis. Temporal betweenness centrality in dynamic graphs. *International Journal of Data Science and Analytics*, 2019. `doi:10.1007/s41060-019-00189-x`.

[34] A. Tumurbaatar and M. J. Sottile. Algebraic algorithms for betweenness and percolation centrality. *Journal of Graph Algorithms and Applications (JGAA)*, 25(1):241–261, 2021. `doi:10.7155/jgaa.00558`.

[35] A. van der Grinten and H. Meyerhenke. Scaling betweenness approximation to billions of edges by mpi-based adaptive sampling. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*, pages 527–535. IEEE, 2020. `doi:10.1109/IPDPS47924.2020.00061`.

[36] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[37] J. Whitbeck, M. Dias de Amorim, V. Conan, and J.-L. Guillaume. Temporal reachability graphs. In *Proceedings of the 18th annual international conference on Mobile computing and networking - Mobicom '12*, page 377. ACM Press, 2012. `doi:10.1145/2348543.2348589`.

[38] M. J. Williams and M. Musolesi. Spatio-temporal networks: reachability, centrality and robustness. *Royal Society Open Science*, 3(6):160196, jun 2016. `doi:10.1098/rsos.160196`.

[39] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu. Path problems in temporal graphs. In *Proceedings of the VLDB Endowment*, pages 721–732, 2014. `doi:10.14778/2732939.2732945`.

[40] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke. Reachability and time-based path queries in temporal graphs. In *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, pages 145–156. Institute of Electrical and Electronics Engineers Inc., jun 2016. `doi:10.1109/ICDE.2016.7498236`.

[41] Y. Yuan, X. Lian, G. Wang, Y. Ma, and Y. Wang. Constrained shortest path query in a large time-dependent graph. *Proceedings of the VLDB Endowment*, 12(10):1058–1070, jun 2019. `doi:10.14778/3339490.3339491`.

[42] K. A. Zweig. *Network analysis literacy: a practical approach to the analysis of networks*. Lecture notes in social networks. Springer, Vienna, 2014.