# Efficient Top-k Temporal Closeness Calculation in Temporal Networks

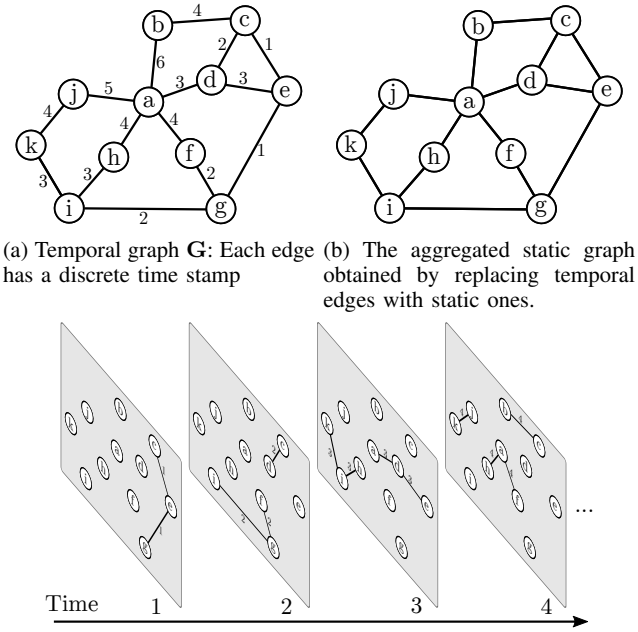
Lutz Oettershagen, Petra Mutzel
*Institute of Computer Science*, *University of Bonn*
Bonn, Germany
{lutz.oettershagen,petra.mutzel}@cs.uni-bonn.de

***Abstract*—We consider the problem of efficiently computing the top-*k* temporal closeness values and the corresponding vertex sets in a given temporal network. The closeness centrality of a vertex in a classical static graph is the reciprocal of the sum of the distances to all other vertices. Temporal distances in networks that change over time take the dynamics into account. In this work, we consider the harmonic temporal closeness with respect to the shortest duration distance. We introduce an efficient algorithm for computing the exact top-*k* temporal closeness values and the corresponding vertices. The algorithm can be generalized to the task of computing all closeness values. Furthermore, we derive a modification leading to a heuristic that often performs well on the majority of real-world data sets and drastically reduces the running times. For the case that edge traversal takes an equal amount of time for all edges, we lift two approximation algorithms to the temporal domain. The algorithms approximate the transitive closure of a temporal graph (which is an essential ingredient for the top-*k* algorithm) and the temporal closeness for all vertices, respectively, with high probability. We experimentally evaluate all our new approaches on real-world data sets and show that they lead to drastically reduced running times while keeping high quality in many cases. Moreover, we demonstrate that the top-*k* temporal and static closeness vertex sets differ quite largely in real-world networks.**

***Keywords*-Temporal Graphs, Temporal Closeness, Centrality**

## I. Introduction

Centrality measures are a cornerstone of social network analyses. One of the most popular and well-researched centrality measures is closeness, first introduced by Bavelas [1]. In a static and undirected graph, the closeness of a vertex is the inverse of the sum of the smallest distances to the other vertices of the network. However, many real-world networks are *temporal*, e.g., in a social network, persons only interact at specific points in time. Recently, the analyses of dynamic networks, or temporal graphs, has gained increasing attention [2], [3], [4], [5], [6]. Here, a temporal network consists of a set of vertices and a set of temporal edges. Each temporal edge is only available at a specific discrete point in time, and edge traversal costs a strictly positive amount of time. Therefore, a temporal graph can be seen as a sequence of static graphs over a fixed set of vertices and edges that evolve over time. Figure 1a shows a temporal graph, where the edges are labeled with the time stamps of their existence. Figure 1c shows the representation as a sequence of static graphs over time. The temporal properties direct any possible flow of information in the network. For example, a dissemination process on the network, such as the spread of rumors, fake news, or diseases, has to respect the forward flow of time. Hence, a meaningful adaption of a conventional path is the time-respecting *temporal path*. In a temporal path, at each vertex, the time stamp of the next edge of the path cannot be earlier than the arrival time at the vertex. Modifications of static closeness to temporal closeness using temporal path and *temporal distances* have been suggested [7], [8], [9], [10]. One of these temporal distances is the shortest duration, i.e., the duration of the *fastest path* between two nodes. Here, we consider the *harmonic temporal closeness* of a vertex. We define it as the sum of the reciprocals of the durations of the fastest paths to all other vertices. Harmonic closeness for non-connected static graphs was introduced in [11]. The reason for using the harmonic variant for temporal closeness is that reachability between vertices, even in an undirected temporal graph, is restricted. It has been shown that in networks modeling dissemination processes, like the spread of viruses or fake news, a vertex




(a) Temporal graph $\mathbf{G}$: Each edge has a discrete time stamp

(b) The aggregated static graph obtained by replacing temporal edges with static ones.

(c) The representation of $\mathbf{G}$ as a sequence of static graphs, in which at time $t$ only the edges with time stamp $t$ are available. Only the first four static graphs are shown, i.e., for $t \leq 4$.

Fig. 1

2374-8486/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDM50108.2020.00049

with a high temporal closeness can be expected to be of high importance for the transportation or dissemination process [12], [9]. In general, high temporal closeness vertices in temporal networks differ from high static closeness and high degree vertices. For example, in the temporal graph shown in Figure 1a, traversing an edge takes one time unit for all edges. Figure 1b shows the aggregated static graph, in which static edges replace the temporal edges. In this static graph, the most central vertex in terms of static closeness is $a$. It also has the most outgoing edges. However, in the temporal graph, the vertex with the highest temporal closeness is vertex $e$. Notice that in the temporal graph, vertex $a$ can only reach its direct neighbors, while vertex $e$ can reach all other vertices. Wu et al. [10] empirically showed that the correlation between temporal closeness in temporal graphs and the static closeness in the aggregated graphs is low. Computing the temporal closeness of all vertices exact can be costly because it demands finding the fastest temporal path between each pair of vertices. However, often knowing the top-$k$ important vertices is sufficient. To the best of our knowledge, there is no published top-$k$ exact or approximation algorithm for computing the temporal closeness.

Our contributions are:

1) We propose an algorithm for computing the top-$k$ harmonic temporal closeness values and the corresponding vertex sets in a temporal network. This algorithm can be simplified for the task of computing the closeness values of all vertices. The algorithms are based on a new minimum duration path algorithm on temporal graphs.
2) We derive heuristics for the temporal closeness of all and the top-$k$ vertices that, in our experimental evaluation, improve running times significantly and introduce only small errors on almost all data sets.
3) In case of equal transition times, we adapt a stochastic approximation for the transitive closure of temporal graphs. The transitive closure is essential for the top-$k$ algorithm. This approximation speeds up the running time of the top-$k$ algorithm substantially. Moreover, we adapt a stochastic sampling algorithm for approximating the temporal closeness of all vertices directly.
4) We comprehensively evaluate our algorithms using real-world temporal networks. As a baseline, we use a temporal closeness algorithm based on the edge stream fastest path algorithm introduced by Wu et al. [10]. Our approaches decrease the running times for all data sets significantly and are up to a factor of 17 times faster than the baseline.

The omitted proofs are provided in the Appendix which is available at https://gitlab.com/tgpublic/tgcloseness.

## II. Related Work

**Closeness and top-*k* closeness:** Comprehensive overviews and introductions of centrality approaches, including closeness centrality, are provided, e.g., in [13], [14], [15]. The problem of ranking vertices and selecting the $k$ most influential vertices has been widely studied, see, e.g., [16], [17], [18]. Bergamini et al. [16] present algorithms for computing the top-$k$ closeness in unweighted static graphs. They start a breadth-first search (BFS) from each vertex in order to find the shortest distance to all other vertices. In each BFS run, they calculate an upper bound for the closeness of the current vertex. If the current vertex cannot have a top-$k$ closeness value, their algorithm stops the computation early. We adapt this strategy for temporal closeness in Section IV. Our algorithm differs by computing the fastest temporal paths, allowing variable transition times, and using corresponding upper bounds. A simple adaption of the static closeness algorithm from [16] would be impossible because it does not take the temporal features, like temporal edges with transition times, or waiting times at vertices into account. In [19], Bisenius et al. extend the same framework to dynamic graphs in which edges can be added and removed. It allows efficient updates of the *static* closeness after edge insertions or deletions. However, their algorithm works for static closeness using unweighted static shortest paths without considering edge availability or transition times. Eppstein and Wang [20] proposed a randomized approximation algorithm for closeness in weighted undirected graphs. It approximates the closeness of all vertices with a small additive error with high probability. Their algorithm is also not directly applicable to the temporal case. In Section V, we introduce a transformation of temporal graphs to an inverse representation such that the algorithm can be used to approximate temporal closeness. Based on [20], Okamoto et al. [21] introduced an algorithm for approximating the top-$k$ closeness. First, they use an approximation to find a candidate set of vertices. Next, they rank the top-$k$ vertices with high probability. Cohen et al. [22] combined the sampling approach and a pivoting technique for approximating the closeness of all vertices.

**Temporal graphs and temporal closeness:** Comprehensive introductions to temporal graphs, including overviews of different temporal centrality measures, are provided in, e.g., [3], [8]. Early work on fastest paths by Cooke and Halsey [23] suggests to iteratively compute $T$ distance matrices for each vertex, where $T$ is a time-dependent parameter chosen beforehand. Xuan et al. [24] introduce algorithms for finding the shortest, fastest, and earliest arrival paths, which are generalizations of Dijkstra's shortest paths algorithm. Their fastest path algorithm only supports unit traversal times for all edges and enumerates all the earliest arrival paths for each possible starting time. In [25], the authors define variants of temporal graph traversals. A depth-first search (DFS) variant can be used to construct a DFS tree starting at a vertex $v$, which contains information about the fastest paths from $v$ to the other vertices. Wu et al. [10] introduce streaming algorithms operating on the temporal edge stream in which the edges arrive with increasing time stamps. They discuss finding the fastest, shortest, latest departure, and the earliest arrival path and suggest using their algorithms for the computation of temporal closeness. We use their state-of-the-art streaming algorithms for the fastest paths and temporal closeness as a baseline in our experiments. In Section IV-E, we further discuss the differences to our algorithms. In [26], the authors introduce *shortest-*

*fastest* paths as a combination of the conventional distance and shortest duration. They extend Brandes' algorithm [27] for computing betweenness centrality in temporal graphs. Nicosia et al. [28] and Kim and Anderson [29] examine various temporal centrality measures in temporal graphs. In both [12] and [9], the authors compare temporal distance metrics and centrality measures to their static counterparts. They show that the temporal versions for analyzing temporal graphs have advantages over static approaches on the aggregated graphs.

## III. Preliminaries

A *directed temporal graph* $\mathbf{G} = (V, \mathbf{E})$ consists of a finite set of vertices $V$ and a finite set $\mathbf{E}$ of directed *temporal edges* $e = (u, v, t, \lambda)$ with $u$ and $v$ in $V$, $u \neq v$, *availability time* (or *time stamp*) $t \in \mathbb{N}$ and *transition time* $\lambda \in \mathbb{N}$. The transition time of an edge denotes the time required to use the edge. We only consider directed temporal graphs—it is possible to model undirectedness by using a forward- and a backward directed edge with equal time stamps and traversal times for each undirected edge. Notice that for a temporal graph, the number of edges is not polynomially bounded by the number of vertices. Given a temporal graph $\mathbf{G}$, removing all time stamps and merging resulting multi-edges, we obtain the aggregated graph $A(\mathbf{G}) = (V, E_s)$ with $E_s = \{(u, v) \mid (u, v, t, \lambda) \in \mathbf{E}\}$. For a directed temporal graph $\mathbf{G} = (V, \mathbf{E})$ and a *time interval* $I = [\alpha, \beta]$ with $\alpha, \beta \in \mathbb{N}$ and $\alpha \leq \beta$, we define the temporal subgraph $\mathbf{G}^I = (V, \mathbf{E}^I)$ with $\mathbf{E}^I = \{(u, v, t, \lambda) \in \mathbf{E} \mid t \geq \alpha \text{ and } t + \lambda \leq \beta\}$. For $u \in V$ let $\tau_+^I(u) = \{t \mid (u, v, t, \lambda) \in \mathbf{E}^I\}$ and $\tau_-^I(u) = \{t + \lambda \mid (v, u, t, \lambda) \in \mathbf{E}^I\}$. Furthermore, let $\hat{\tau}_+^I = \max\{|\tau_+^I(u)| \mid u \in V\}$, and $\hat{\tau}_-^I = \max\{|\tau_-^I(u)| \mid u \in V\}$, i.e., the largest numbers of different starting or arrival times at any $v \in V$. Finally, let $\mathcal{T}(\mathbf{G})$ be the set of all availability times in $\mathbf{G}$.

### A. Temporal Paths

A *temporal path* $P$ between vertices $v_1, v_{\ell+1} \in V$ of length $\ell$ is an alternating sequence of vertices and temporal edges $(v_1, e_1 = (v_1, v_2, t_1, \lambda_1), v_2, \ldots, e_\ell = (v_\ell, v_{\ell+1}, t_\ell, \lambda_\ell), v_{\ell+1})$ such that each vertex in $P$ is visited exactly once and $t_i + \lambda_i \leq t_{i+1}$ for $1 \leq i < \ell$. For notational convenience we sometimes omit edges. The *starting time* of $P$ is $s(P) = t_1$, the *arrival time* is $a(P) = t_\ell + \lambda_\ell$, and the *duration* is $d(P) = a(P) - s(P)$. A *fastest* path is a path with minimal duration. We say vertex $v$ is *reachable* from vertex $u$ if there exists a temporal $(u, v)$-path. In contrast to classical static graphs, even undirected temporal graphs are, in general, not strongly connected, and have non-symmetric and limited reachability between nodes with respect to temporal paths. Paths and reachability can be restricted to a time interval $I$ and the corresponding subgraph $\mathbf{G}^I$. Let $R^I(u)$ be the set of temporally reachable vertices from $u$ in the time interval $I$, i.e., the set of vertices that can be reached from $u$ using a temporal path in $\mathbf{G}^I$. We define $r^I(u) = |R^I(u)|$.

### B. Harmonic Temporal Closeness

For a temporal graph $\mathbf{G} = (V, \mathbf{E})$ and a time interval $I$, let $\mathcal{P}_{uv}^I$ be the set of all temporal paths between $u, v \in V$ in $\mathbf{G}^I$. We define the shortest duration between $u$ and $v$ as $d^I(u, v) = \min_{P \in \mathcal{P}_{uv}^I}(d(P))$. If $v$ is not reachable from vertex $u$ during the interval $I$, we set $d^I(u, v) = \infty$, and we define $\frac{1}{\infty} = 0$. Due to the restricted reachability in temporal graphs, we use the *harmonic* variant of temporal closeness. Marchiori and Latora [11] introduced harmonic closeness in static graphs.

**Definition 1** (Harmonic Temporal Closeness)**.** *Let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph and $I$ a time interval. We define the harmonic temporal closeness for $u \in V$ with respect to $I$ as $c^I(u) = \sum_{v \in V \setminus \{u\}} \frac{1}{d^I(u,v)}$. We call $c_{\mathrm{n}}^I(u) = \frac{c^I(u)}{|V|}$ the normalized harmonic temporal closeness.*

Now we define the top-$k$ temporal closeness problem.

**Definition 2.** *For a temporal graph $\mathbf{G} = (V, \mathbf{E})$ and $k \in \mathbb{N}$, the* Top-$k$ Harmonic Temporal Closeness Problem *asks for the $k$ largest values of the harmonic temporal closeness and the set of all vertices in $V$ with these values.*

In the following, we often drop the word *harmonic* and call the problem temporal closeness.

## IV. Algorithms for Temporal Closeness

First, we present a new fastest path algorithm, which we then use for the top-$k$ temporal closeness computation. The new algorithm is tailored to be part of our top-$k$ algorithm and operates on the adjacency list representation of the temporal graph, i.e., for each vertex, all out-going edges are in a list.

### A. A Label Setting Fastest Path Algorithm

In general, a fastest path does not consist of fastest sub-paths. For example, in Figure 1c the $(e, a)$-path $P_1 = (e, c, d, a)$ has a duration of $d(P_1) = 4 - 1 = 3$ and is optimal. The $(e, d)$-path $P_2 = (e, c, d)$ has a duration of $d(P_2) = 2$ and is not a fastest path, because the $(e, d)$-path $P_3 = (e, d)$ has duration $d(P_3) = 1$. To handle this property, we use a label setting algorithm that finds the fastest paths from a start vertex $u$ during a time interval $I$. The algorithm uses labels, where each label $l = (v, s, a)$ represents a $(u, v)$-path that starts at time $s$ at vertex $u$ and arrives at time $a$ at vertex $v$. For each vertex $v \in V$, the algorithm keeps all such labels in a list $\Pi[v]$ and uses a dominance check when a new label is created to remove labels that cannot lead to optimal paths. We use the dominance relation, which is also used in [10].

**Definition 3** (Dominance)**.** *We say label $(v, s', a')$ dominates label $(v, s, a)$ if $s < s'$ and $a \geq a'$, or $s = s'$ and $a > a'$.*

A non-dominated label does not necessarily represent a fastest path. However, it might represent a prefix-path of a fastest path. On the other hand, a dominated label cannot represent a fastest path or a prefix-path of a fastest path. Therefore, all dominated labels can be deleted. Using a dominance check, Algorithm 1 only keeps labels that may lead to a fastest path. In the case of two equal labels, we only need to keep

one. Besides the label lists $\Pi[v]$, we use a priority queue $\mathcal{Q}$ containing all labels which still need to be processed. In each iteration of the while loop, we get the label $(v, s, a)$ with the smallest duration $a - s$ from the priority queue. At this point, if the algorithm discovers $v$ for the first time, the shortest duration $d^I(u, v)$ is found. The following theorem states the

**Algorithm 1**

**Input:** $\mathbf{G} = (V, \mathbf{E})$, time interval $I = [\alpha, \beta]$ and $u \in V$
**Output:** Min. duration of all $(u, v)$-paths for all $v \in V$ in $\mathbf{G}^I$

1: Initialize PQ $\mathcal{Q}$ and insert $l = (u, 0, 0)$
2: Initialize $d[v] = \infty$ for all $v \in V$, $d[u] = 0$, $F = \emptyset$
3: Initialize empty lists $\Pi[v]$ for all $v \in V$
4: **while** $\mathcal{Q}$ not empty and $|F| < |V|$ **do**
5: $\quad l = (v, s, a) \leftarrow \mathcal{Q}.extractMin$
6: $\quad$ **if** $v \notin F$ **then**
7: $\quad\quad d[v] = a - s$
8: $\quad\quad F = F \cup v$
9: $\quad$ **for** each outgoing edge $e = (v, w, t, \lambda)$ from $v$ **do**
10: $\quad\quad$ **if** $l.a \leq e.t$ and $\alpha \leq e.t \leq \beta - e.\lambda$ **then**
11: $\quad\quad\quad$ **if** $l.s = 0$ **then** $l' = (w, t, t + \lambda)$
12: $\quad\quad\quad$ **else** $l' = (w, l.s, t + \lambda)$
13: $\quad\quad\quad$ remove dominated labels from $\Pi[w]$ and $\mathcal{Q}$
14: $\quad\quad\quad$ **if** $l'$ is not dominated **then**
15: $\quad\quad\quad\quad \mathcal{Q}.insert(l')$ and $\Pi[w].add(l')$
16: **return** $d$

correctness and asymptotic running time of Algorithm 1.

**Theorem 1.** *Let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph, $u \in V$, and $I$ a time interval. And, let $\delta_m^+$ be the maximal out-degree in $\mathbf{G}$, $\pi = \min\{\hat{\tau}_-^I, |\tau_+^I(u)|\}$ and $\xi = \max\{\pi\delta_m^+, \log(|\mathbf{E}^I|)\}$. Algorithm 1 returns the durations of the fastest $(u, v)$-paths for all $v \in V$ in $\mathbf{G}^I$ in running time $\mathcal{O}(|V| + |\mathbf{E}| \cdot |\tau_+^I(u)| \cdot \xi)$.*

### *B. Computing Top-k Temporal Closeness*

Based on the fastest path algorithm presented in the previous section, we now introduce the top-$k$ temporal closeness algorithm. Given a temporal graph $\mathbf{G}$ and a time interval $I$, Algorithm 2 computes the exact top-$k$ temporal closeness values and the corresponding vertices in $\mathbf{G}^I$. If vertices share a top-$k$ temporal closeness value, the algorithm finds all of these vertices. We adapt the pruning framework introduced by Bergamini et al. [16] for the temporal closeness case. In contrast to their work, we compute the fastest paths in temporal graphs instead of BFS in static graphs and introduce upper bounds that take temporal aspects like transition and waiting times into account. We iteratively run Algorithm 1 for each $u \in V$. During the computations, we determine upper bounds of the closeness and stop the closeness computation of $u$ if we are sure that the closeness of $u$ cannot be in the set of the top-$k$ values. For calculating the upper bounds, we use two pairwise disjunctive subsets of the vertices that get updated every iteration, i.e., $F_i(u) \cup T_i(u) \subseteq V$, with

1) $F_i(u)$ contains $u$ and all vertices for which we already computed the exact temporal distance from $u$, and
2) $T_i(u)$ contains all vertices $w$ for which there is any edge $e = (v, w, t, \lambda) \in \mathbf{E}$ with $v \in F_i(u)$ and $w \notin F_i(u)$.

Together with the set $R^I(u)$ of reachable vertices, which does not change during the algorithm, we obtain the upper bound $\bar{c}_i^I(u)$ of the temporal closeness of vertex $u$ in iteration $i$ as

$$\bar{c}_i^I(u) = \sum_{v \in F_i(u)} \frac{1}{d^I(u, v)} + \sum_{v \in T_i(u)} \frac{1}{\text{lower bound for } d^I(u, v)} + \frac{|R^I(u)| - |F_i(u)| - |T_i(u)|}{\text{common lower bound } d^I \text{ for all } v \in R^I(u)}.$$

In the following, we introduce upper bounds for the temporal closeness, such that we can use Algorithm 1 to compute the duration $c^I(u)$ exactly or stop the computation if the vertex cannot achieve a top-$k$ temporal closeness value. Algorithm 2 has as input a temporal graph $\mathbf{G}$, time interval $I$, and the number of reachable vertices $r^I(u)$ for all $u \in V$. Algorithms for computing the reachability are discussed in Section IV-D. Algorithm 2 first removes all edges that are not leaving or arriving during the time interval $I$ and proceeds on $\mathbf{G}^I$. Next, the algorithm orders the vertices by decreasing number of out-going edges and starts the computation of the closeness for each vertex $v_j$ in the determined order. The intuition behind processing the vertices by decreasing out-degree is that a vertex with many out-going edges can reach many other vertices fast. The processing order of the vertices is crucial in order to stop the computations for as many vertices as soon as possible. If all vertices with the largest $k$ temporal closeness values are be processed first, the computations for the following vertices can be stopped early. For each $v_j$, the shortest durations from $v_j$ to the reachable vertices are computed using the adapted version of Algorithm 1. In each iteration of the while loop, the algorithm extracts the label $(v, s, a)$ with the smallest duration $a - s$ from the priority queue, and if $v$ is still in $T_{i-1}(v_j)$ then it found the shortest duration $d^I(v_j, v)$.

**Lemma 1.** *If during the iterations of the while loop a vertex $v$ is moved from $T_{i-1}(v_j)$ to $F_i$ (line 15 f.), the duration $d[v] = a - s$ equals the duration of a fastest $(v_j, v)$-path.*

Let $(l_1 = (w_1, s_1, a_1), \ldots, l_p = (w_p, s_p, a_p))$ be the sequence of labels returned by the *extractMin* call of the priority queue. Then the durations are non-decreasing, i.e., $a_i - s_i \leq a_h - s_h$ for $1 \leq i < h \leq p$. And, in iteration $i$, when reaching line 27, the label $l_{i+1}$ that is returned from the priority queue in iteration $i+1$ is determined, and $d_{i+1} = a_{i+1} - s_{i+1}$ is the next possible duration to any reachable vertex that is not yet in $F_i(v_j)$. Now, let $\Delta$ be the minimal temporal waiting time at a vertex $w$ over all vertices, i.e., $\Delta = \min_{w \in V}\{t_b - (t_a + \lambda_a) \mid (x, w, t_a, \lambda_a), (w, y, t_b, \lambda_b) \in \mathbf{E}^I \text{ and } t_a + \lambda_a \leq t_b\}$. Furthermore, let $\lambda_{min}$ the smallest transition time in $\mathbf{E}^I$.

**Lemma 2.** *In Algorithm 2, during the inner while loop in line 27, for each vertex $w \in T_i(v_j)$, the duration $d(v_j, w)$ is lower bounded by $d_{i+1}$. And, for each vertex $r \in R^I(v_j) \setminus (F_i(v_j) \cup T_i(v_j))$, it is $d(v_j, r) \geq d_{i+1} + \Delta + \lambda_{min}$.*

**Algorithm 2**

**Input:** A temporal graph $\mathbf{G} = (V, \mathbf{E})$, time interval $I = [\alpha, \beta]$, number of reachable vertices $r^I(u)$ for all $u \in V$
**Output:** Top-$k$ temporal closeness values and vertices

$\mathbf{E}^I \leftarrow \{(u, v, t, \lambda) \in \mathbf{E} \mid t \geq \alpha \text{ and } t + \lambda \leq \beta\}$
Determine $\lambda_{min}$ and $\Delta$
Let $v_1, \ldots, v_n \in V$ in order of decreasing out-degree
**for** $j = 1 \ldots n$ **do**
    Initialize label $l = (v_j, 0, 0)$ at vertex $v_j$
    Initialize PQ $\mathcal{Q}$ and insert $l$
    Initialize $d[v] = \infty$ for all $v \in V$
    Initialize $F_0(v_j) = \emptyset$, $T_0(v_j) = \{v_j\}$
    Initialize empty lists $\Pi[v]$ for all $v \in V$
    $d[v_j] = 0$, $i = 1$
    **while** $\mathcal{Q}$ not empty and $|F_{i-1}(v_j)| < |V|$ **do**
        $l = (v, s, a) \leftarrow \mathcal{Q}.extractMin$
        **if** $v \in T_{i-1}(v_j)$ and $v \notin F_{i-1}(v_j)$ **then**
            $d[v] = a - s$
            $T_i(v_j) = T_{i-1}(v_j) \setminus \{v\}$
            $F_i(v_j) = F_{i-1}(v_j) \cup \{v\}$
        **else** $T_i(v_j) = T_{i-1}(v_j)$
        **for** each $e = (v, w, t, \lambda) \in \mathbf{E}^I$ **do**
            **if** $l.a \leq e.t$ **then**
                **if** $l.s = 0$ **then** $l' = (w, t, t + \lambda)$
                **else** $l' = (w, l.s, t + \lambda)$
                remove dominated labels from $\Pi[w]$ and $\mathcal{Q}$
                **if** $l'$ is not dominated **then**
                    $\mathcal{Q}.insert(l')$ and $\Pi[w].add(l')$
                **if** $w \notin F_i(v_j)$ **then**
                    $T_i(v_j) = T_i(v_j) \cup \{w\}$
        update $\bar{c}(v_j)$
        **if** $\bar{c}(v_j) <$ smallest value in top $k$ values **then**
            stop computation for vertex $v_j$
    update top-$k$ results with $v_j$ and $c(v_j) = \bar{c}(v_j)$
**return** top-$k$ values and corresponding vertices

Altogether, we have the following upper bound.

**Theorem 2.** *In Algorithm 2, the temporal closeness $c_d(u)$ in iteration $i$ of the while loop is less or equal to*

$$\bar{c}(u) = \sum_{v \in F_i(u)} \frac{1}{d(u,v)} + \frac{|T_i(u)|}{d_{i+1}} + \frac{|R^I(u)| - |F_i(u)| - |T_i(u)|}{d_{i+1} + \Delta + \lambda_{min}}.$$

Finally, we discuss the running time of the algorithm.

**Theorem 3.** *Let $\delta_m^+$ be the maximal out-degree in $\mathbf{G}^I$, $\pi = \min\{\hat{\tau}_-^I, \hat{\tau}_+^I\}$, and $\xi = \max\{\pi\delta_m^+, \log(|\mathbf{E}^I|)\}$. The running time of Algorithm 2 is in $\mathcal{O}(|V|^2 + |V||\mathbf{E}| \cdot \hat{\tau}_+^I \cdot \xi)$.*

We can easily adapt this algorithm to compute the closeness for all vertices. In this case, we do not need the number of reachable vertices, and we do not need to keep track of the partitioning of the vertex set or the upper bound $\bar{c}(v_i)$.

### C. Heuristic Modification

We propose the following modification of our algorithm to obtain heuristic algorithms for computing all or top-$k$ temporal closeness values and vertices. During the computation of the closeness for a vertex $v_j$, after for a vertex $v$ the duration $d(v_j, v)$ is set and $v$ is added to the set $F_i$, the heuristic algorithms do not further consider or generate labels for $v$. Therefore, vertex $v$ is assigned the duration $a - s$ when label $(v, s, a)$ is processed, and new labels are only generated for edges $(v, w, t, \lambda)$ with $a \leq t$. All further labels $(v, a', s')$ that are returned from the priority queue in the following iterations are ignored, even if $a' < a$.

### D. The Number of Reachable Vertices

The top-$k$ temporal closeness algorithms need as input the number $r^I(u)$ of reachable vertices in $G^I$ for all $u \in V$. There are different possibilities to obtain the number of reachable vertices exactly. We can determine all $r^I(u)$ using $|V|$ times a temporal DFS or BFS, cf. [25]. If the adjacent edges stored in chronological order at each vertex it takes $\mathcal{O}(|V|^2 + |V||\mathbf{E}|)$ total running time. Alternatively, we can use a one-pass streaming algorithm similar to the earliest-arrival path algorithm in [10] with $\mathcal{O}(|V|^2 + |V||\mathbf{E}|)$ running time. In Section V-A we present an approximation for speeding up the computation of the number of reachable vertices.

### E. Comparison to the baseline algorithm

For a temporal graph $\mathbf{G} = (V, \mathbf{E})$ and a time interval $I$, the baseline algorithm first removes all edges that are not in $I$. Next, it runs the fastest path edge streaming algorithm from Wu et al. [10] for each $u \in V$ to determine the closeness $c^I(u)$. Their fastest path algorithm uses a single pass over the edges, which need to be sorted by increasing time steps. One crucial difference is that their algorithm may update the minimal duration of a vertex after it was visited the first time. This is necessary if, e.g., the last edge of the edge stream is the fastest $(u, v)$-path consisting of one edge. However, for our top-$k$ algorithms, the fastest duration between two vertices $u$ and $v$ must be determined when $v$ is discovered for the first time. For a temporal graph $\mathbf{G} = (V, \mathbf{E})$ spanning time interval $I$, let $\delta_m^-$ be the maximal in-degree in $\mathbf{G}$, and let $\pi = \min\{\delta_m^-, |\tau_+^I(u)|\}$. The running time of the edge stream algorithm for computing fastest path is in $\mathcal{O}(|V| + |\mathbf{E}| \log \pi)$ and in graphs with equal transition time for all edges in $\mathcal{O}(|V| + |\mathbf{E}|)$. For $\pi' = \min\{\delta_m^-, \hat{\tau}_+^I\}$, starting their algorithm from each vertex leads to temporal closeness algorithms with running times of $\mathcal{O}(|V|^2 + |V||\mathbf{E}| \log \pi')$, or $\mathcal{O}(|V|^2 + |V||\mathbf{E}|)$, respectively. Notice that this is faster than the worst case running time of our algorithm. However, the streaming algorithm always has to scan all edges. Even for a vertex $v$ with no out-going edge, the edge stream algorithm has to traverse over all edges because in each iteration it only knows the edges up to the point in time of the current

edge and can not rule out possible future edges incident to $v$. Our algorithm, however, stops in this case after one iteration. Therefore, our algorithm performs well on real-world data sets.

## V. Approximations for Equal Transition Times

We lift two approximations from the static to the temporal domain. The first one is for the number of reachable vertices and is based on the estimation framework introduced by Cohen et al. [30]. The second one approximates the temporal closeness for all vertices and adapts the undirected static closeness approximation from Wang et al. [20]. Both algorithms are applicable if the temporal graph has equal transition times for all edges. This is often the case for social and contact networks. First, we introduce the concept of the temporal transpose of a temporal graph to generalize the algorithms for directed temporal graphs with equal transition times.

**Definition 4.** *For a temporal graph $\mathbf{G} = (V, \mathbf{E})$, we call $\mathcal{I}(\mathbf{G}) = (V, \tilde{\mathbf{E}})$ with edge set $\tilde{\mathbf{E}} = \{(v, u, t_{max} - t, \lambda) \mid (u, v, t, \lambda) \in \mathbf{E}\}$ and $t_{max} = \max\{t + \lambda \mid (u, v, t, \lambda) \in \mathbf{E}\}$ the* temporal transpose *of* $\mathbf{G}$.

The following lemma is the basis for the approximations.

**Lemma 3.** *Let $\mathbf{G}$ be a temporal graph in which all edges have equal transition times, $\mathcal{I}(\mathbf{G})$ its temporal transpose, and $u, v \in V(\mathbf{G})$. For each $(u, v)$-path $P_{uv}$ with duration $d(P_{uv}) = d_P$ in $\mathbf{G}$ there exists a $(v, u)$-path $P_{vu}$ with duration $d(P_{vu}) = d_P$ in $\mathcal{I}(\mathbf{G})$ and vice versa.*

### A. Approximation for the Number of Reachable Vertices

We propose an approximation algorithm for the number of reachable vertices based on the estimation framework introduced by Cohen et al. [30]. The non-weighted version processes the following way. For two sets $X$ and $Y$ let $S : Y \to 2^X$. Furthermore, let $L$ be an oracle that if it is presented a random permutation $r : X \to \{1, \dots, |X|\}$ it returns a mapping $l : Y \to X$ such that for all $y \in Y$, $l(y) \in S(y)$ and $r(l(y)) = \min_{x \in S(y)} r(x)$. In $h$ rounds, a uniformly randomly chosen value from the interval $[0, 1]$ is assigned to each $x \in X$, i.e., the rank $\mathcal{R}_i(x)$ in round $i$. In each round, the sorted ranking induces a permutation on $X$ that is presented to the oracle $L$, which returns the mapping $l_i : Y \to X$. For an estimator $\hat{s}(y) = \frac{h}{\sum_{i=1}^{n} \mathcal{R}_i(l_i(y))} - 1$ the following holds.

**Theorem 4.** *(Cohen et al. [30]) Let $0 < \epsilon < 1$. Then $P(||S(y)| - \hat{s}(y)| \geq \epsilon |S(y)|) = e^{-\Omega(\epsilon^2 \cdot h)}$.*

For a temporal graph $\mathbf{G}$ and a time interval $I$, Algorithm 3 computes the estimates $\hat{r}^I(u)$ of the number of reachable vertices $r^I(u)$, for all $u \in V$. Given $\mathbf{G}$, $I$ and $h \in \mathbb{N}$, our algorithm first computes the temporal transpose $\mathcal{I}(\mathbf{G}^I)$. Notice that we only need to add edges $(u, v, t, \lambda) \in \mathbf{E}$ with $t \in I$ to the temporal transpose $\mathcal{I}(\mathbf{G}^I)$. The algorithm then proceeds with the following steps on $\mathcal{I}(\mathbf{G}^I)$. In $h$ rounds, we first assign a uniformly randomly chosen value from the interval $[0, 1]$ to each vertex, i.e., the rank $\mathcal{R}(v)$ of the vertex $v$. Let $v_1, \dots, v_n$ be the vertices ordered by increasing rank. The algorithm then determines for each $v_i$ the set of reachable vertices from $v_i$, i.e., $R(v_i)$, and sets the mapping $l_j(v)$ of all newly found $v \in R(v_i)$ to $\mathcal{R}(v_i)$.

**Algorithm 3**

**Input:** A temporal graph $\mathbf{G} = (V, \mathbf{E})$, $h \geq 1$, time interval $I$
**Output:** Estimates of $\hat{r}^I(u)$ for all $u \in V$

```
Calculate I(G^I)
for j = 1, ..., h do
    G̃ ← I(G^I)
    assign R(v) for all v ∈ V uniformly from [0, 1]
    sort v_1, ..., v_n by increasing rank
    for i = 1, ..., n do
        compute R(v_i) and remove used edges in G̃
        set l_j(v) = R(v_i) for all non-marked v ∈ R(v_i)
        mark all v ∈ R(v_i)
for v ∈ V do
    r̂^I(v) = h / (Σ_{j=0}^{h} l_j(v)) − 1
```

**Theorem 5.** *Let $\mathbf{G}$ be a temporal graph, $I$ a time interval and $h \in \mathbb{N}$. Algorithm 3 approximates the reachability $\hat{r}^I(u)$ for each $u \in V$ and $0 < \epsilon < 1$ such that*

$$P\big(|r^I(u) - \hat{r}^I(u)| \geq \epsilon r^I(u)\big) = e^{-\Omega(\epsilon^2 \cdot h)}.$$

Computing $\mathcal{I}(\mathbf{G}^I)$ takes $\mathcal{O}(|\mathbf{E}|)$ and assigning $\mathcal{R}(v)$ for all $v \in V$ is possible in $\mathcal{O}(|\mathbf{E}^I| + |V|)$. Sorting is possible in expected $\mathcal{O}(|V|)$, and computing $R(v)$ takes $\mathcal{O}(|\mathbf{E}^I| + |V|)$.

**Theorem 6.** *The running time of Algorithm 3 is in $\mathcal{O}(|\mathbf{E}| + h \cdot (|\mathbf{E}^I| + |V|))$.*

After approximating the reachability $\hat{r}^I(u)$ for all $u \in V$, we can use them for the top-$k$ temporal closeness algorithms. However, notice that underestimating the number of reachable vertices may stop the computation for a vertex that does have a top-$k$ temporal closeness value.

### B. Approximation of the Temporal Closeness

We lift the randomized algorithm for undirected, static graphs from Wang et al. [20] to the temporal domain. Algorithm 4 computes an approximation of the normalized temporal closeness for each vertex in a directed temporal graph with equal transition times. Given a temporal graph $\mathbf{G}$, a time interval $I$ and sampling size $h \in \mathbb{N}$, the algorithm first computes the temporal transpose $\mathcal{I}(\mathbf{G}^I)$, and then samples $h$ vertices $v_1, \dots, v_h$. For each vertex $v_i$ it determines the shortest durations to all vertices $w \in V \setminus \{v_i\}$ in $\mathcal{I}(\mathbf{G}^I)$. Due to Lemma 3, we can estimate the closeness of a vertex $u$ in $\mathbf{G}^I$ by averaging the distances $d(v, u)$ found in $\mathcal{I}(\mathbf{G}^I)$.

**Theorem 7.** *Let $\mathbf{G}$ be a temporal graph, $I$ a time interval and $h \in \mathbb{N}$. Algorithm 4 approximates the normalized temporal closeness $\hat{c}_n^I(u)$ for each vertex $u \in V$, such that for $h = \log n \cdot \epsilon^{-2}$ the probability $P\big(|\hat{c}_{\mathrm{n}}^I(u) - c_{\mathrm{n}}^I(u)| \geq \epsilon\big) \leq \frac{2}{n^2}$. The running time is in $\mathcal{O}(|\mathbf{E}| + h \cdot (|V| + |\mathbf{E}^I|))$.*

**Algorithm 4**

**Input:** Temporal graph $\mathbf{G} = (V, \mathbf{E})$, $h$, time interval $I$
**Output:** Estimates $\hat{c}_{\mathrm{n}}^{I}(u)$ for $u \in V$

```
Calculate I(G^I)
Sample v_1, ..., v_h from V
for i = 1 ... h do
    Compute d(v_i, w) for all w ∈ V
for u ∈ V do
    Let ĉ_n^I(u) = (1/h) Σ_{i=1}^{h} 1/d(v_i, u)
```

## VI. Experiments

We address the following questions:

- **Q1** How do the running times of the algorithms for the top-$k$ temporal closeness, the temporal closeness for all vertices, and the baseline compare to each other?
- **Q2** How much does the reachability estimation speed up the computation of the top-$k$ temporal closeness, and how good are the results?
- **Q3** How do the heuristics compare to the top-$k$ and the exact algorithms in terms of running time and solution quality?
- **Q4** How well does the sampling algorithm in terms of running time and approximation quality perform?
- **Q5** How does the temporal closeness compare to the static closeness in the aggregated graph?

### A. Data Sets

We used the following real-world temporal graph data sets: 1. *Infectious*: A data set from the *SocioPatterns* project.[1] The Infectious graph represents face-to-face contacts between visitors of the exhibition *Infectious: Stay Away* [31]. 2. *Arxiv*: An authors collaboration network from the *arXiv's High Energy Physics - Phenomenology (hep-ph)* section [32]. Vertices represent authors and edges collaborations. The time stamp of an edge is the publication date. 3. *Facebook*: This graph is a subset of the activity of a Facebook community over three months and contains interactions in the form of wall posts [33]. 4. *Prosper*: A network based on a personal loan website.[2] Vertices represent persons, and each edge a loan from one person to another person. 5. *WikipediaSG*: The network is based on the *Wikipedia* network. Each vertex represents a *Wikipedia* page, and each edge a hyperlink between two pages [34]. 6. *WikiTalk*: Vertices represent users and edges edits of a user's talk page by another user [35]. 7. *Digg* and 8. *FlickrSG*: Digg and Flickr are social networks in which vertices represent persons and edges friendships. The times of the edges indicate when the friendship was formed [36], [37]. For *WikipediaSG* and *FlickrSG*, we chose a random time interval in size of ten percent of the total time span. For all others, the time interval spans over all edges. Table I gives an overview over the statistics of the networks (for*WikipediaSG* and *FlickrSG* the statistics are for the subgraph $\mathbf{G}^I$). The transition times are equal for all edges in all data sets.

[1] http://www.sociopatterns.org
[2] www.prosper.com

TABLE I: Statistics and properties of the real world networks, with $\delta_m^-$ (resp. $\delta_m^+$) being the maximal in-degree (resp. out-degree), and $|E_s|$ the number of edges in the aggregated graph.

| Data set | Properties | | | | | |
|---|---|---|---|---|---|---|
| | $\lvert V\rvert$ | $\lvert\mathbf{E}\rvert$ | $\delta_m^+$ | $\delta_m^-$ | $\lvert\mathcal{T}(\mathbf{G})\rvert$ | $\lvert E_s\rvert$ |
| Infectious | 10 972 | 415 912 | 457 | 544 | 76 943 | 52761 |
| Arxiv | 28 093 | 4 596 803 | 9 301 | 3 962 | 2 337 | 3 148 447 |
| Facebook | 63 731 | 817 035 | 998 | 219 | 333 924 | 817 035 |
| Prosper | 89 269 | 3 394 978 | 9 436 | 1 071 | 1 259 | 3 330 224 |
| WikipediaSG | 208 142 | 810 702 | 1 574 | 4 939 | 223 | 810 702 |
| WikiTalk | 225 749 | 1 554 698 | 113 867 | 36 413 | 1 389 632 | 565 476 |
| Digg | 279 630 | 1 731 653 | 995 | 12 038 | 6 864 | 1 731 653 |
| FlickrSG | 323 181 | 1 577 469 | 3 153 | 2 128 | 20 | 1 577 469 |

### B. Experimental Protocol and Algorithms

All experiments were conducted on a workstation with an AMD EPYC 7402P 24-Core Processor with 3.35GHz and 256GB of RAM running Ubuntu 18.04.3 LTS. We used GNU CC Compiler 7.4.0 with the flag `--O2`, and implemented the following algorithms in C++:

- TC-Top-$k$ is our top-$k$ temporal closeness algorithm.
- TC-All is our temporal closeness algorithm for computing the exact values for all vertices.
- TC-Approx is our temporal closeness approximation.
- EdgeStr is the temporal closeness algorithm based on the edge stream algorithm for equal transition times [10].

For TC-Top-$k$ and TC-All we also implemented the heuristic (Heuristic) described in Section IV-C. Our source code and the data sets are available at https://gitlab.com/tgpublic/tgcloseness.

### C. Results and Discussion

**Q1** Table II shows the running times of the exact temporal closeness algorithms. We computed the numbers of reachable vertices exactly and set $k \in \{1, 10, 100, 1000\}$ for the top-$k$ algorithms. For all data sets, the top-$k$ algorithms need significantly less running time than EdgeStr and TC-All. For the *Infectious* data set, the running time can be reduced by a factor between 10 ($k = 1000$) and 17 ($k = 1$) compared to EdgeStr. Figure 2 shows the reduction of the running time in percent compared to EdgeStr. For *Prosper* and *WikipediaSG* the running times compared to EdgeStr are decreased by more than $85\%$ and $72\%$ for each $k \in \{1, 10, 100, 1000\}$. In case of *Facebook* the decrease is between $62\%$ ($k = 1000$) and $72\%$ ($k = 1$). Also, for the other data sets, considerable improvements are reached, i.e., for $k = 1$ and $k = 10$, the running time is reduced by at least $60\%$. With increasing $k$, the running time also increases because the upper bounds calculated during the run of TC-Top-$k$ converge slower at the lower end of the top-$k$ values. Our exact algorithm TC-All is faster than EdgeStr for the *Infectious, Facebook, Prosper* and *WikipediaSG* data sets. For the *Infectious* data set our exact algorithm is more than ten times faster, and for the Prosper

TABLE II: Running times in seconds for the exact temporal closeness algorithms.

| Data set | Algorithm | | | | | |
|---|---|---|---|---|---|---|
| | TC-TOP-1 | TC-TOP-10 | TC-TOP-100 | TC-TOP-1000 | TC-ALL | EDGESTR |
| *Infectious* | 1.19 | 1.26 | 1.39 | 1.77 | 1.77 | 21.15 |
| *Arxiv* | 339.48 | 367.10 | 546.76 | 1076.83 | 1 876.88 | 1 488.43 |
| *Facebook* | 107.18 | 108.57 | 113.82 | 146.29 | 213.69 | 389.84 |
| *Prosper* | 264.70 | 267.09 | 269.98 | 288.76 | 465.83 | 1 917.84 |
| *WikipediaSG* | 394.10 | 400.02 | 400.88 | 439.76 | 768.24 | 1622.04 |
| *WikiTalk* | 1 190.07 | 1 314.41 | 1 765.39 | 2 899.65 | 3 468.53 | 3 158.30 |
| *Digg* | 2 334.34 | 2 339.54 | 2 422.64 | 3 697.73 | 8 541.81 | 5 741.24 |
| *FlickrSG* | 4 212.86 | 4 228.94 | 4 299.79 | 4 792.36 | 14 162.06 | 10 666.30 |

TABLE III: Running times in seconds for the algorithms using the reachability approximation with $h = 5$. The running times are the average and standard deviations over ten repetitions.

| Data set | Algorithm with Reachability Approximation | | |
|---|---|---|---|
| | TC-TOP-10 | TC-TOP-100 | TC-TOP-1000 |
| *Infectious* | 1.30±0.01 | 1.45±0.01 | 1.88±0.01 |
| *Arxiv* | 117.25±0.34 | 310.36±0.53 | 1075.13±5.10 |
| *Facebook* | 33.81±2.83 | 41.14±4.68 | 77.32±5.39 |
| *Prosper* | 224.33±3.09 | 227.56±0.48 | 246.65±1.49 |
| *WikipediaSG* | 346.95±25.20 | 340.39±2.31 | 369.37±2.43 |
| *WikiTalk* | 1 010.12±5.86 | 1 472.38±10.63 | 2 678.99±4.31 |
| *Digg* | 957.80±6.19 | 1 056.20±5.55 | 2 353.76±2.78 |
| *FlickrSG* | 965.75±2.03 | 1 014.69±1.78 | 1 510.15±2.73 |

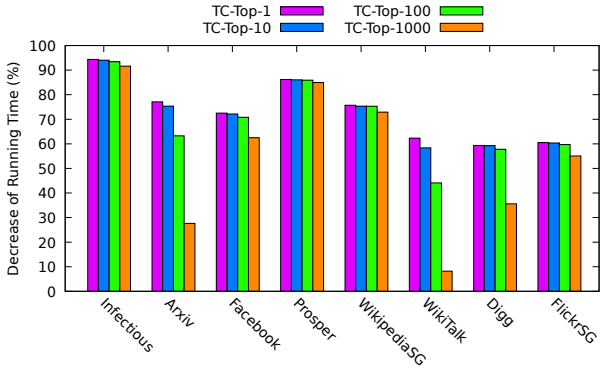



Fig. 2: Decrease in running time of the top-$k$ algorithms in percent compared to baseline EDGESTR.

data set it is more than four times faster than the baseline. In case of the *WikipediaSG* data set, TC-ALL is more than twice as fast, and it decreases the running time for *Facebook* by 45%. For the *Arxiv, Digg, WikiTalk* and the *FlickrSG* data sets, EDGESTR is faster than TC-ALL. The reason is a higher number of labels per vertex that are generated during the iterations of Algorithm 2. This leads to a worse performance of TC-ALL for these data sets.

**Q2** Table III shows the average running times and standard deviations for our top-$k$ temporal closeness algorithms using the reachability approximation over ten repetitions. We approximate $\hat{r}(u)$ for all $u \in V$ using Algorithm 4 with parameter $h = 5$. For the relatively small *Infectious* network, the approximation cannot improve the running time. For all

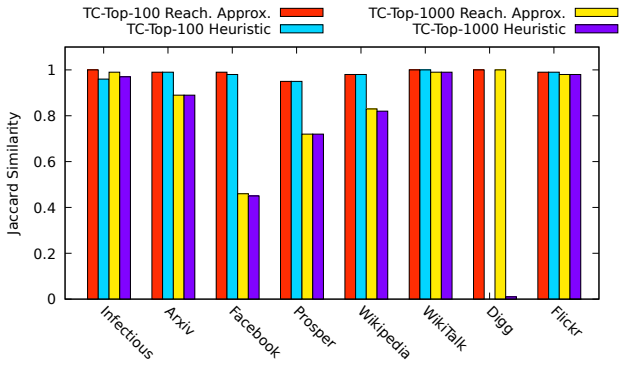



Fig. 3: Mean Jaccard similarity between the exact top-$k$ temporal closeness sets and the sets calculated using the reachability approximation with $h = 5$, and the Jaccard similarity between the exact top-$k$ sets and the ones computed using the heuristic top-$k$ temporal closeness algorithms.

other data sets, the reachability approximation reduces the running time. Notice the large improvements for the *Facebook*, *Digg*, and *FlickrSG* data sets of at least 45%, 36%, and 68%, respectively. Even though there are approximation guarantees for the reachability approximation (Theorem 6), we are not able to give a guarantee for finding the top-$k$ sets. The reason is that underestimating the number of reachable vertices may stop the computation of a vertex that has a top-$k$ temporal closeness value. To evaluate the solution quality, we measure the *Jaccard similarity* between top-$k$ vertex sets computed using the algorithms with the reachability approximation and the exact top-$k$ sets. The Jaccard similarity between two sets $A$ and $B$ is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Values closer to one mean a higher similarity of the sets. For $k = 10$ most data sets have perfect similarity of one, only the *Infectious* data set has for $k = 10$ a similarity of 0.96. Figure 3 shows the average Jaccard similarities for $k = 100$ and $k = 1000$. The standard deviation is zero for all experiments, but for *Infectious* and $k = 1$ where it is 0.08. In case of $k = 1000$ for *Facebook* and *Prosper* the similarities are lowest with 0.42 and 0.72, respectively. For the *Infectious, WikiTalk, Digg* and *Flickr* data sets, we have high similarities between 0.98 and 1.00.

**Q3** The running times using the heuristic algorithms are lower than those of the exact algorithms for all data sets, see

TABLE IV: Running times in seconds for the heuristic and sampling temporal closeness algorithms.

| Data set | Algorithm | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | TC-TOP-1 HEURISTIC | TC-TOP-10 HEURISTIC | TC-TOP-100 HEURISTIC | TC-TOP-1000 HEURISTIC | TC-ALL HEURISTIC | TC-APPROX $p = 0.1$ | TC-APPROX $p = 0.2$ | TC-APPROX $p = 0.5$ |
| *Infectious* | 0.64 | 0.65 | 0.67 | 0.80 | 0.73 | 0.25±0.01 | 0.47±0.01 | 1.12±0.01 |
| *Arxiv* | 117.73 | 119.72 | 140.48 | 251.02 | 384.71 | 410.78±5.16 | 862.46±10.71 | 2 067.93±7.52 |
| *Facebook* | 98.80 | 99.42 | 101.94 | 119.09 | 132.02 | 28.22±0.37 | 56.61±1.00 | 131.58±1.20 |
| *Prosper* | 95.12 | 94.25 | 96.52 | 106.57 | 193.22 | 48.83±0.94 | 98.00±0.62 | 228.96±0.48 |
| *WikipediaSG* | 335.76 | 350.89 | 335.05 | 365.05 | 622.41 | 84.51±0.67 | 170.38±2.92 | 353.54±4.23 |
| *WikiTalk* | 887.10 | 895.10 | 911.08 | 1 057.72 | 1 523.15 | 1 573.23±10.06 | 3 173.40±27.47 | 7 744.86±35.24 |
| *Digg* | 2 066.09 | 2 072.70 | 2 049.86 | 2 273.61 | 5 044.37 | 1 682.70±27.82 | 3 356.85±42.60 | 8 259.91±62.70 |
| *FlickrSG* | 3 915.19 | 3 919.05 | 3 928.20 | 4 049.44 | 7 375.41 | 1 656.13±7.58 | 3 288.93±27.47 | 6 946.62±44.30 |

TABLE V: Mean approximation error and standard deviation of TC-APPROX with sampling sizes $h = p \cdot |V|$.

| Data set | TC-APPROX | | |
|---|---|---|---|
| | $p = 0.1$ | $p = 0.2$ | $p = 0.5$ |
| *Infectious* | 0.58±0.06 | 0.58±0.07 | 0.42±0.09 |
| *Arxiv* | 0.55±0.10 | 0.58±0.09 | 0.39±0.35 |
| *Facebook* | 0.44±0.11 | 0.42±0.17 | 0.30±0.10 |
| *Prosper* | 0.45±0.10 | 0.43±0.12 | 0.31±0.12 |
| *WikipediaSG* | 0.30±0.09 | 0.28±0.14 | 0.20±0.16 |
| *WikiTalk* | 0.08±0.08 | 0.07±0.10 | 0.05±0.04 |
| *Digg* | 0.18±0.00 | 0.16±0.01 | 0.15±0.01 |
| *FlickrSG* | 0.35±0.14 | 0.34±0.10 | 0.27±0.10 |

Table IV. For *Arxiv* the running time is decreased by up to 79% compared TC-ALL with having an average relative deviation from the exact closeness values of less than $4 \cdot 10^{-6}$. For *Flickr* the average deviation is 0.02%. The highest error is for *Digg* with 22.43%. Figure 3 shows the Jaccard similarity between top-100 and top-1000 vertex sets computed using the heuristic and the exact sets. For all but the *Digg* data set the results for $k = 10$ are equal and for $k = 100$ the similarity is over 0.95. For $k = 1000$ the variance is much higher, for *WikiTalk* we have 0.99 and on the other end *Facebook* with 0.45. The similarity scores, beside for the *Digg* data set, are similar to the ones for using the reachability approximation (see Figure 3). The heuristic fails for the *Digg* dataset with a Jaccard similarity of 0 ($k = 100$) and 0.002 ($k = 1000$) due to the high error rate of the estimated closeness values.

**Q4** We ran the Algorithm 4 ten times for each data set with the sampling sizes $h = p \cdot n$ with $p \in \{0.1, 0.2, 0.5\}$. Table IV reports the average running times and the standard deviations. Theorem 7 shows that TC-APPROX with high probability computes closeness values with a small additive error. For a fixed error $\epsilon$, the sample size $h$ grows logarithmically in $n$, and the approximation is most suitable for large data sets. Choosing a large enough $h$ for data sets with not many vertices is not feasible. For $p = 0.5$ the running times for *Arxiv, WikiTalk* and *Digg* exceed the running times using the exact algorithms. As expected, with increased sampling sizes, the approximation error is reduced. Table V shows the approximation errors for the data sets. For $p = 0.1$ an approximation error of only 8% is reached for *WikiTalk* while using less than half the running time of EDGESTR.

TABLE VI: Jaccard similarity between the top-$k$ temporal closeness vertices and static closeness vertices in the aggregated graph.

| Data set | Top-$k$ size | | |
|---|---|---|---|
| | $k = 10$ | $k = 100$ | $k = 1000$ |
| *Infectious* | 0.25 | 0.18 | 0.26 |
| *Arxiv* | 0.81 | 0.41 | 0.41 |
| *Facebook* | 0.00 | 0.00 | 0.01 |
| *Prosper* | 0.33 | 0.45 | 0.41 |
| *WikipediaSG* | 0.00 | 0.15 | 0.39 |
| *WikiTalk* | 0.42 | 0.45 | 0.63 |
| *Digg* | 0.00 | 0.00 | 0.00 |
| *FlickrSG* | 0.42 | 0.34 | 0.25 |

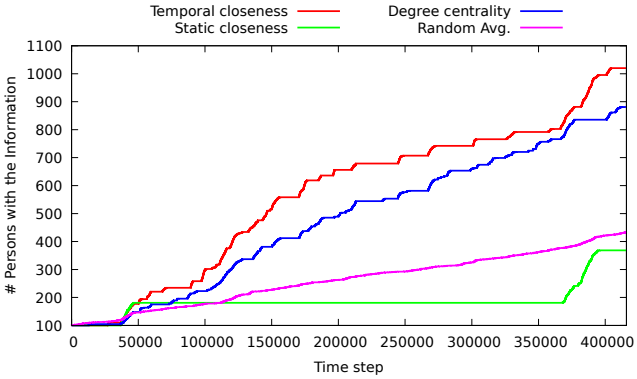



Fig. 4: Comparing different centrality measures for choosing the seed of a dissemination process.

**Q5** We measured the Jaccard similarity between the temporal and static top-$k$ closeness values. Table VI shows that the similarity is very low (for *Facebook* and *Digg* even zero). Furthermore, we simulated an information diffusion process in the *Infectious* network. This information could be, e.g., (fake) news, a viral campaign, or some infectious disease. For selecting an initial seed of vertices, we used the top-$k$ temporal closeness vertices, the top-$k$ highest degree vertices, top-$k$ static closeness vertices, and $k$ randomly chosen vertices. The static closeness has been computed on the aggregated graph $A(\mathbf{G})$. We set $k = 100$ and evaluated the spread over time. Each person that has the information propagates it with a probability of 10%. We repeated the experiments ten times. Figure 4 shows the results. When using the temporal top-$k$

vertices as seeds, the information is distributed to the highest number of people. Using the vertices with the highest degree as seeds lead to the second-best results. The static closeness results cannot compete because the temporal restrictions are not respected during the computation of the static closeness vertices, hence not represented in the top-$k$ static closeness vertices. The randomly chosen vertices perform better than static closeness.

## VII. Conclusion

We introduced algorithms for computing temporal closeness in temporal graphs. The basis for our algorithms is a new fastest path algorithm using a label setting strategy and which might be of interest in its own. Our top-$k$ temporal closeness algorithms improved the running times for all real-world data sets. We introduced simple heuristic modifications that proved to be useful for most of the data sets. Furthermore, we adapted two randomized approximation algorithms, one for estimating the number of reachable vertices and one for approximating the closeness for all vertices. Also, the reachability approximation is interesting in itself. Both randomized approaches lead to significant speed-ups, and the sampling algorithm for temporal closeness has only a small additive error with high probability. Our approaches allow us to efficiently find the most relevant vertices and their temporal closeness in temporal graphs.

## Acknowledgment

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

## References

[1] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[2] D. Braha and Y. Bar-Yam, "Time-dependent complex networks: Dynamic centrality, dynamic motifs, and cycles of social interactions," in *Adaptive Networks*. Springer, 2009, pp. 39–50.

[3] P. Holme, "Modern temporal network theory: a colloquium," *The European Physical Journal B*, vol. 88, no. 9, p. 234, 2015.

[4] G. Kossinets and D. J. Watts, "Empirical analysis of an evolving social network," *Science*, vol. 311, no. 5757, pp. 88–90, 2006.

[5] N. Masuda and P. Holme, "Detecting sequences of system states in temporal networks," *Scientific Reports*, vol. 9, no. 1, pp. 1–11, 2019.

[6] G. Palla, A.-L. Barabási, and T. Vicsek, "Quantifying social group evolution," *Nature*, vol. 446, no. 7136, pp. 664–667, 2007.

[7] R. K. Pan and J. Saramäki, "Path lengths, correlations, and centrality in temporal networks," *Physical Review E*, vol. 84, no. 1, p. 016105, 2011.

[8] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard, "Time-varying graphs and social network analysis: Temporal indicators and metrics," *arXiv preprint arXiv:1102.0629*, 2011.

[9] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia, "Analysing information flows and key mediators through temporal centrality metrics," in *Proc. 3rd Workshop on Social Network Systems*, 2010, pp. 1–6.

[10] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.

[11] M. Marchiori and V. Latora, "Harmony in the small-world," *Physica A: Statistical Mechanics and its Applications*, vol. 285, no. 3-4, pp. 539–546, 2000.

[12] J. Tang, I. Leontiadis, S. Scellato, V. Nicosia, C. Mascolo, M. Musolesi, and V. Latora, "Applications of temporal graph metrics to real-world networks," in *Temporal Networks*. Springer, 2013, pp. 135–159.

[13] K. Das, S. Samanta, and M. Pal, "Study on centrality measures in social networks: a survey," *Social Network Analysis and Mining*, vol. 8, no. 1, p. 13, 2018.

[14] A. Landherr, B. Friedl, and J. Heidemann, "A critical review of centrality measures in social networks," *Business & Information Systems Engineering*, vol. 2, no. 6, pp. 371–385, 2010.

[15] F. A. Rodrigues, "Network centrality: an introduction," in *A Mathematical Modeling Approach from Nonlinear Dynamics to Complex Systems*. Springer, 2019, pp. 177–196.

[16] E. Bergamini, M. Borassi, P. Crescenzi, A. Marino, and H. Meyerhenke, "Computing top-$k$ closeness centrality faster in unweighted graphs," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 5, pp. 53:1–53:40, 2019.

[17] E. Yan and Y. Ding, "Applying centrality measures to impact analysis: A coauthorship network analysis," *Journal of the American Society for Information Science and Technology*, vol. 60, no. 10, pp. 2107–2118, 2009.

[18] J. Zhan, S. Gurung, and S. P. K. Parsa, "Identification of top-k nodes in large networks using Katz centrality," *J. Big Data*, vol. 4, no. 1, pp. 1–19, 2017.

[19] P. Bisenius, E. Bergamini, E. Angriman, and H. Meyerhenke, "Computing top-k closeness centrality in fully-dynamic graphs," in *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX*. SIAM, 2018, pp. 21–35.

[20] D. Eppstein and J. Wang, "Fast approximation of centrality," in *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*. ACM/SIAM, 2001, pp. 228–229.

[21] K. Okamoto, W. Chen, and X.-Y. Li, "Ranking of closeness centrality for large-scale social networks," in *Frontiers in Algorithmics, Second Annual International Workshop, FAW*. Springer, 2008, pp. 186–195.

[22] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck, "Computing classic closeness centrality, at scale," in *Proceedings of the second ACM conference on Online social networks*, 2014, pp. 37–50.

[23] K. L. Cooke and E. Halsey, "The shortest route through a network with time-dependent internodal transit times," *Journal of Mathematical Analysis and Applications*, vol. 14, no. 3, pp. 493–498, 1966.

[24] B. B. Xuan, A. Ferreira, and A. Jarry, "Computing shortest, fastest, and foremost journeys in dynamic networks," *International Journal of Foundations of Computer Science*, vol. 14, no. 02, pp. 267–285, 2003.

[25] S. Huang, J. Cheng, and H. Wu, "Temporal graph traversals: Definitions, algorithms, and applications," *CoRR*, vol. abs/1401.1919, 2014.

[26] 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*, pp. 1–16, 2019.

[27] U. Brandes, "A faster algorithm for betweenness centrality," *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[28] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal Networks*. Springer, 2013, pp. 15–40.

[29] H. Kim and R. Anderson, "Temporal node centrality in complex networks," *Physical Review E*, vol. 85, no. 2, p. 026107, 2012.

[30] E. Cohen, "Size-estimation framework with applications to transitive closure and reachability," *Journal of Computer and System Sciences*, vol. 55, no. 3, pp. 441–453, 1997.

[31] L. Isella, J. Stehl, A. Barrat, C. Cattuto, J.-F. Pinton, and W. Van den Broeck, "What's in a crowd? Analysis of face-to-face behavioral networks," *Journal of Theoretical Biology*, vol. 271, no. 1, pp. 166–180, 2011.

[32] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, pp. 2–es, 2007.

[33] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the 2nd ACM workshop on Online social networks*, 2009, pp. 37–42.

[34] A. E. Mislove, "Online social networks: measurement, analysis, and applications to distributed information systems," Ph.D. dissertation, Rice University, 2009.

[35] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Governance in social media: A case study of the wikipedia promotion process," in *Fourth International AAAI Conference on Weblogs and Social Media*, 2010.

[36] T. Hogg and K. Lerman, "Social dynamics of digg," *EPJ Data Science*, vol. 1, no. 1, p. 5, 2012.

[37] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proceedings of the first workshop on Online social networks*, 2008, pp. 25–30.