Social Network Analysis and Mining (2021) 11:51
https://doi.org/10.1007/s13278-021-00759-7

ORIGINAL ARTICLE

Evaluating metrics in link streams

Frédéric Simard[1]

Received: 29 November 2019 / Revised: 13 May 2021 / Accepted: 18 May 2021

Abstract
We seek to understand the topological and temporal nature of temporal networks by computing the distances, latencies and lengths of *shortest fastest* paths. Shortest fastest paths offer interesting insights about connectivity that were unknowable until recently. Moreover, distances and latencies tend to be computed by separate algorithms. We developed four algorithms that each compute all those values efficiently as a contribution to the literature. Two of those methods compute metrics from a fixed source temporal node. The other two, as a significant contribution to the literature, compute the metrics between all pairs of source and destination temporal nodes. The methods are also grouped by whether they work on paths with *delays* or not. Proofs of correctness for our algorithms are presented as well as bounds on their temporal complexities as functions of temporal network parameters. Experimental results show the algorithms presented perform well against the state of the art and terminate in decent time on real-world datasets. One purpose of this study is to help develop algorithms to compute centrality functions on temporal networks such as the betweenness centrality and the closeness centrality.

Keywords Temporal networks · Link streams · Algorithms · Shortest paths · Fastest paths · Distances · Latencies

1 Introduction

Network science has been greatly influenced in recent years by temporal networks. Researchers in various fields have observed that real data varies over time and that static networks are insufficient to capture the full extent of some phenomenon. Different models of temporal networks have been suggested, among which the *Link Streams* of Latapy et al. (2018) that captures the network evolution in continuous time. As is the case with other forms of networks, the notions of paths and distances are fundamental to the study of link streams. Kempe et al. (2002) mention the use of time-respecting paths to study temporal networks. They further mention applications to epidemiology, in which one would seek information about the spread of a virus in a population. Human interactions can also be analyzed with temporal networks as has been observed by Tang et al. (2010a) and the link stream framework can help advance those studies by allowing edges to have durations. Although online social networks can be thought to vary in discrete time, with tweets and retweets on Twitter for example, in real social networks the interactions have durations which are important to take into account in order to have an accurate description of the data. In practice, studies have emerged from the SocioPatterns Collaboration that includes datasets on face-to-face contacts (see Cattuto et al. 2010; SocioPatterns 2021) with temporal labels. Those datasets are valuable tools to more accurately investigate aspects of social networks such as homophily (Stehlé et al. 2013) and epidemics (Moinet et al. 2018).

A short version of this text was presented at the *International Conference on Advances in Social Networks Analysis and Mining (ASONAM '19)*, in Vancouver, Canada (Simard 2019b)

Frédéric Simard
fsima063@uottawa.ca

[1] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

1.1 Shortest fastest paths

Latapy et al. developed the notion of *shortest fastest paths* in their link stream model as a type of paths that gather together the temporal as well as the structural information of a link stream. A shortest fastest path is one that is shortest among the fastest paths between two endpoints. This type of path is used to define a *betweenness* centrality and it appears other distance-based centrality functions could be so defined as well. Simard et al. (2021) provide an explicit algorithm to compute this centrality in a link stream and the algorithms in the current manuscript can help speed up this process by computing the metrics used in their method. A social network can thus be analyzed through different perspectives: using the *distance* to measure how the connectivity of a group varies over time, the *latency* to measure how quickly an information can spread into a group of people and the length of a shortest fastest path to measure how efficiently this information is relayed. Note also how the time a shortest path starts and ends influences the information it can spread. Shortest fastest paths describe a natural notion of communication *efficiency*: when a viral rumour (such as a piece of disinformation) spreads over a network it can spread *quickly* and those actors on *short* fastest paths from the source to any receiver can be considered as efficient spreaders.

Therefore, shortest and shortest fastest paths not only enable one to compute Latapy et al.'s betweenness centrality over link streams, but also quantify the reachability of different nodes over different times. Many authors noted how important this task is given it can help determine how information spreads over a social network, for example what person or group of people most likely initiated a campaign over an online social network, but also what individuals should be restricted in the event of a pandemic. The COVID-19 pandemic leads to a trove of online misinformation and there is ongoing research to study the phenomenon (see for example Cinelli et al. 2020). Similarly, epidemic spreading over temporal networks is still being investigated, for example in the recent work of Ciaperoni et al. (2020), and some authors integrate centrality functions in their studies.

1.2 Contributions and impact

Our main contributions in this work are two groups of algorithms that compute metrics of shortest (fastest) paths in a link stream. Two algorithms in the first group work from a single source, while the two in the latter work from multiple sources. The two sets of algorithms are split on whether they assume paths have strictly positive or null delays. All methods return the lengths of shortest paths, the lengths of shortest fastest paths as well as pairs of starting and arrival times of temporal (fastest) paths. Some of that information is summarized as *reachability triples* (a, b, c) (see Definition 4.1) such that for every fixed source s and node v, a is a largest starting time from s to (b, v) and c is the *distance* from (a, s) to (b, v). There are three major novel aspects in this work. First, we compute multiple metrics at once, whereas many other authors devise separate algorithms for the same tasks. Second, we compute lengths of shortest fastest paths, which is a novel metric in the literature. Finally, we present algorithms that work from multiple sources. This has not been considered recently in the literature and one has to go back to the work of Xuan et al. (2003) to find similar algorithms that work with multiple sources. In short, our major contributions are:

1. To provide four algorithms each of which compute multiple known metrics;
2. To design algorithms that compute the lengths of shortest fastest paths;
3. To design algorithms that compute metrics from multiple sources.

A major impact of this study is the ability to efficiently compute values required to compute the betweenness centrality of a temporal node in a link stream. It also makes it faster to compute shortest paths between large numbers of nodes in link streams. The novel ability to compute reachability triples for all destinations (and sources) is relevant due to the importance of shortest paths in the analysis of temporal networks. We think our algorithms are also simple yet powerful enough that they could be extended to other metrics such as arrival times of foremost paths and lengths of shortest foremost paths. This seems to hold in particular if the temporal dimension is the first argument optimized over, such as in shortest fastest paths or shortest foremost paths.

Our algorithms are evaluated on datasets taken from the KONECT library of networks (Kunegis 2019). A description of the datasets used can be found in Appendix B.

General definitions are presented in Sect. 2, followed by a state of the art in Sect. 3. Then, we present our main methods in Sect. 4, experiments in Sect. 5 and we conclude in Sect. 6.

2 Background

Most definitions are taken from Latapy et al. (2018). A link stream L is a tuple $L = (T, V, E)$ where $T \subseteq \mathbb{R}$ is a set of time instants, V is a finite set of nodes (vertices) and $E \subseteq T \times V \otimes V$ is a set of links (edges). Here, $V \otimes V$ denotes the set of unordered pairs of vertices and we write $uv \in V \otimes V$. We say an element $(t_v, v) \in T \times V$ is a temporal vertex.

An edge of E is a tuple (t, uv). Given an interval $I \subseteq T$, we write $(I, uv) \subseteq E$, instead of $I \times \{uv\} \subseteq E$, to mean all edges (t, uv) such that $t \in I$ are in E. We say an edge $(I, uv) \subseteq E$ is maximal if there exists no other edge $(J, uv) \subseteq E$ such that $I \subset J$. We say a maximal edge $([a, b], uv) \subseteq E$ starts at a, ends on b and has duration $b - a$. Given the simple edge (a, uv), we write $\mathrm{dur}(a, uv) = b - a$ for the duration of the edge $([a, b], uv)$. We let Ω be the set of *event times* of T, that is $\Omega := \{t \in T; \exists \text{ maximal edge } ([t, t'], uv) \subseteq E \text{ or } ([t', t], uv) \subseteq E\}$. Elements of $\Omega \times V$ are called *event nodes*. We write $E_\Omega := \{(t, uv) \in E; t \in \Omega\}$.

A maximal edge, as well as Ω and $\Omega \times V$ are illustrated on the link stream of Fig. 1. On this link stream, $([1, 2], cb) \subset E$ is a maximal edge, whereas $([1, 1.5], cb) \subset E$ is not. Thus, $\Omega = \{0, 1, 2, 3\}$.

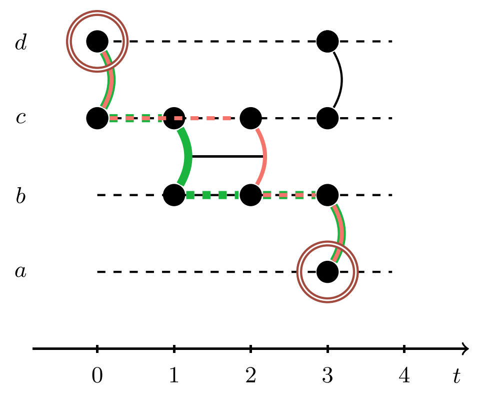


Fig. 1 A simple link stream with maximal edge ([1, 2], *cb*)

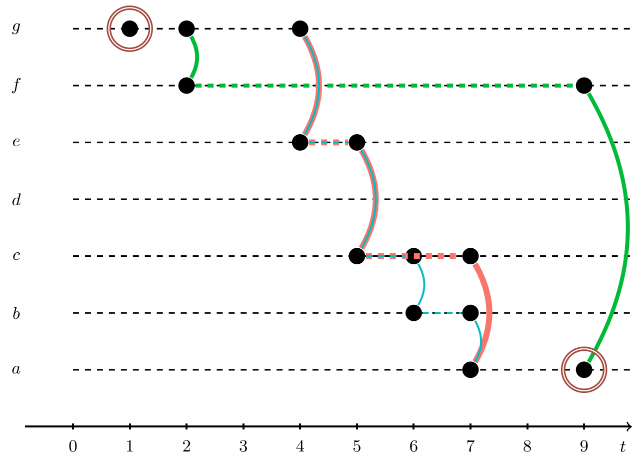


Fig. 2 The shortest path from (1, *g*) to (9, *a*) (both encircled) is drawn in green. The two fastest paths are drawn in red and in blue. The sole shortest fastest path is the red one. Observe that, $d_f((1,g),(9,a)) = 3 > d_f((1,g),(9,f)) + d_f((9,f),(9,a)) = 2$ (color figure online)

The time-induced graph G_t induced by a time $t \in T$ is defined as $G_t = (V, \{uv;(t,uv) \in E\})$. In a link stream L, a path P from $(\alpha, u) \in T \times V$ to $(\omega, v) \in T \times V$ is a sequence $(t_0, u_0, v_0), (t_1, u_1, v_1), \ldots, (t_k, u_k, v_k)$ of elements of $T \times V \times V$ such that $u_0 = u$, $v_k = v$, $t_0 \geq \alpha$, $t_k \leq \omega$ and for all i, $t_i \leq t_{i+1}$, $v_i = u_{i+1}$ and $(t_i, u_i v_i) \in E$. We say that such a path starts at t_0, arrives at t_k, has length $k+1$ and duration $t_k - t_0$. We write $(\alpha, u) \rightsquigarrow (\omega, v)$ to mean that there exists a path from (α, u) to (ω, v) and say (ω, v) is reachable from (α, u). We also call t_0 a starting time and t_k an arrival time from (α, u) to (ω, v). Each path between two fixed temporal nodes (α, u) and (ω, v) defines a pair of starting time and associated arrival time. On the link stream of Fig. 1, two paths are illustrated: the green one $P_1 = (0, d, c), (1, c, b), (3, b, a)$ and the red one $P_2 = (0, d, c), (2, c, b), (3, b, a)$. Both have the same starting and arrival times from (0, *d*) to (3, *a*), namely times 0 and 3. Both paths are fastest. We can also say s is a starting time from a temporal node $(\alpha, u) \in T \times V$ to a node $v \in V$, in which case there exists some time $t \in T$ such that s is the starting time of a path from (α, u) to (t, v). Same goes for the arrival times.

We say a path P is shortest if it has minimal length and call its length the *distance* from (α, u) to (ω, v), written $d((\alpha, u), (\omega, v))$. Similarly, P is fastest if it has minimal duration, in which case this duration is called the *latency* from (α, u) to (ω, v) and is written $l((\alpha, u), (\omega, v))$. Note that if $(\alpha, u) \rightsquigarrow (\omega, v)$, there exists at least one pair of starting time and arrival time (s, a) such that $l((\alpha, u), (\omega, v)) = a - s$. Then, we say s is a latest starting time and a an earliest arrival time. Finally, P is called shortest fastest if it has minimal length among the set of fastest paths from (α, u) to (ω, v). We call its length the *sf-metric* from (α, u) to (ω, v) and write it $d_f((\alpha, u), (\omega, v))$. In general, this is not a distance as it does not respect the triangular inequality and is only a premetric, a simple counterexample is shown in Fig. 2. On the same figure are drawn a shortest path, two fastest paths and a unique shortest fastest path.

3 Related work

This work is related to the study of Wu et al. (2014) and our algorithms can be applied in the same contexts as their shortest and fastest paths methods. The main contribution of the present work is to compute *sf*-metrics, as well as distances and latencies, in a single pass over a dataset. Separately, Wu et al.'s fastest and shortest paths methods are insufficient to compute centralities such as the betweenness of a link stream, while an algorithm combining them to produce *sf*-metrics is not efficient because it requires iterating multiple times over the dataset. Meanwhile, our methods iterate only once over the dataset to produce the three metrics and are suitable for studying different aspects of a link stream. We also output information on the starting and arrival times of shortest (fastest) paths. This study was instigated as a first step in computing Latapy et al.'s betweenness centrality defined in Latapy et al. (2018) and computed in Simard et al. (2021).

More recently, Himmel et al. (2019) devised a generic algorithm to compute optimal paths in temporal graphs from one source node to all other destinations. Their algorithm is generic in the sense that it can compute different types of optimal paths such as shortest, fastest and foremost. Their main algorithm computes those paths separately. They do consider a method to combine some optimization criteria, such as fastest and shortest; however, this is done through a linear combination of optimization objectives. This is in contrast to the present work, which is focused on a bilevel optimization approach (see Colson et al. 2007): the criterion that a path be fastest can never be violated, at the possible expense of paths not being shortest. We also present algorithms from multiple sources, which is not the case in their work. Brunelli et al.

(2021) took a similar approach to Himmel et al. and devised a generic algorithm to compute Pareto optimal paths to generalize multiple criteria (shortest paths, foremost paths, etc.). Again, their method only works from a single source. Moreover, they present the temporal complexity of their method but do not derive the explicit complexities when their method is applied to specific problems. In particular, it is not clear what complexity their method would achieve on the task of computing lengths of shortest fastest paths and how a concrete implementation would fare against our programs. In 2019, Li et al. (2019) improved on the work of Wu et al. by using dynamic programming approaches to compute shortest paths, fastest paths and (restricted) earliest-arrival paths. Although interesting, this work comes with the same restrictions that we observed in the work of Wu et al.

Furthermore, this work is also close to Tang et al. (2010b) since these authors define a *betweenness* centrality on temporal networks in terms of fastest shortest paths. Whether to use fastest shortest or shortest fastest paths (or any other type of path that combines temporal and structural information) depends on what information one wants to emphasize and on the context of the study. Shortest and fastest paths were also studied by Xuan et al. (2003) and we were inspired by their all-pairs fastest path method to develop Algorithm 2 and 4. The latter are relevant to compute some centralities because metrics between all pairs of (temporal) nodes may be required. To our knowledge, Xuan et al.'s method is the only of its kind to return latencies between all pairs of nodes. More recently, Casteigts et al. (2015) adopted the same strategies as Xuan et al. for computing shortest and fastest paths in a distributed way.

Casteigts et al. (2012) also offer a survey of temporal networks that includes many applications of shortest and fastest paths. In particular, such paths can be used to study the reachability of a temporal node from another. It appears from that survey that either the distance or the latency is often used as a temporal metric to evaluate how well a temporal node can communicate with another. In this regard, the *sf*-metric can be used as another temporal function since it combines the temporal as well as the structural information into a single map. Note that the notion of *foremost* paths (or journeys) is also used by some authors (such as Casteigts et al. 2015) to study temporal reachability. A foremost path only has minimal arrival time, while its starting time is unconstrained. This type of path is also useful in many studies and we expect that our algorithms can be extended to those cases to output lengths of *shortest foremost* paths. Finally, Casteigts et al. (2020) and Thejaswi and Gionis (2020) studied another type of temporal paths called *restless*, such that one cannot wait more that a prescribed amount of time on each node. This is an interesting constraint that we have not considered. However, we expect that our methods can also be extended to fit these constraints.

Finally, observe that the link stream framework is also close to the Time-Varying Graphs framework of Casteigts et al. (2012). Thus, all results presented in this paper carry to this other framework as well. We did not intend to survey the different models of temporal graphs present in the literature, as this is out of the scope of this work. To the best of our knowledge, there are two major models that allow edges to have duration and allow time to flow continuously: the Link Stream of Latapy et al. and the Time-Varying Graph of Casteigts et al. Other models, such as temporal networks (see Holme and Saramäki 2012), evolving graphs (Ferreira 2004) and temporal-event graphs (Mellor 2017) mostly focus either on discrete timesteps or on zero transmission delay. Our algorithms also work on those models. The time-dependent network model mentioned by Brunelli et al. (2021) is slightly more general than the link stream we use since it allows the transmission delay over the edges to follow a non-constant function.

4 Multiple-targets shortest fastest paths algorithms

The full implementations of the algorithms presented here, in C++, can be found online (Simard 2019b).

We present here two main methods, Algorithms 1 and 2 that compute the distances, latencies and *sf*-metrics from one source event node to all other event nodes. Algorithm 2 builds on the first method to compute those values for all pairs of event nodes. Section 4.4 also presents Algorithm 3 that was derived from Algorithm 1. This new method works with positive *delays*, a case that is often considered in the literature. Algorithm 4 also works with positive delays and is derived from Algorithm 2. We focus on the first two algorithms.

We present some small results that lead the way to those algorithms. The strategy for all methods is the same: we compute the distances from any temporal node (s_v, u) to (t_v, v) such that s_v is the *largest* (or maximal) starting time from any (t_u, u) to (t_v, v). If it happens that $t_v - s_v = l\big((s_v, u), (t_v, v)\big)$, then this distance is the *sf*-metric from the former to the latter temporal node. Otherwise, since we iterate chronologically over Ω, this latency must have been computed at a time earlier than t_v and is saved in memory.

All algorithms described in this section are presented in Appendix A in order not to disrupt the reading.

4.1 Two simple lemmas

The algorithms we present compute what we call *reachability triples* that contain information about the lengths of shortest paths from one temporal node to another as well as the starting and arrival times of those paths.

Definition 4.1 (*Reachability triples*) Let (t_s, s) be an event node. If there exists a shortest path of length l from (t_s, s) to the event node (t_y, y) that starts on a largest starting time $t \in \Omega$, then we say (t, t_y, l) is a reachability triple from (t_s, s) to y.

In the following, we write R_v for the dictionary of reachability triples from a fixed source event node to any node v. In order to reduce the cost of operations in R_v, we assume this dictionary is implemented in such a way that R_v holds keys s_v and $\mathrm{R}_v[s_v]$ holds pairs (a_v, d_v) that form reachability triples (s_v, a_v, d_v). This makes it easier to search for starting times s_v.

All algorithms compute distances *from largest starting times only*. Those distances are contained in dictionaries R_v for each $v \in V$ as part of reachability triples. Note that if a link stream reduces to a network, that is if the set of time instants T is a singleton, then each R_v will contain the usual distances from a fixed source to v. The temporal nature of a link stream forces us to take starting and arrival times into account when looking for shortest paths. Moreover, reachability triples could also be defined without the constraint that starting times are largest; however, the algorithms would not be as efficient because the dictionaries would grow larger with $|\Omega|$.

Lemma 4.2, due to Wu et al. (2014), states that shortest paths are prefix-shortest. We say a path $P_{(t_s,s)(t_u,u)}$ from a temporal node (t_s, s) to another temporal node (t_u, u) is a prefix of another path $P_{(t_s,s)(t_v,v)}$ from the same source to temporal node (t_v, v) if $P_{(t_s,s)(t_u,u)}$ is a subsequence of $P_{(t_s,s)(t_v,v)}$.

Lemma 4.2 *Let $P_{(t_s,s)(t_v,v)}$ be a shortest path from a temporal node (t_s, s) to another (t_v, v). Then, every prefix $P_{(t_s,s)(t_u,u)}$ of $P_{(t_s,s)(t_v,v)}$ is a shortest path from (t_s, s) to (t_u, u).*

Let (t_s, s) and (t, v) be two temporal nodes. We define the outer distance from (t_s, s) to (t, v), $d((t_s, s), (t^-, v))$, as either $\lim_{t_0 \to t^-} d((t_s, s), (t_0, v))$, when $t > t_s$, or $d((t_s, s), (t, v))$, when $t_s = t$. Lemma 4.3 suggests it suffices to compute distances in induced graphs G_t for any time t to deduce the distances between two temporal nodes.

Lemma 4.3 *Let (t_s, s) be a source temporal node and (t_y, y) be a temporal node reachable from the source by a non-empty shortest path. Then, there exists $t_s \le t \le t_y$ and a connected component C of G_t such that*

$$d((t_s, s), (t_y, y)) = \min_{u,v \in C} d((t_s, s), (t^-, u)) + d((t, u), (t, v)) + d((t, v), (t_y, y)). \tag{1}$$

Proof Let $P = (t_1, u_1, u_2), \dots, (t_n, u_n, u_{n+1})$ be a non-empty shortest path from (t_s, s) to (t_y, y). Then, $t_y \ge t_n \ge t_1 \ge t_s$ and $u_1 = s, u_{n+1} = y$. There exist non-empty subpaths in P of the form $(t_j, u_j, u_{j+1}), \dots, (t_j, u_k, u_{k+1})$. Let $Q = (t_j, u_j, u_{j+1}), \dots, (t_j, u_k, u_{k+1})$ be such a subpath with the largest number of elements. By Lemma 4.2, the prefix of P from (t_s, s) to (t_j^-, u_j) is shortest and its length is $d((t_s, s), (t_j^-, u_j))$. Moreover, the subpath of P from (t_j, u_{k+1}) to (t_y, y) must also be shortest with length $d((t_j, u_{k+1}), (t_y, y))$. Finally, since P is shortest and the two subpaths formed by $P \setminus Q$ are shortest, Q must also be a shortest path. Then, Q has length $d((t_j, u_j), (t_j, u_{k+1}))$. The result follows by letting $t = t_j$ and $C = \{u_j, u_{j+1}, \dots, u_{k+1}\}$ be a connected component of G_t. □

4.2 A single-source method

In this section, we present Algorithm 1 that computes the distances (from largest starting times), latencies and *sf*-metrics from a source event node (t_s, s) to all other reachable event nodes. This algorithm mixes iterations on the induced graphs G_t for each time $t \in \Omega$ with an all-pairs distances method on their connected components. Recall that if s^* is the largest starting time from the source (t_s, s) to some temporal node (t, v), then either $t - s^* = l\big((t_s, s), (t, v)\big)$ or not. If so, then $d((t_s, s), (t, v))$ is the *sf*-metric from (t_s, s) to (t, v). This length is computed with Lemma 4.3 by using the outer distances saved in memory as well as the all-pairs distance method on G_t. Thus, when we iterate over all pairs (s_v, d_v) of starting time and outer distance from the source to (t, v), we can deduce the duration and the length of the shortest fastest paths from the source to (t, v). This method uses a set D that is assumed sorted in lexicographic order. Sorting D helps lower the temporal complexity, but is not fundamental to understand the algorithm.

Remark 4.4 In all algorithms, we assume the dictionaries use self-balanced binary trees in order to obtain logarithmic worst-case complexities with simple structures. In our implementations, we used hash tables and heaps (see Remark 4.7) to lower the running times.

Furthermore, all operations on dictionaries such as min and max are well-defined whenever the relevant keys are present in the dictionaries. We assume the absence of a key is properly handled.

Before proving that Algorithm 1 is correct, let us go through a small example in order to build intuition. Other algorithms are highly similar.

Example 4.5 Consider again the link stream of Fig. 2. Suppose the source is again $(1, g)$, $t = 7$ and $C = \{a, b, c\}$. Thus, Algorithm 1 will look for shortest (fastest) paths that can reach temporal nodes $(7, a)$, $(7, b)$ and $(7, c)$. The unique largest starting time from the source to C at time 7 is $s_v = 4$. This time is given by the greatest key in R_u for any $u \in C$. Then, we iterate over the outer distances from $(4, g)$ to $(7, v)$ for each

$v \in C$. Note how the time of the source has changed from 1 to 4. By definition, and since the link stream is discrete, outer distances are given as the distances from (4, g) to (6, v) for each $v \in C$. Thus, we find outer distances 2 from (4, g) to (7, c) and 3 from (4, g) to (7, b). Node a is discovered at time 7 and its outer distance does not exist before that. Finally, combining the outer distances with the distances inside the graph induced by C at time 7, we find that the distance from (4, g) to (7, c) is 2, 3 from (4, g) to (7, b) and also 3 from (4, g) to (7, a). This last distance is given by the combination between the outer distance from (4, g) to (7, c) and the distance in C from (7, c) to (7, a). Since node a is discovered first at time 7, that is its first arrival time from (1, g) is 7, then the latency from (1, g) to (7, a) is $l((1, g), (7, a)) = 7 - 4 = 3$ and the distance from (1, g) to (7, a) is the *sf*-metric from the former to the latter.

Proposition 4.6 *Algorithm* 1 *correctly computes the latencies and sf-metrics from a source event node to all other reachable event nodes as well as the set of dictionaries* $\{\mathrm{R}_v; v \in V\}$. *It requires at most* $O\big(|V|^2|\Omega| \log |\Omega| + |V|^3|\Omega|\big)$ *operations in the worst case.*

Proof of correctness Let $(t_v, v) \in \Omega \times V$ be some reachable destination. Let us show by induction on $\Delta := |\{t_0 \in \Omega; t_v \geq t_0 \geq t_s\}|$ that $d[(t_v, v)] = d_f\big((t_s, s), (t_v, v)\big)$, $f[(t_v, v)] = l\big((t_s, s), (t_v, v)\big)$ and R_v is correct up to time t_v.

- When $\Delta = 1$, we iterate only on time t_s and the result is clear.
- Suppose the result holds for all $k < \Delta$. Let $(t_1, \dots, t_{\Delta-1})$ be the times previously iterated over on Line 3 and t_Δ the current time. By the induction hypothesis, by time $t_{\Delta-1}$, all values of R_w, for all $w \in V$, are correctly updated. Let C_v be the connected component of G_{t_Δ} containing v. If $s \in C_v$, then the result follows as in the case with $\Delta = 1$. Then, suppose $s \notin C_v$. Since each R_v is correctly updated up to time $t_{\Delta-1}$ for each reachable $v \in V$, D contains triples $(-s_w, d_w, w)$ for each $w \in C_v$ that have been visited prior to $t_{\Delta-1}$ from the source from a starting time s_w. The set D contains the largest starting time s_w from the source to (t_Δ, w). Then, either $t_\Delta + s_w = l\big((t_s, s), (t_\Delta, w)\big)$ or this latency is given by some $f[(t_0, w)]$ such that $t_0 < t_\Delta$. Let's iterate on $(-s_w, d_w, w)$.

  By Lemma 4.3, there exists a time $-s_w \leq t_i \leq t_\Delta$ and a connected component C_i of G_{t_i} such that $d((-s_w, s), (t_\Delta, w)) = \min_{x,y \in C_i} d((-s_w, s), (t_i^-, x))$ $+ d((t_i, x), (t_i, y)) + d((t_i, y), (t_\Delta, w))$. The sequence of distances

  $$d((-s_w, s), (-s_w, u)), \dots, d((-s_w, s), (t_\Delta, u))$$

  is non-increasing for each $u \in V$ because each element is minimal. Thus, since $w \in C_v$, in particular this lemma holds with $t_i = t_\Delta$ and $C_i = C_v$. Then,

  $$\begin{aligned} d((-s_w, s), (t_\Delta, w)) = & \min_{x,y \in C_v} d((-s_w, s), (t_\Delta^-, x)) \\ & + d((t_\Delta, x), (t_\Delta, y)) \\ & + d((t_\Delta, y), (t_\Delta, w)) \\ = & \min_{x \in C_v} d((-s_w, s), (t_{\Delta-1}, x)) \\ & + d((t_\Delta, x), (t_\Delta, w)). \end{aligned}$$

  By the induction hypothesis, the outer distance $d_x = d((-s_w, s), (t_{\Delta-1}, x))$ can be recovered from $(-s_w, t_{\Delta-1}, d_x) \in \mathrm{R}_x$ for each $x \in C_v$. Then, using d_x and the dictionary d' returned by the all-pairs distances algorithm on Line 6, the expression above reduces to $d((-s_w, s), (t_\Delta, w)) = \min_{x \in C_v} d_x + d'[(x, w)]$. In the last equation, the intermediary node $x \in C_v$ over which the minimum is taken is irrelevant. If $y \in U_x$, then the distance from the source to y is the same as the distance from the source to x. Thus, it holds that:

  $$\begin{aligned} d((-s_w, s), (t_\Delta, w)) &= \min_{x \in C_v} d_x + d'[(x, w)] \\ &= \min_{x \in C_v} \min_{y \in U_x} d_y + d'[(y, w)] \\ &= \min_{x \in C_v} d_x + \min_{y \in U_x} d'[(y, w)]. \end{aligned}$$

  Thus, when we iterate on the element $(-s_w, d_w, w)$ from D, we construct the set U_w of nodes at distance d_w from $(-s_w, s)$ at time t_Δ. The last equation is thus used to insert into R_w the right triple $(-s_w, t_\Delta, d((-s_w, s), (t_\Delta, w)))$ for each $w \in C_v$. When we have iterated over all of D, all dictionaries R_v are correct at time t_Δ. Finally, it suffices to observe that once $f[(t_\Delta, w)]$ is updated with its final value, then by definition the update of $d[(t_\Delta, w)]$ on Line 24 yields the *sf*-metric from (t_s, s) to (t_Δ, w) for each w.

□

Proof of complexity Let us write $n := |V|, m_t := |E_t|$ and $\omega := |\Omega|$. On each time $t \in \{t_0 \in \Omega; t_0 \geq t_s\}$, we first look up the connected components of G_t, which requires at most $O\big(n + m_t\big)$ operations. On each component C of G_t, we run an all-pairs distances method, which makes at most $O\big(n^2 + |C|m_t\big)$ operations. Observe that the $O\big(n^2\big)$ factor is for the initialization of a $V \times V$ array, which can be done once for the whole link stream. In order to see if a node has been reached at time t, it suffices at all previous times i to write distance d_{uv} in $d[u, v]$ as (i, d_{uv}).

For each node $v \in V$, the list in $\mathrm{R}_v[-s_u]$ contains at most ω elements since there can be at most as many pairs in $\mathrm{R}_v[-s_u]$ as there are arrival times on v. The same goes for the number of keys s_v in R_v.

We only need one distance in $\mathrm{R}_v[-s_u]$, thus one distance per node, and D can be constructed with at most $O(|C|)$ operations for all $v \in C$. Inserting and removing an element from $\mathrm{R}_v[-s_u]$ takes at most $O(\log \omega)$ operations. The costliest operation is finding the value of $d_{\min}$, which involves searching in the set U of size $|C|$. This value is recomputed for every source $u \in C$, destination $w \in C$ and the set $U \subseteq C$ changes with every iteration. The **for** loop on Line 14 will make at most $O\big(|C|^2(|C| + \log|\omega|)\big)$.

The **for** loop over G_t will make at most

$$O\big(n^2\big) + \sum_{C \subseteq V} O\big(|C|m_t + |C|^2(|C| + \log \omega)\big) \leq O\big(nm_t + n^2 \log \omega + n^3\big).$$

Recall that $\sum_{t\in\Omega} m_t = |E_\Omega|$. Summing over all times in Ω, we deduce a complexity of $O\big(nm_\Omega + n^2\omega \log \omega + n^3\omega\big)$. The final complexity follows from the fact that $E_\Omega \subseteq \Omega \times V \otimes V$. □

Computing the minimum distance between all pairs of nodes in a connected component C, while accounting for the outer distances is tricky and leads to the factor of $|V|^3$. We do not expect this can be much reduced. Indeed, in the worst case all nodes of a time-induced graph G_t are reachable from the source from different starting times and are in the same connected component. Thus, $|V|$ distances would have to be updated. Following Eq. (1), in order to update the distance from the source to any node w, we need to look at the distances from the source to any reachable node of G_t as well as the distance from those nodes to w. In the worst case, this amounts to running a single-source shortest path method on G_t from w. Doing this $|V|$ times on each induced graphs leads to a lower bound of $\Omega\big(|V||E_\Omega|\big)$.

Remark 4.7 Dictionary R_v might hold multiple triples (s, a, d) with the same starting time s, that is $\mathrm{R}_v[s]$ can grow linearly with $|\Omega|$. This is the simplest implementation of this dictionary that contains exactly all information. Since we only query for the largest starting time and the smallest distance (given that starting time), a more convenient way to implement R_v is with max and min heaps. Thus, R_v can be a max-heap while $\mathrm{R}_v[s_v]$, for any key s_v, would be a min-heap of elements $(d_v, [a_v^1, a_v^2])$, sorted on the distance d_v. Here, a_v^1 is the first arrival time on v from (s_v, s) with distance d_v, while a_v^2 is the last one. This defines the interval $[a_v^1, a_v^2]$ during which shortest paths of length d_v are known which summarizes with one pair $|\Omega|$ amount of information. This would remove the logarithmic factor from the complexity of Algorithm 1, which in practice leads to significant improvement. However, since the data structure is different, accessors to R_v would have to be modified. Thus, if one specifically needs to know exactly when a path of length d_v arrives on v from (s_v, s), then this structure would be inadequate.

Corollary 4.8 *Algorithm* 1 *can be implemented with heaps to make at most* $O\big(|V|^3|\Omega|\big)$ *operations.*

Proof This follows from Proposition 4.6 and Remark 4.7. □

We use the sets V, E_Ω and Ω as parameters to evaluate the temporal complexities of our algorithms. These appear as natural choices since Ω indicates how the temporal dimension affects the number of operations while E_Ω is a surrogate for E, which is in general uncountable.

4.3 A multiple-sources *sf*-metrics method

Suppose Ω is finite and starts on some time a. Algorithm 2 returns a set of dictionaries of *sf*-metrics D_{uv} for each pair of nodes $(u, v) \in V^2$ of dictionary $D_{uv}[s_{uv}] = (a_{uv}, d_{uv})$ such that $l\big((s_{uv}, u), (a_{uv}, v)\big) = a_{uv} - s_{uv}$ and $d((s_{uv}, u), (a_{uv}, v)) = d_{uv}$. During its execution, it updates a dictionary D^0 such that $D_{uv}[t] = (a_{uv}, d_{uv})$, $t \in \mathrm{R}_v$ and $(a_{uv}, d_{uv}) \in \mathrm{R}_v[t]$ from $(a, u) \in T \times V$. This dictionary helps in computing D and in constructing R_v from any source. It also returns a set of dictionaries F_{uv} of latencies.

Proposition 4.9 *Algorithm* 2 *returns the latencies, sf-metrics and dictionaries* R_v *between all pairs of nodes in at most* $O\big(|V|^3|\Omega| \log|\Omega|\big)$ *operations.*

Proof of correctness Let us show that $D^0[u, v][t_v]$ holds correct reachability triples from (a, u) to (t_v, v) for any two nodes u, v and time t_v. Thus, let us fix those three variables. Let us show this by induction on $\Delta := |\{t \in \Omega; a \leq t \leq t_v\}|$.

- If $\Delta = 1$, then either u and v are in the same connected component C of G_{t_v} or not. This part is clear.
- Suppose the result holds for any $k < \Delta$. Let $(t_1, \dots, t_{\Delta-1})$ be the sequence of times previously iterated over. Let C_v be the connected component containing v at time t_Δ. If $u \in C_v$, then we argue as in the first case and the result follows. Otherwise, by the induction hypothesis, there must exist a largest starting time s_v from u to (t_Δ, v) that can be found in $SA[u, w][t_{\Delta-1}]$, for some $w \in C_v$ since each such node w is connected to v. Observe that $SA[u, v][t_{\Delta-1}]$ contains pairs of largest starting time and arrival time from u to $(t_{\Delta-1}, v)$. Observe also that t_Δ is again an arrival time on v. Thus, it suffices to compute the distance from (s_v, u) to (t_Δ, v) to obtain a reachability triple (s_v, t_Δ, d_v) from (a, u) to v. We argue as in the proof of Algorithm 1 that Algorithm 2 returns this distance d_v. The update $D[u, v][t_\Delta][s^*] \leftarrow d^*$ again follows the same reasoning as before.

□

Proof of complexity Again, let $n := |V|, m_t := |E_t|$ and $\omega := |\Omega|$. The costliest operations occur in the **for** loop starting on Line 11. There are at most ω keys on each SA_{uv}, for any $u, v \in V$. For any $t \in \Omega$ and $u, v \in V$, the size of $SA_{uv}[t]$ is upper-bounded by ω since the starting time is maximal. Thus, at most $O(\log \omega)$ operations are required. Finding the largest starting time s_v requires in the worst case $O\big(|C_v| \log \omega\big)$ operations.

Dictionary $D^0_{uv}[t]$, for any $u, v \in V$ and $t \in \Omega$, has a size at most ω^2, thus the loop over C_v (on Line 16) to find $d_{\min}$ requires at most $O\big(|C_v| \log \omega\big)$ operations.

Meanwhile, inserting into SA_{uv} takes at most $O(\log \omega)$ operations. The **for** loop on Line 11 thus makes at most:

$$\sum_{u\in C}\sum_{v\in V\setminus C} O\big(|C_v|\log\omega\big) \le \sum_{u\in C}\sum_{v\in V} O\big(|C_v|\log\omega\big) \le \sum_{u\in C} O\big(n^2\log\omega\big)$$

operations. This loop is itself repeated for all connected components $C \subseteq V(G_t)$, which in turn yields:

$$\sum_{C\subseteq V}\sum_{u\in C} O\big(n^2\log\omega\big) = \sum_{u\in V} O\big(n^2\log\omega\big)$$

operations. Thus, this method should make at most $O\big(n^2 + nm_t\big) + O\big(n^3 \log \omega\big)$ operations in the worst case on each time t. This number of operations is repeated at most ω times. Observe again that $O\big(nm_\Omega\big) \subset O\big(n^3\omega\big)$. □

Observe that Algorithm 1 needs to be called $|V|$ times in order to deduce the lengths of all shortest fastest paths from any source to any destination, since it discovers all starting times from each source. The multiple-sources algorithm is empirically faster when the desired output is the set of *sf*-metrics from all sources to all destinations. The temporal complexities of both methods are affected mostly by the induced graphs G_t. In Sect. 4.4, we will see that complexities decrease drastically on cases such as γ-paths with $\gamma > 0$ since we can remove the dependency on those time-induced graphs. The complexity could also be reduced slightly with the use of heaps; however, this would make accessing information more difficult.

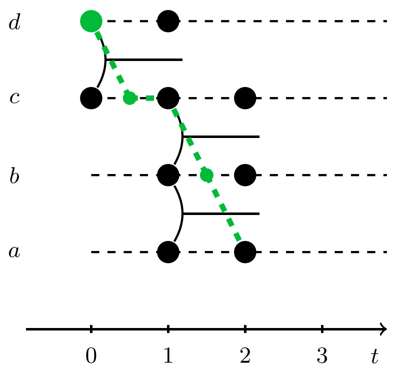


Fig. 3 A γ-path (in green) with $\gamma = \frac{1}{2}$ from (0, d) to (2, a) (color figure online)

4.4 Shortest paths with delays

In the literature on temporal walks, it is commonly assumed that a path has a transmission delay γ and we call this a γ-path. This is the case with Wu et al. whose algorithm we wish to compare Algorithm 1 against since their shortest path procedure is the most efficient known in temporal networks. Transmission delays are natural when modeling the spread of information and the time required for spreading is commensurable with the time interval during which the network is observed.

A γ-path in a link stream is a path $(t_1, u_1, u_2), \dots, (t_n, u_n, u_{n+1})$ such that $t_i \geq t_{i-1} + \gamma$ for all $1 < i \leq n$ and some $\gamma \in \mathbb{R}_+$.[1] We call γ the *delay* and note that the usual path corresponds to a 0-path. An example γ-path is shown in Fig. 3 from (0, d) to (2, a) with $\gamma = \frac{1}{2}$. Note that with this choice of γ, the subpath from (1, c) to (2, a) is the unique path from the former to the latter temporal node. When $\gamma > 0$, it is not necessary to iterate over connected components, since all nodes of a component do not communicate with each other, and we can simplify Algorithm 1 and 2 in order to reduce their numbers of operations. Thus, we present Algorithm 3 and 4 that are deduced from Algorithm 1 and 2 and assume $\gamma > 0$. Their correctness and temporal complexities follow from the same arguments used in Proposition 4.6 and 4.9.

Observe that since paths have delays, we must ensure an edge appears long enough to be crossed given that delay.

[1] More generally, we could let $\gamma : E \to \mathbb{R}_+$ be a function of the edge to traverse. This was not considered here, but our methods could be extended for this case.

Proposition 4.10 *When $\gamma > 0$, Algorithm 3 computes the latencies and sf-metrics from a source event node to all reachable event nodes as well as the set of dictionaries* R_v, *for all* $v \in V$, *in at most* $O(|V| + |E_\Omega| \log |\Omega|)$ *operations.*

Proof Correctness follows from the same reasoning as in Proposition 4.6. For the complexity, we first initialize a dictionary of $|V|$ elements, then we iterate over all edges of E_Ω and perform logarithmic searches on lists that grow with $|\Omega|$.

Note that, by the same argument as for Algorithm 1, the complexity can be lowered to $O(|V| + |E_\Omega|)$ by using a combination of max and min heaps.

Finally, in Algorithm 3, the dictionaries d and f are implemented such that the keys are nodes and values are pairs (t, k) such that t is the time value k is computed at that node. For example, if $(t, f_v) \in f[v]$, then the latency from the source to (t, v) is f_v. This enables us to sort dictionaries by time.

We also extended Algorithm 2 to the case of $\gamma > 0$ and devised Algorithm 4. This method is also guaranteed by the proof of Algorithm 2.

Proposition 4.11 *When $\gamma > 0$, Algorithm 4 returns the latencies, sf-metrics and dictionaries* R_v *between all pairs of nodes in at most* $O(|V|^2 + |V||E_\Omega| \log |\Omega|)$ *operations.*

Proof Correctness follows from Proposition 4.9. For complexity, initialization alone of the dictionaries requires $O(|V|^2)$ operations. Let (t, uv) be an edge of E_Ω. Then, searching in the dictionaries for each values takes at most $O(\log |\Omega|)$ operations when they are implemented as balanced trees. This is repeated for all nodes $w \in V \setminus \{v\}$ and all edges $(t, uv) \in E_\Omega$.

5 Experiments

We present some experiments to highlight the running times of our algorithms. In the first one, we compare Algorithm 3 with the single-source shortest path method from Wu et al. Algorithm 3 (SSMD$_\gamma$) acts as a surrogate for Algorithm 1 (SSMD) since Wu et al. designed their method to work on paths with strictly positive delays. Moreover, these authors evaluated their method from a small set of source nodes on large datasets and we follow the same procedure. In a second experiment, we compared the running times of our two methods for null delays on synthetic link streams. We also compared Algorithm 2 (MSMD) and Algorithm 4 (MSMD$_\gamma$) on synthetic link streams. Finally, we compared Algorithm 3 and 4 on the same task on some datasets.

Algorithm 2 was inspired by Xuan et al.'s fastest paths method that does not return distances. Comparing the two methods would be unfair against ours.

Remark 5.1 *(Experimental setup)* All experiments were run on a single machine with 2.6 GHz Intel Core i7 processor and 16 Gb of RAM. All methods were implemented in C++ with standard libraries, including Wu et al.'s method. We implemented standard approaches to compute connected components and all pairs distances in graphs. The full code can be found online (Simard 2019b).

5.1 Runtime comparison with the literature

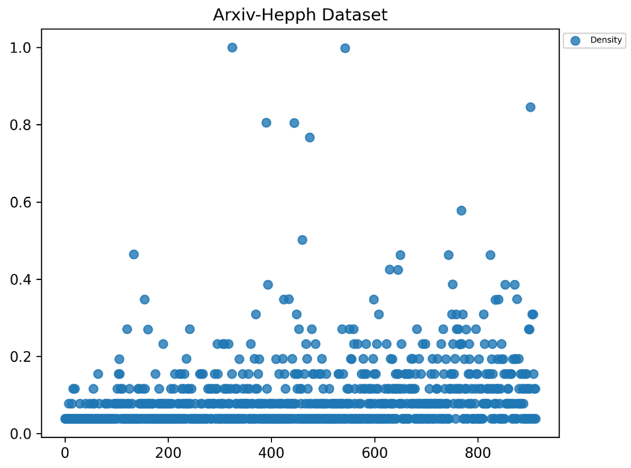
Let us compare how Algorithm 3 (SSMD$_\gamma$) fares against Wu et al.'s shortest path algorithm. This is our comparison with the literature.

Wu et al. analyzed their method with the framework of temporal graphs. We translate their temporal complexity with link stream parameters, upper bounding M with $|E_\Omega|$ and $d_{\max}$ with $|\Omega|$. Thus, the shortest path algorithm of Wu et al. makes at most $O(|V| + |E_\Omega| \log |\Omega|)$ operations in the worst case, which is the same temporal complexity as SSMD$_\gamma$.

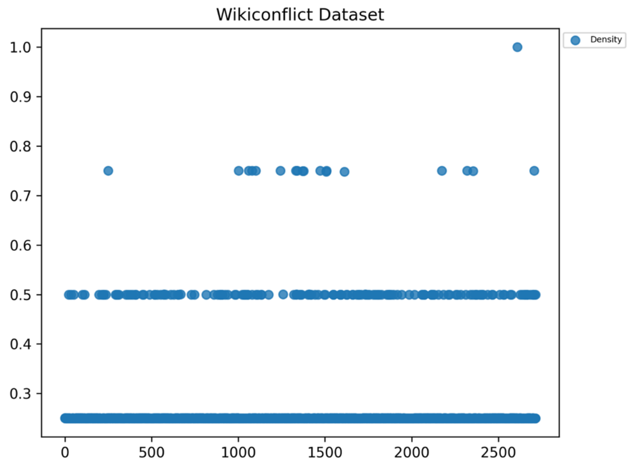
We ran experiments on link streams of various sizes, as measured with $|V|$, $|\Omega|$ and $|E_\Omega|$. We used the same datasets as Wu et al. and added some, randomly chose 200 different source nodes from each and ran both methods one after the other. The full results (in s) can be found in Table 1. The running times of Wu et al.'s method are comparable to those of SSMD$_\gamma$. Our method does more operations, since it must compute latencies as well and ensure the distances correspond to the *sf*-metrics, so this is encouraging and unexpected. All datasets are heterogenous, which explains the variability in running times and we have not yet pinpointed any hidden link stream parameter that would precisely explain this variability, including the measure used for the visualizations of Fig. 4.

Remark 5.2 *(Notes on the datasets used)* The datasets are only used as benchmarks. They all describe discrete temporal networks and can be found as part of the KONECT library of networks (Kunegis 2013). Only the values of the parameters $|V|$, $|\Omega|$ and $|E_\Omega|$ are extracted in Table 1 since only these were required for our experiments.

A short description of the datasets used follows in Appendix B. Figure 4 shows the temporal evolution of some network measures on two of the datasets used: Arxiv-Hepph and Wikiconflict. On each, we sampled 0.1% of the dataset to build a sublink stream. On each sublink stream and on each induced graph G_t, we evaluated the density of G_t and normalized the values to [0, 1]. We chose this measure as an indicator of the temporal evolution of each dataset. It shows how varied the datasets are. Even though in both cases we observe strong fluctuations in the density, in the Wikiconflict dataset the density appears stratified. Meanwhile, in the Arxiv-Hepph dataset this density is mostly low and takes more distinct values in the range [0, 0.4], which hints that its connectivity varies more than in the other dataset. The Wikiconflict therefore seems to



(a) Scatterplot of values from the Arxiv-Hepph dataset



(b) Scatterplot of values from the Wikiconflict dataset

Fig. 4 Visualization of the temporal evolution of the density on sublink streams of two datasets, Arxiv-Hepph and Wikiconflict. Each sublink stream was constructed by sampling 0.1% of the dataset. On each sublink stream, we evaluated on each induced graph G_t the density of G_t. Values are normalized to [0, 1]

have groups of edges of similar sizes appear at regular times, while in the other dataset there is a less obvious pattern in the appearances of edges.

Remark 5.3 *(Further implementation details)* Previous results (Simard 2019a) showed SSMD_γ to be unstable on some datasets compared with Wu et al. This can be explained by the data structure used to implement R. Note that $\mathrm{R}_v[s_v]$, for any node v and starting time s_v, can grow as $O(|\Omega|)$ which can be large. Thus, even logarithmic search in this structure can be costly. Notice also that we only ever need the largest key of R_v and the smallest distance of $\mathrm{R}_v[s_v]$. Thus, we reimplemented dictionary R with (max/min)-heaps as explained in Remark 4.7. This provides constant time access to the largest/smallest element. This is a choice of implementation to increase the performance of this method and all results presented here that specifically test the performance of SSMD_γ used that implementation. The new method is considerably faster and is comparable with Wu et al.'s method. We did not reimplement the latter's method using heaps because at each step they remove *dominated* elements which naturally tends to make their structures smaller.

Figure 2 shows statistics on the running times and ratios between Algorithm 3 and Wu et al. Observe that even with large running times, the ratios $\left(\frac{\mathrm{SSMD}_\gamma}{\text{Wu et al.}}\right)$ are still decent.

5.2 Comparison between algorithms SSMD and MSMD

Algorithms SSMD and MSMD were run on a set of randomly generated link streams of size $|V|$ ranging from 100 to 170, with increments of 10. Although the link streams are small in scale, the running times are significant since we compute the distances from every source to every destination. The link streams were constructed by generating Erdös-Renyi graphs $G(n, p)$, with $n = |V|$ and $p = 0.7$. Then, on each edge (u, v), we drew a time instant $t \in \{0, 1, \dots, 7\}$ uniformly at random and added both directed edges (t, uv) and (t, vu) to E. In this case, edges have no duration and the time instants are integers: this helps ensure the size of Ω is fixed and small, so the running times scale only with $|V|$ and $|E_\Omega|$.

Figure 5a shows the running times of each algorithms on a link stream with a fixed number of nodes. We observe that, as the number of nodes involved increases, the amount of time taken by SSMD grows faster than that of MSMD. This gives clear indication that this latter method is faster than the former. In terms of scale, the MSMD method manages a link stream of 170 nodes and about 20,000 edges in less than 3 s. Its counterpart takes more than 25 min for the same calculations.

Since MSMD is more scalable than SSMD, we generated a new set of link streams, again with the same process as before, with time instants drawn uniformly at random in the set $\{0, \dots, 10\}$ while the duration of an edge (t, uv) is drawn uniformly at random in the subset $\{0, \dots, 10 - t\}$. In that case, the size of Ω barely varies. We let $|V| \in \{10, 20, \dots, 190, 200\}$. The results are summarized in Fig. 5b. We fitted, with the statistical software R (R Core Team 2013), a linear model on the runtime of MSMD as function of $|E_\Omega|$ in order to extrapolate the runtime of this method for larger values of link stream parameters. Since the number of nodes and event times are low compared to the number of edges, the trend is linear in the number of edges. The fit is reasonable and this is sufficient to illustrate the scaling trend. Extrapolating, we obtain the values shown with the blue line. The trend does not suggest the method is at this point scalable to big link streams as in

Table 1 Runtime comparisons (in s) between SSMD$_\gamma$ and Wu et al.'s method

Dataset	$\|V\|$	$\|\Omega\|$	$\|E_\Omega\|$	Wu et al.	SSMD$_\gamma$	Runtime ratio
facebook-wosn	63731	204914	817035	16.40	12.90	0.8
contact	274	15662	28244	1.00	1.16	1.2
lkml person	337509	624757	1565683	54.20	18.80	0.3
delicious ut	4512099	1583	301186579	5190.00	8740.00	1.7
movielens	16528	34535	95580	2.49	1.20	0.5
dnc	1891	10176	39264	1.12	1.35	1.2
enron	87273	178721	1148072	23.10	22.40	1.0
munmun twitter	530418	175218	4664605	174.00	196.00	1.1
lastfm band	174077	1058994	19150868	103.00	124.00	1.2
elec	7118	90741	103675	3.77	3.43	0.9
epinions-rating	755760	501	13668320	137.00	75.60	0.6
flickr-growth	2302925	134	33140017	1400.00	2180.00	1.6
dblp	12590	30	49759	1.18	0.36	0.3
sociopatterns hyper	113	973	20818	0.28	0.25	0.9
digg	30398	9125	87627	3.67	1.62	0.4
prosper loans	89269	1259	3394979	33.10	27.30	0.8
sociopatterns infect	410	223	17298	0.47	0.37	0.8
delicious ui	25221771	1583	301186579	6950.00	2200.00	0.3
mit	96	33452	1086404	14.30	13.20	0.9
wikiconflict	116836	215982	2917785	37.70	40.80	1.1
slashdot-threads	51083	67327	140778	9.16	6.18	0.7
lastfm song	1084620	1058994	19150868	197.00	91.20	0.5
arxiv-hepph	28093	2337	4596803	56.70	57.90	1.0
youtube-growth	3223585	203	9375374	518.00	299.00	0.6

Datasets with runtime less than 10 s are in bold

Table 2 Summary statistics of running times (in s) between algorithms SSMD$_\gamma$ and Wu et al. (2014), with Q. standing for quartile

Method	Min	1st Q.	Median	Mean	3rd Q.	Max
SSMD$_\gamma$ (s)	0.25	1.55	20.6	588.0	99.4	8740
Wu et al. (s)	0.28	3.38	28.1	622.0	146.0	6950
Ratios	0.31	0.53	0.85	0.84	1.09	1.68

The ratio goes above 1 only at about the 3rd quartile. Outliers still exist

general the sizes of V and Ω would also grow to be much larger than in this experiment.

5.3 MSMD against MSMD$_\gamma$ on synthetic link streams with varying densities

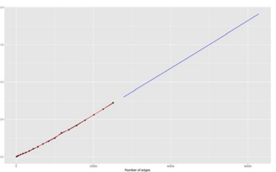
We ran algorithms MSMD and MSMD$_\gamma$ side-by-side on synthetic link streams to see how well the latter performs against the former. It is expected that MSMD$_\gamma$ would be faster. However, instead of looking at the trend on link streams with varying number of nodes, we investigated this trend against the density of the stream. Thus, we constructed link streams as before from graphs $G(n, p)$, while varying the parameter p instead of n. The density of the resulting link stream varies with p. The number of nodes is fixed to 200 and edges have positive integer duration, to allow duration while keeping the cost of computation low.

Experimentally, the density of each time-induced graph of the link stream affects the performance of MSMD$_\gamma$ against MSMD. This can be seen in Fig. 6: as the value of p increases, after a threshold around $p \approx 0.63$, MSMD$_\gamma$ takes longer to complete than MSMD. This suggests that when many edges appear at the same time, it becomes advantageous to compute all-pairs distances in the time-induced graphs and reuse them on each connected components. Note that these do not solve exactly the same problem since one works with $\gamma > 0$ while the other does not. However, if the choice of γ is a modeling parameter,[2] this can be interesting to take into account. It is not clear why the running time of MSMD is decreasing for

[2] The choice of whether γ should be positive or null depends on the application. For example, if one models the interactions in minutes between individuals that meet face-to-face, then the transmission delay γ between any two individuals is negligible.

$\|V\|$	$\|E_\Omega\|$	MSMD (s)	SSMD (s)
100	6814	0.77	209.97
110	8284	0.98	341.77
120	9908	1.15	444.65
130	11654	1.30	555.58
140	13554	1.58	785.46
150	15564	1.77	952.33
160	17744	2.09	1371.26
170	20060	2.43	1586.48

(a) Comparisons between algorithms SSMD and MSMD. Edges have zero duration. Number of event times $|\Omega|$ is 11, $p = 0.7$ and times are in seconds.



(b) Runtimes of MSMD in seconds as a function of E_Ω. Edges have positive integer durations. Probability $p = 0.7$. The black dots and line are results, red line is statistically fitted and blue line is the prediction. Since $|V|$ and $|\Omega|$ are kept relatively low, the running times increase linearly with $|E_\Omega|$.

Fig. 5 Runtimes (in s) of algorithms SSMD and MSMD on synthetic link streams. Link streams are generated randomly with fixed seed (color figure online)

low probabilities, $p \in (0, 0.25)$. Recall that the connectivity threshold of $G(n, p)$ is $\frac{\log n}{n}$. Evaluating this adjusted threshold in the form of $\frac{\log |V||\Omega|}{|V||\Omega|} \approx 0.035$ does not explain why the two lines intersect.

Remark 5.4 In link streams, whether edges have durations or not and whether those durations are integers or real numbers can slow down our methods as durations influence the sizes of Ω and E_Ω. Running times with real edge durations can sometimes be prohibitive, so we focused our experiments on integer durations.

5.4 Comparing algorithms SSMD$_\gamma$ and MSMD$_\gamma$ on real datasets

Algorithm MSMD$_\gamma$ can terminate in reasonable time on some datasets, as opposed to MSMD. In order to probe this method further, we tested how long it would take for this method to terminate on some datasets against SSMD$_\gamma$.

We compared both methods on selected datasets on which SSMD$_\gamma$ took less than 10 s to finish (as observed in Table 1). This way, we could expect it to finish in a decent amount of time from all sources. We ran SSMD$_\gamma$ from all sources alongside MSMD$_\gamma$. For the same task, it is faster to run the specialized method MSMD$_\gamma$. However, note that both methods could finish in a short amount of time on real-world datasets, which is positive. Statistics on the running times of both methods are summarized in Table 3.

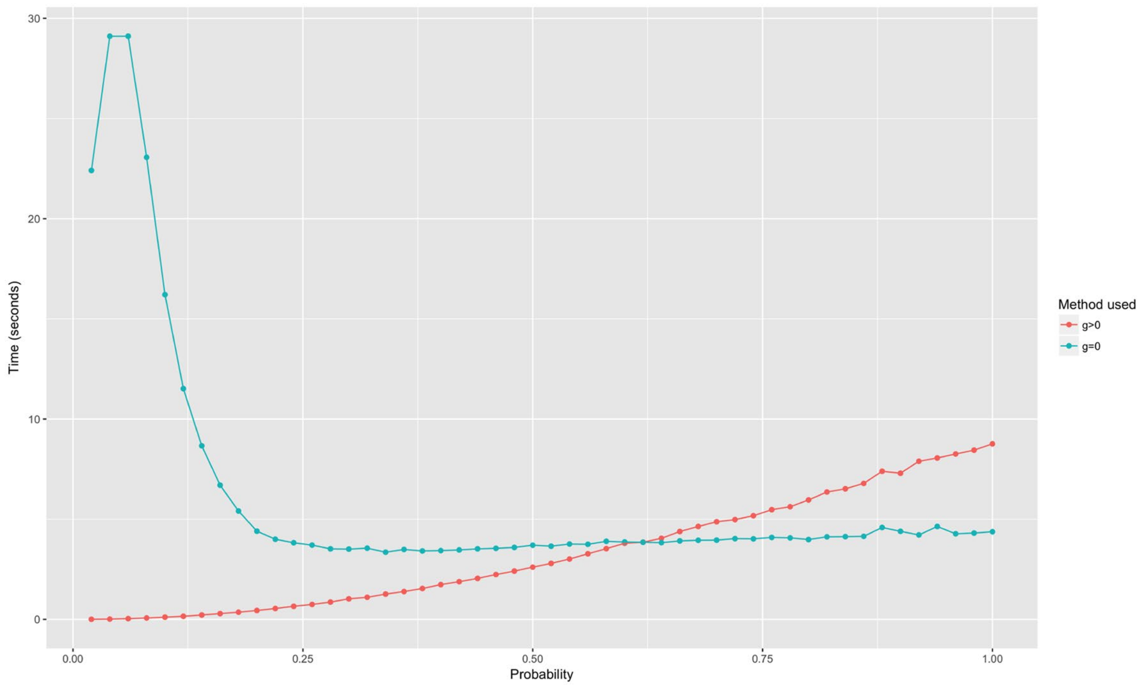


Fig. 6 Running time (in s) of MSMD$_\gamma$ (in red) and MSMD (in blue) on synthetic link streams generated with $G(200, p)$ with varying values of p. As p increases, so does the density of each induced graphs, favoring MSMD over MSMD$_\gamma$. Edges have positive integer duration (color figure online)

Table 3 Summary statistics of the running times (in s) between algorithms SSMD_γ and MSMD_γ

Method	Min	1st Q.	Median	Mean	3rd Q.	Max
SSMD_γ (s)	1.10	2.89	3.77	48.80	71.0	216.0
MSMD_γ (s)	0.30	2.16	3.97	12.90	24.6	27.6

In practice, SSMD_γ is faster on some datasets than its counterpart. However, statistically MSMD_γ is better suited to compute all metrics

6 Conclusion

In this paper, we presented different algorithms to compute metrics between pairs of event nodes. As opposed to similar known algorithms, those methods return all metrics at once in a single pass over the dataset. Moreover, the starting and arrival times of shortest paths are returned, which is valuable information to compute, for example, the betweenness centrality of temporal nodes.

Algorithm SSMD works from a fixed source and is suitable when not all pairwise metrics are required. Our experiments show that SSMD_γ is comparable to the state-of-the-art method to compute distances from a source node to all other nodes. These results improve on previous ones (Simard 2019a) and make use of a more efficient data structure. Even though some experiments are advantageous to our methods, we do not claim they are in general faster than the state of the art. However for the task at hand of computing all metrics at once on a link stream, we think we can fairly conclude that our methods are usable in practice. The experiment in Sect. 5.1 was designed to illustrate if SSMD_γ could be used in real-world setting. Thus, comparing it to another shortest path method is relevant since it does not do significantly more operations that this type of algorithm. One could surely implement the shortest path algorithm of Wu et al. to make it faster than SSMD_γ.

In practice, MSMD has finished its task faster than its counterpart on synthetic link streams. Since the link streams used were smaller than what we would expect from real-world instances, we extrapolated the running times produced by MSMD. At this point, scalability is an issue, when $\gamma = 0$, and we could not expect to run this method on realistic link streams and obtain results in a reasonable amount the time. Thus, in order to speed up the computation time, we suggest studying how to lessen the amount of operations in either methods by skipping some temporal nodes and extrapolating the distances. Also, finding ways not to have to recompute the connected components and the all-pairs distances at every time would also be helpful in improving both methods.

Both algorithms SSMD_γ and MSMD_γ, when $\gamma > 0$, could finish their task on some datasets in a decent amount of time. Algorithm MSMD_γ took less than 30 s to finish all tasks, even on datasets of almost 100, 000 edges. Thus, in that case, realistic datasets of decent size could be handled by our methods.

In light of our experiments as well as the recent literature (see Himmel et al. 2019), it would be interesting to have a lower bound on the complexity of computing our metrics. Since shortest paths methods seem to be based on the same arguments as shortest paths algorithms in graphs, it might also be relevant to deduce a lower bound in terms of the latter. That is, how much slower is it to compute distances in a link stream as opposed to computing distances in a graph? Can this complexity be expressed in terms of natural parameters of the link stream such as V, Ω and E_Ω?

Aside from scalability, another limitation of this study lies in the ordering of the objective functions we chose to optimize. Namely, we compute lengths of shortest fastest paths. If one were to require lengths of fastest shortest paths, our methods would need to be redesigned. Moreover, the multitude of possible combinations of optimal paths to compute (foremost paths, shortest foremost paths, etc.) is not all considered in this work. Brunelli et al. tackled this limitation by building a more general framework for optimal paths. We believe our methods can be modified to compute some other types of paths combining temporal and structural information hierarchically, such as shortest foremost paths. In turn, those paths can be used to compute other centralities than the betweenness centrality or to investigate different topics such as reachability. In Algorithm 3, for example, whenever we consider an edge (t, uv) with $u \neq s$, we have access to the last arrival time a_u from the source at time s_u. If we instead considered the *first* such arrival time, then we could decide if the path from (s_u, s) to (t, v) that involves the edge (t, uv) is *restless* (see Casteigts et al. 2020) or not and update the relevant dictionaries accordingly, say of *restless* reachability triples.

Algorithms

Algorithm 1: SSMD *sf*-metric

```
Input: $L = (T, V, E)$ a link stream, $\Omega$ the set of event times, $(t_s, s)$ a source event node
Output: Dictionaries $d, f$ of sf-metrics and latencies from $(t_s, s)$ to all other event nodes, set of dictionaries $\mathrm{R}_v$ for each $v \in V$
1  $f, d \leftarrow$ create dictionaries
2  for $v \in V$ do $\mathrm{R}_v \leftarrow$ create dictionary
3  for $t \in$ Sorted($\{t_0 \in \Omega \,|\, t_0 \geq t_s\}$) do
4      for $C \in$ connected_components($G_t$) do
5          $H \leftarrow G_t$.induced_subgraph($C$)
6          $d' \leftarrow$ all_pairs_distances($H$)
7          $D \leftarrow \{\}$
8          if $s \in C$ then $D$.insert($-t, 0, s$)
9          else
10             $s_v \leftarrow \max_{u \in C} \mathrm{R}_u$.last()
11             for $v \in C$ do
12                 $D$.insert all $(-s_v, d_v, v)$ such that
13                 $(a_v, d_v) \in \mathrm{R}_v[s_v]$ with largest $a_v$
14         for $(s_u, d_u, u) \in$ Sorted($D$) do
15             $U \leftarrow \{v \in C \,|\, \exists a_v : (a_v, d_u) \in \mathrm{R}_v[-s_u]\}$
16             if $u = s$ then $U \leftarrow \{s\}$
17             for $w \in C$ do
18                 $(\_, d_*) \leftarrow \mathrm{R}_w[s_u]$.last()
19                 $d_{\min} \leftarrow \min(d_u + \min_{u \in U} d'[(u, w)], d_*)$
20                 $\mathrm{R}_w[-s_u]$.remove all $(t, d_0)$ s.t. $d_0 > d_{\min}$
21                 $\mathrm{R}_w[-s_u]$.insert($t, d_{\min}$)
22                 $f^*_w \leftarrow \min_{(t_0, w) \in f} f[(t_0, w)]$
23                 $f[(t, w)] \leftarrow \min(t + s_u, f^*_w)$
24                 $d[(t, w)] \leftarrow$ min $d_0$ s.t. $s_0 \in R_w$, $(a_0, d_0) \in R_w[s_0]$ and $a_0 - s_0 = f[(t, w)]$
25 return $d, f, \{\mathrm{R}_v \,|\, v \in V\}$
```

Algorithm 2: MSMD *sf*-metric

```
Input: $L = (T, V, E)$ a link stream, $\Omega$ the set of event times
Output: $F$ a dictionary of latencies, $D^0$ a dictionary of reachability triples, $D$ a dictionary of sf-metrics
1  for $u, v \in V$ do $SA_{uv}, F_{uv}, D_{uv}, D^0_{uv} \leftarrow$ create sorted dictionaries
2  for $t \in \Omega$ do
3      $t^- \leftarrow$ last time of $\Omega$ before $t$
4      for $C \in$ connected_components($G_t$) do
5          $H \leftarrow G_t$.induced_subgraph($C$)
6          $d_C \leftarrow$ all_pairs_distances($H$)
7          for $u, v \in C$ do
8              $SA_{uv}$.insert($t, t$)
9              $D^0_{uv}[t]$.insert($t, t, d_C[u, v]$)
10             $F_{uv}[t]$.insert(0)
11         for $u \in C, v \in V \setminus C$ do
12             $C_v \leftarrow$ conn. component of $G_t$ containing $v$
13             $s_v \leftarrow \max_{w \in C_v, (s,a) \in SA_{uv}}(s)$
14             $SA_{uv}$.insert($s_v, t$)
15             $d_{\min} \leftarrow \infty$
16             for $w \in C_v$, do
17                 $(\_, \_, d_w) \leftarrow D^0_{uw}[t^-]$.last()
18                 $d_{\min} \leftarrow \min(d_{\min}, d_w + d_C[w, v])$
19             $D^0_{uv}[t]$.insert($s_v, t, d_{\min}$)
20             $l_{uv} \leftarrow \min_{(s,a) \in SA_{uv}}(a - s)$
21             $l = \min(l_{uv}, F_{uv}[t^-])$
22             $F_{uv}[t] \leftarrow l$
23             $(s^*, a^*) \leftarrow$ pair $(s, a) \in SA_{uv}$ s.t. $a - s = l$
24             $D_{uv}[t][s^*] \leftarrow d^*$ s.t. $(s^*, a^*, d^*) \in D^0_{uv}[t]$
25 return $F, D^0, D$
```

Algorithm 3: SSMD *sf*-metric with $\gamma > 0$ (SSMD$_\gamma$)

```
Input: $L = (T, V, E)$ a link stream, $\Omega$ the set of event times, $(t_s, s)$ a source event node
Output: Dictionaries $d, f$ of sf-metrics and latencies from $(t_s, s)$ to all other event nodes, set of dictionaries $\{\mathrm{R}_v \,|\, v \in V\}$
1  $d, f, \leftarrow$ create dictionaries
2  for $v \in V$ do $\mathrm{R}_v \leftarrow$ create dictionary
3  for $(t, uv) \in$ Sorted($E_\Omega$) s.t. $t \geq t_s$ do
4      if $u = s$ then $\mathrm{R}_s[t]$.insert($t, 0$)
5      if $\mathrm{R}_u \neq \emptyset$ and dur($t, u, v$) $\geq \gamma$ then
6          $s_u \leftarrow \mathrm{R}_u$.last()
7          $(a_u, d_u) \leftarrow \mathrm{R}_u[s_u]$.last()
8          if $s_u$ exists then
9              $d_v \leftarrow d_u + 1$
10             if $\mathrm{R}_v[s_u]$ does not contain $(t', d')$ s.t. $t' \leq t + \gamma$ and $d' < d_v$ then
                   $\mathrm{R}_v[s_u]$.insert($t + \gamma, d_v$)
11         $f_v \leftarrow t - s_u$
12         if $f[v] \neq \emptyset$ then
13             $(\_, f'_v) \leftarrow f[v]$.last()
14             if $f'_v < f_v$ then $f_v \leftarrow f'_v$
15         $f[v]$.add($t, f_v$)
16         $d_{\mathrm{fas}} \leftarrow$ min $d_0$ s.t. $(a_0, d_0) \in R_v[s_u]$ and $a_0 - s_u = f_v$
17         if $d_{\mathrm{fas}}$ exists then $d[v]$.add($t, d_{\mathrm{fas}}$)
18 return $d, f, \{\mathrm{R}_v \,|\, v \in V\}$
```

Algorithm 4: MSMD *sf*-metric with $\gamma > 0$ (MSMD$_\gamma$)

```
Input: $L = (T, V, E)$ a link stream, $\Omega$ the set of event times
Output: $F$ a dictionary of latencies, $D^0$ a dictionary of reachability triples, $D$ a dictionary of sf-metrics
for $u, v \in V$ do $SA_{uv}, F_{uv}, D_{uv}, D^0_{uv} \leftarrow$ create sorted dictionaries
for $(t, u, v) \in \text{Sorted}(E_\Omega)$ s.t. $\text{dur}(t, u, v) \geq \gamma$ do
    $SA_{uv}[t]$.insert$(t, t + \gamma)$
    $D^0_{uv}[t]$.insert$(t, t + \gamma, 1)$
    $F_{uv}[t]$.insert$(0)$
    for $w \in V \setminus \{v\}$ do
        $s_w \leftarrow \max_{(s,a) \in SA_{wu}[t^-]}(s)$
        if $s_w$ exists then
            ▷ There is a path from $w$ to $v$ that starts on $s_w < t$
            $SA_{wv}[t]$.insert$(s_w, t + \gamma)$
            $d_v \leftarrow D^0_{wv}[t^-]$.last()
            $d_u \leftarrow D^0_{wu}[t^-]$.last()
            if $d_v > d_u + 1$ then
                $D^0_{wv}[t]$.insert$(s_w, t + \gamma, d_u + 1)$
            else $D^0_{wv}[t]$.insert$(s_w, t + \gamma, d_v)$
        $l_{wv} \leftarrow \min_{(s,a) \in SA_{wv}[t]}(a - s)$
        $l \leftarrow \min(l_{wv}, F_{wv}[t^-])$
        $F_{wv}[t] \leftarrow l$
        $(s^*, a^*) \leftarrow$ pair $(s, a) \in SA_{wv}[t]$ s.t. $a - s = l$
        $D_{wv}[t][s^*] \leftarrow d^*$ s.t. $(s^*, a^*, d^*) \in D^0_{wv}[t]$
return $F, D^0, D$
```

Datasets

The following datasets were used in the experiments. More documentation can be found online (Kunegis 2019). Datasets are part of the KONECT project (Kunegis 2013). Descriptions below are found in the directories of the datasets.

1. `arxiv-hepph` A co-citation network from the website arXiv arxiv.org of scientific papers. Papers are taken from the high energy physics phenomenology (hep-ph) section. Two papers are linked if they are both cited by another paper, with the timestamp indicating the date of the latter's publication.
2. `contact` A network of human contacts. It describes the Haggle network, a project funded by the European Union. Each edge describes a contact between two persons measured by carried wireless devices.
3. `dblp` A citation network. Extracted from the website dblp.uni-trier.de/ of scientific publications. Nodes are publications and two publications are linked if one cites the other.
4. `delicious ui` A user-url network from the website https://del.icio.us (now deprecated). Edges connect users to the urls they tagged.
5. `delicious ut` An interaction network from the same website. Edges connected users to the tags they used.
6. `digg` A communication network. Extracted from the website digg.com. Edges connect two users when one replied to the other.
7. `dnc` A communication network. Extracted from the 2016 Democratic National Committee email leak of the american Democratic Party. Edges connect two people if one sent an email to the other.
8. `elec` An online contact network. Represents admin elections in the English Wikipedia. Edges connect two users if one voted for or against the other.
9. `enron` A communication network. Describes email exchanges in the former company Enron.
10. `epinions-ratings` A ratings network. The website seems deprecated. Each edge connects a user and a product they rated.
11. `facebook-wosn` A social network. Data are extracted from a portion of the website Facebook facebook.com. Edges connect users that are friends on the website.
12. `flickr-growth` A social network. Edges describe friendship connections on the website flickr.com.
13. `lastfm band` An interaction network between users and bands. From the website last.fm. Edges connect users to bands they listened to.
14. `lastfm song` An interaction network between users and songs listened. From the same website as above, edges connect users to songs they listened to.
15. `lkml person` An interaction network of people on the Linux Kernel Mailing List (lkml). Edges connect people to threads in the mailing list they contributed to.
16. `mit` A human contact network. Edges connect 100 students when they had contact with each other.
17. `movielens` An interaction network. Extracted from the website http://movielens.umn.edu/. Edges connect users to the tags they used.
18. `munmun twitter` An interaction network of users and tags. Extracted from the website twitter.com. Edges connect users to the tags they used in tweets.
19. `prosper loans` An interaction network. Extracted from the website prosper.com. Edges connect people (lenders) to other people (borrowers) to whom they lent money.
20. `slashdot-threads` A comunication network. Extracted from the website https://slashdot.org/. Edges connect users when one replied to another.
21. `sociopatterns hyper` A human contact network. Edges represent face-to-face contacts of more than 20 s.
22. `sociopatterns infect` A human contact network. Edges represent face-to-face contacts of more than 20 s.
23. `wikiconflict` A network of online contacts. Extracted from the english Wikipedia https://en.wikip

edia.org/wiki/Main_Page. Each edge connects users to other users with whom they are in editing conflicts.

24. `youtube-growth` A social network. Extracted from the website youtube.com. Edges connect users with their friends on the platform.

Acknowledgements The authors would like to thank the Professors Paola Flocchini, Jean-Lou De Carufel and Nicola Santoro for their valuable insights.

Funding This work was supported by the Fonds de Recherche Québécois Nature et Technologies (FRQNT) (Grant No. 197014).

Declarations

Conflicts of interest There are no conflicts of interest.

Code availability Implementations of the four algorithms developed in this work are available online: Simard (2019b). The datasets used are also available on the web: Kunegis (2019).

References

Brunelli F, Crescenzi P, Viennot L (2021) On computing Pareto optimal paths in weighted time-dependent networks. Inf Process Lett 168:1–12. https://doi.org/10.1016/j.ipl.2020.106086

Casteigts A, Flocchini P, Quattrociocchi W, Santoro N (2012) Time-varying graphs and dynamic networks. Int J Parallel Emergent Distrib Syst 27(5):387–408

Casteigts A, Flocchini P, Mans B, Santoro N (2015) Shortest, fastest, and foremost broadcast in dynamic networks. Int J Found Comput Sci 26(4):499–522. https://doi.org/10.1142/S0129054115500288

Casteigts A, Himmel AS, Molter H, Zschoche P (2020) Finding temporal paths under waiting time constraints. In: 31st international symposium on algorithms and computation (ISAAC 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik

Cattuto C, Van den Broeck W, Barrat A, Colizza V, Pinton JF, Vespignani A (2010) Dynamics of person-to-person interactions from distributed RFID sensor networks. PLoS ONE 5(7):1–9. https://doi.org/10.1371/journal.pone.0011596

Ciaperoni M, Galimberti E, Bonchi F, Cattuto C, Gullo F, Barrat A (2020) Relevance of temporal cores for epidemic spread in temporal networks. Sci Rep 10(1):1–15. https://doi.org/10.1038/s41598-020-69464-3 (**arXiv:2003.09377**)

Cinelli M, Quattrociocchi W, Galeazzi A, Valensise CM, Brugnoli E, Schmidt AL, Zola P, Zollo F, Scala A (2020) The covid-19 social media infodemic. Sci Rep 10(1):1–10

Colson B, Marcotte P, Savard G (2007) An overview of bilevel optimization. Ann Oper Res 153(1):235–256. https://doi.org/10.1007/s10479-007-0176-2 (**arXiv:1011.1669v3**)

Ferreira A (2004) Building a reference combinatorial model for MANETs. IEEE Netw 18(5):24–29. https://doi.org/10.1109/MNET.2004.1337732

Himmel AS, Bentert M, Nichterlein A, Niedermeier R (2019) Efficient computation of optimal temporal walks under waiting-time constraints. arXiv:1909.01152

Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125. https://doi.org/10.1016/j.physrep.2012.03.001

Kempe D, Kleinberg J, Kumar A (2002) Connectivity and inference problems for temporal networks. J Comput Syst Sci 64(4):820–842. https://doi.org/10.1006/jcss.2002.1829

Kunegis J (2013) Konect: the koblenz network collection. In: Proceedings of the 22nd international conference on world wide web. ACM, pp 1343–1350

Kunegis J (2019) The konect project. http://konect.cc/. Accessed: 21 Oct 2019

Latapy M, Viard T, Magnien C (2018) Stream graphs and link streams for the modeling of interactions over time. Soc Netw Anal Min 8(1):61

Li M, Xin J, Wang Z, Liu H (2019) Accelerating minimum temporal paths query based on dynamic programming. In: International conference on advanced data mining and applications. Springer, pp 48–62

Mellor A (2017) The temporal event graph. J Complex Netw 6(4):639–659. https://doi.org/10.1093/comnet/cnx048

Moinet A, Pastor-Satorras R, Barrat A (2018) Effect of risk perception on epidemic spreading in temporal networks. Phys Rev E 97:012313. https://doi.org/10.1103/PhysRevE.97.012313

R Core Team (2013) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org/

Simard F (2019a) On computing distances and latencies in Link Streams. In: Proceedings of The 2019 IEEE/ACM international conference on advances in social networks analysis and mining. ACM, Vancouver, Canada

Simard F (2019b) SSMD and MSMD repository. https://bitbucket.org/simfr404/linkstreams_cpp/src/master/. accessed: 18 May 2021

Simard F, Magnien C, Latapy M (2021) Computing betweenness centrality in link streams. arXiv preprint arXiv:210206543

SocioPatterns (2021) Sociopatterns collaboration. www.sociopatterns.org. Accessed: 18 May 2021

Stehlé J, Charbonnier F, Picard T, Cattuto C, Barrat A (2013) Gender homophily from spatial behavior in a primary school: a sociometric study. Soc Netw 35(4):604–613. https://doi.org/10.1016/j.socnet.2013.08.003

Tang J, Musolesi M, Mascolo C, Latora V (2010a) Characterising temporal distance and reachability in mobile and online social networks. ACM SIGCOMM Comput Commun Rev 40(1):118. https://doi.org/10.1145/1672308.1672329

Tang J, Musolesi M, Mascolo C, Latora V, Nicosia V (2010b) Analysing information flows and key mediators through temporal centrality metrics. In: Proceedings of the 3rd workshop on social network systems (SNS '10). ACM, Paris, France. https://doi.org/10.1145/1852658.1852661

Thejaswi S, Gionis A (2020) Restless reachability in temporal graphs. arXiv preprint arXiv:201008423

Wu H, Cheng J, Huang S, Ke Y, Lu Y, Xu Y (2014) Path problems in temporal graphs. Proc VLDB Endow 7(9):721–732. https://doi.org/10.14778/2732939.2732945

Xuan BB, Ferreira A, Jarry A (2003) Computing shortest, fastest, and foremost journeys in dynamic networks. Int J Found Comput Sci 14(02):267–285