Computing top-k temporal closeness in temporal networks

Lutz Oettershagen¹ · Petra Mutzel¹

Received: 29 January 2021 / Revised: 27 November 2021 / Accepted: 3 December 2021 /

Published online: 8 January 2022

© The Author(s) 2021

Abstract

The closeness centrality of a vertex in a classical static graph is the reciprocal of the sum of the distances to all other vertices. However, networks are often dynamic and change over time. Temporal distances take these 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 heuristic modifications that perform well on real-world data sets and drastically reduce 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 the considered temporal networks.

Keywords Temporal graphs · Temporal closeness · Centrality · Top- k ranking

1 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, have gained increasing attention [5, 18, 26, 32, 40]. 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 (called the *availability time*), and edge traversal costs a strictly positive amount of time (called the *transition time*).

✉ Lutz Oettershagen
lutz.oettershagen@cs.uni-bonn.de

¹ Institute of Computer Science, University of Bonn, Bonn, Germany

Important examples are human and social contact networks, communication networks, and transportation networks. The transition times in transportation networks correspond to the times needed for spatial movements, e.g., the duration of flights or the travel time of buses between the bus stops. For communication networks, the transition times depend on the type of communication, e.g., email communication uses unit transition times. In a temporal network modeling phone usage, the transition time of an edge may be the duration of a phone call between two users. In the case of unit transition times, 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 [7, 18, 52]. Modifications of static closeness to temporal closeness using temporal path and *temporal distances* have been suggested [11, 28, 41, 45, 49, 52]. One of these temporal distances is the shortest duration, i.e., the duration of a *fastest path* between two nodes. Here, we consider the *harmonic temporal closeness* of a vertex. It is defined as the sum of the reciprocals of the *durations* of the fastest paths to all other vertices. Furthermore, we define the *harmonic temporal in-closeness* as the sum of the reciprocals of the durations of the fastest paths arriving at a vertex. Harmonic closeness for non-connected static graphs was introduced in [31]. The reason for using the harmonic variant for temporal closeness and in-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 with a high temporal closeness can be expected to be of high importance for the transportation or dissemination process [48, 49]. In general, high temporal closeness vertices in temporal networks can differ from high static closeness and high degree vertices. For example, in the temporal graph shown in Fig. 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 vertex a can only reach its direct neighbors in the temporal graph, while vertex e can reach all other vertices. Computing the exact temporal closeness values of all vertices can be costly because it demands finding a fastest temporal path between each pair of vertices. However, often knowing the top- k important vertices is sufficient. Our contributions are:

1. We propose an algorithm for computing the top- k harmonic temporal closeness values and the corresponding vertices 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 two 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 all our data sets.
3. For temporal graphs with equal transition times, we introduce the temporal transpose, which is a generalization of reversing the edges in a directed static graph. This transformation enables our algorithms to compute the temporal in-closeness. Furthermore, using the temporal transpose, we adapt a stochastic approximation for the transitive closure of

(c) The representation of 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 Example of an undirected temporal graph, its aggregated static graph, and its interpretation as a sequence of static graphs

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. [52]. Our approaches decrease the running times for all data sets significantly.

Differences to the conference version The main differences and additions to the conference version [38] are the following. We added the definition of temporal in-closeness. Our new results show that with the temporal transpose, our algorithms can efficiently compute the temporal in-closeness. Furthermore, we introduce two new heuristics with detailed descriptions and analyses. Our experimental evaluation for the two new heuristics shows better results for both of them compared to the one in the conference version. Finally, we extended our experiments section with a comparison between temporal closeness, static closeness, degree centrality, and reachability.

2 Related work

Comprehensive overviews and introductions of centrality approaches, including closeness centrality, are provided, e.g., in [12,27,43,46]. The problem of ranking vertices and selecting

the k most influential vertices has been widely studied, see, e.g., [3,53,54]. Bergamini et al. [3] 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 Sect. 4. 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 [3] 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 [4] 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 [13] 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 also is not directly applicable to the temporal case. In Sect. 5, we introduce a transformation of temporal graphs to an inverse representation such that the algorithm can be used to approximate temporal closeness. Based on [13], Okamoto et al. [39] 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. [9] combined the sampling approach and a pivoting technique for approximating the closeness of all vertices.

An early overview of different dynamic graph models is given by Harary and Gupta [15]. More recent and comprehensive introductions to temporal graphs, including overviews of different temporal centrality measures, are provided in, e.g., [18,28,45]. Michail [33] gives an algorithmic oriented introduction to temporal graphs. The early work on fastest paths by Cooke and Halsey [10] suggests to iteratively compute T distance matrices for each vertex, where T is a time-dependent parameter chosen beforehand. Kempe et al. [21] discuss time-respecting paths and related connectivity problems. Xuan et al. [7] 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 [19], 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 , containing information about the fastest paths from v to the other vertices. The authors of [36] consider bicriteria temporal paths in weighted temporal graphs, where each edge has an additional cost value. They propose an algorithm that enumerates all efficient paths with polynomial-time delay.

Wu et al. [52] introduce streaming algorithms operating on the temporal edge stream in which the edges arrive with non-decreasing time stamps. They discuss finding fastest, shortest, latest departure, and earliest arrival paths and suggest using their algorithms to compute temporal closeness. We use their state-of-the-art streaming algorithms for the fastest paths and temporal closeness in the baseline in our experiments. In Sect. 4.5, we further discuss the differences to our algorithms.

For temporal graphs with unit traversal times, Crescenzi et al. [11] introduce a temporal closeness variant based on the durations of earliest arrival paths by integrating over all starting times in a given time interval. For the exact closeness computation, they use a variant of the edge streaming algorithm. For finding the top- k closeness vertices, the authors define a backward version of their algorithm and provide a variant of Eppstein and Wangs [13] algorithm

for closeness approximation. After approximating the closeness of all vertices, the vertices are ranked according to the approximated closeness. Finally, the exact closeness is computed for the $K > k$ highest-ranked vertices in order to obtain a final top- k ranking. In [50], the authors introduce *shortest-fastest* paths as a combination of the conventional distance and shortest duration. They extend Brandes' algorithm [6] for computing betweenness centrality in temporal graphs. There are works on temporal walk-based centrality measures, e.g., the authors of [2] adapt the walk-based Katz centrality to temporal graphs, and Rozenstein and Gionis [44] give a temporal PageRank variant. Nicosia et al. [37], and Kim and Anderson [25] examine a wide range of properties of temporal graphs, including various temporal centrality measures. In [48] and [49], the authors compare temporal distance metrics and temporal centrality measures to their static counterparts. They reveal that the temporal versions for analyzing temporal graphs have advantages over static approaches on the aggregated graphs.

3 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* (or *traversal time*) $\lambda \in \mathbb{N}$. The transition time of an edge denotes the time required to traverse the edge. We only consider directed temporal graphs—it is possible to model undirectedness using a forward- and a backward-directed edge with equal time stamps and traversal times for each undirected edge. Notice that in 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 traversal times, 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 $\mathbf{G}$.

Figure 2a shows an example of a temporal graph $\mathbf{G}$ with $\mathcal{T}(\mathbf{G}) = \{1, 2, 5, 6, 7, 8\}$. At each edge the availability and transition time is shown. Notice that $\mathbf{G} = \mathbf{G}^I$ for $I = [1, 12]$, $\hat{\tau}_+^I = \max\{1, 2, 3\} = 3$ and $\hat{\tau}_-^I = \max\{0, 1, 2\} = 2$. Restricting the temporal graph to the interval $J = [5, 9]$ leads to the temporal subgraph shown in Figure 2b. For $\mathbf{G}^J$ it is $\mathcal{T}(\mathbf{G}^J) = \{5, 6, 7\}$, $\hat{\tau}_+^J = 1$ and $\hat{\tau}_-^J = 1$.

3.1 Temporal paths

A *temporal path* P between vertices $v_1, v_{\ell+1} \in V$ of length ℓ is an alternating sequence of vertices and temporal edges

$$(v_1, e_1 = (v_1, v_2, t_1, \lambda_1), v_2, \dots, e_\ell = (v_\ell, v_{\ell+1}, t_\ell, \lambda_\ell), v_{\ell+1})$$

such that $e_i \in \mathbf{E}$, 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 or vertices. 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

(a) Example for a temporal graph G . At each edge the availability and transition time is given as pair (t, λ) .

(b) The temporal subgraph $G^{[5,9]}$ of G that is restricted to the time interval $[5, 9]$.

Fig. 2 Example of a temporal graph and a time-restricted subgraph

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 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 G^I . We define $r^I(u) = |R^I(u)|$.

For example, in Fig. 2a there are three paths between vertices a and d . The first one consists of only the edge $P_1 = ((a, d, 1, 5))$ and has a duration of $d(P_1) = 6 - 1 = 5$. The second path is $P_2 = ((a, b, 2, 1), (b, d, 7, 2))$ with duration $d(P_2) = 9 - 2 = 7$. And lastly, path three $P_3 = ((a, b, 5, 2), (b, d, 7, 2))$ with duration $d(P_3) = 9 - 5 = 4$. Therefore, P_3 is the only fastest (a, d) -path. Notice that the subpath $P'_3 = ((a, b, 5, 2))$ is not a fastest (a, b) -path. The only fastest (a, b) -path consists of only edge $(a, b, 2, 1)$ and has a duration of one. Furthermore, notice the restricted reachability due to the temporal constraints. For example vertex b is able to reach vertex d , and vertex d can reach vertex c . However, the path from b to d has an arrival time of nine and is too late to be continued with one of the outgoing edges at d , which start at six and eight, respectively.

3.2 Harmonic temporal closeness

For a temporal graph $G = (V, E)$ and a time interval I , let $\mathcal{P}_{uv}^I$ be the set of all temporal paths between $u, v \in V$ in 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 [31] introduced harmonic closeness in static graphs.

Definition 1 (Harmonic Temporal Closeness) Let $G = (V, E)$ 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 $c_n^I(u) = \frac{c^I(u)}{|V|}$ 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.

Notice that the temporal closeness can be defined using either the outgoing or incoming fastest paths, i.e., using $d^I(u, v)$ or $d^I(v, u)$. In the first case, the temporal closeness describes how well vertex u can reach the other vertices in terms of duration. The second case describes how well the other vertices can reach vertex u . Therefore, analogous to the harmonic temporal closeness, we define the harmonic temporal in-closeness.

Definition 3 (Harmonic Temporal in-Closeness) Let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph and I a time interval. We define the harmonic temporal in-closeness for $u \in V$ with respect to I as $c_{in}^I(u) = \sum_{v \in V \setminus \{u\}} \frac{1}{d^I(v, u)}$.

In Sect. 5, we introduce the *temporal transpose* of a temporal graph that enables our temporal closeness algorithms to calculate the harmonic temporal in-closeness in case that all edges have equal transition times.

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

4 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.

4.1 A label setting fastest path algorithm

Recall that, in general, a sub-path of a fastest path is not necessarily a fastest path. We deal with this problem by using a label setting algorithm that finds the fastest paths from a start vertex u to all other vertices 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 [52].

Definition 4 (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 (line 5). At this point, if the algorithm discovers v for the first time, the shortest duration $d^I(u, v)$ is found.

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: $l = (v, s, a) \leftarrow \mathcal{Q}.\text{extractMin}$
- 6: **if** $v \notin F$ **then**
- 7: $d[v] = a - s$
- 8: $F = F \cup v$
- 9: **for** each outgoing edge $e = (v, w, t, \lambda)$ from v **do**
- 10: **if** $l.a \leq e.t$ and $\alpha \leq e.t \leq \beta - e.\lambda$ **then**
- 11: **if** $l.s = 0$ **then** $l' = (w, t, t + \lambda)$
- 12: **else** $l' = (w, l.s, t + \lambda)$
- 13: remove dominated labels from $\Pi[w]$ and $\mathcal{Q}$
- 14: **if** l' is not dominated **then**
- 15: $\mathcal{Q}.\text{insert}(l')$ and $\Pi[w].\text{add}(l')$
- 16: **return** d

Lemma 1 Let $(l_1 = (w_1, s_1, a_1), \dots, l_p = (w_p, s_p, a_p))$ be the sequence of labels returned by the `extractMin` call of the priority queue. Then the durations are non-decreasing, i.e., $a_i - s_i \leq a_j - s_j$ for $1 \leq i < j \leq p$.

Proof The duration is strictly increasing in the length of a path because we have strictly positive transition times, i.e., using an edge takes at least one time step. We use induction over the iteration i of the while loop (line 4). The first initial label has duration $a - s = 0$. Now assume the hypothesis holds after the i -th iteration, for an $i \geq 1$. In the $(i + 1)$ -th iteration, first the label l_{i+1} with the currently shortest duration is returned from the priority queue. The inner for loop (line 9) iterates over all outgoing-edges at w and a new label might be added to the priority queue for each possible extension of the (u, w) -path. The duration of all newly inserted labels is larger than the one of l_{i+1} and therefore also larger than the duration of any label that was already in the queue when l_{i+1} was returned. However, labels newly inserted during one iteration of the while loop may have equal duration. $\square$

Lemma 2 If during the iterations of the while loop a vertex v is inserted in F (line 6 ff.), the duration $d[v] = a - s$ is equal to the duration of a fastest (u, v) -path.

Proof Vertex v is added to F when label $l = (v, s, a)$ is returned from the priority queue and if $v \notin F$. After a vertex has been added to F , it remains in F . Therefore, l is the first label that is processed for vertex v . Due to Lemma 1, it follows that the durations of the processed labels are non-decreasing. Consequently, $a - s$ is the shortest duration of a fastest path from u to v . $\square$

The following theorem states the 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 δ_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 $\mathcal{O}(|V| + |\mathbf{E}| \cdot |\tau_+^I(u)| \cdot \xi)$.

Proof Let $w \in R^I(u)$ be a temporally reachable vertex. There has to be at least one, or the temporal closeness of u is 0. Because w is reachable, we know that there exists a fastest temporal (u, w) -path $P = (v_1 = u, e_1, \dots, e_{\ell-1}, v_{\ell} = w)$. By induction over the length ℓ , we show that for each prefix-path a corresponding label is generated. For the prefix-path of length $h = 0$, we have the initial label at vertex u . We assume that for some $h \geq 0$ the hypothesis holds. Now, for the case $h + 1$, we have the prefix-path $P_h = (v_1 = u, e_1, \dots, e_h, v_{h+1})$ which consists of $(v_1 = u, e_1, \dots, v_h)$ and edge e_h arriving at vertex v_{h+1} . By applying the induction hypothesis, a label (v_h, s', a') for P_h was created and added to $\mathcal{Q}$.

The only way the label could get deleted without being processed is by being dominated by a label (v_h, s'', a'') . Without loss of generality, we can assume that not both $s' = s''$ and $a' \geq a''$ holds (in this case we consider the path represented by (v_h, s'', a'')). Assume that $s' < s''$ and $a' \geq a''$ then P would not be a fastest temporal path. Therefore, the label does not get deleted from $\mathcal{Q}$ due to dominance checks.

However, eventually, it gets processed in some iteration, and a label for P_{h+1} is added to the priority queue. The label representing P_{h+1} cannot be dominated for similar reasons as for the label for P_h . Hence, at some point, vertex w is reached for the first time and added to the set F . With Lemma 2, it follows that a fastest temporal path has been found, and $d[w]$ is set to the shortest duration.

The total number of labels is upper bounded by $|\mathbf{E}^I| \cdot |\tau_+^I(u)|$ and $\mathcal{Q}$ will eventually be empty and the algorithm terminates.

For the running time, we have the following considerations. Initialization is done in $\mathcal{O}(|V|)$. The while loop (line 4) is called for every label (v, s, a) , and the inner for loop (line 9) is called for each out-going edge at vertex v in $\mathbf{G}$. During the inner for loop, a new label is created and compared to all labels at vertex w for the dominance check. Due to the dominance check, for each possible arrival time, we may only keep the label with the latest starting time and no equal labels. And, for two labels with equal starting times, we only keep the one with the earlier arrival time. Therefore, the number of labels at a vertex w is less or equal to the maximum of the number of different arrival times at any $v \in V$ and different starting times at u , i.e., the size of $\Pi[w]$ is at most π . The domination checks for all outgoing edges (line 9 ff.) can be done in $\pi\delta_m^+$. The total number of labels generated is less or equal to $|\mathbf{E}^I| \cdot |\tau_+^I(u)|$. However, at any time there are at most $|\mathbf{E}^I|$ labels in the priority queue, because we only keep the label with the latest starting time per edge. Therefore, the cost for extracting and deleting a label from the priority queue is amortized $\mathcal{O}(\log(|\mathbf{E}^I|))$ using a Fibonacci heap, and inserting is possible in constant time. Because we have to check for each edge in $e \in \mathbf{E}$ if e is in the time interval I , the total running time is in $\mathcal{O}(|V| + |\mathbf{E}| + |\mathbf{E}^I| \cdot |\tau_+^I(u)| \cdot \xi) = \mathcal{O}(|V| + |\mathbf{E}| \cdot |\tau_+^I(u)| \cdot \xi)$. $\square$

4.2 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. [3] 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 an 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

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

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 number of reachable vertices are discussed in Sect. 4.4. 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. After ordering the vertices, 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 $d^I(v_j, v)$.

Let $(l_1 = (w_1, s_1, a_1), \dots, l_p = (w_p, s_p, a_p))$ 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. It holds that $d_{i+1} = a_{i+1} - s_{i+1}$ is the next possible duration to any reachable vertex that is not yet in $F_i(v_j)$. Now, let Δ 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 3 *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 $d(v_j, r) \geq d_{i+1} + \Delta + \lambda_{\min}$.*

Proof Let $w \in T_i(v_j)$. Then its duration is not found yet. The next label l_{i+1} provides the next possible duration any reachable vertex can have with $d_{i+1} = a_{i+1} - s_{i+1}$. Moreover, for any reachable vertex z that is not in $F_i(v_j) \cup T_i(v_j)$, we have to account for at least the additional waiting time at a vertex in $y \in T_i(v_j)$ plus the transition time from y to z . $\square$

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

```

1:  $\mathbf{E}^I \leftarrow \{(u, v, t, \lambda) \in \mathbf{E} \mid t \geq \alpha \text{ and } t + \lambda \leq \beta\}$ 
2: Determine  $\lambda_{min}$  and  $\Delta$ 
3: Let  $v_1, \dots, v_n \in V$  in order of decreasing out-degree
4: for  $j = 1 \dots n$  do
5:   Initialize label  $l = (v_j, 0, 0)$  at vertex  $v_j$ 
6:   Initialize PQ  $\mathcal{Q}$  and insert  $l$ 
7:   Initialize  $d[v] = \infty$  for all  $v \in V$ 
8:   Initialize  $F_0(v_j) = \emptyset, T_0(v_j) = \{v_j\}$ 
9:   Initialize empty lists  $\Pi[v]$  for all  $v \in V$ 
10:   $d[v_j] = 0, i = 1$ 
11:  while  $\mathcal{Q}$  not empty and  $|F_{i-1}(v_j)| < |V|$  do
12:     $l = (v, s, a) \leftarrow \mathcal{Q}.extractMin$ 
13:    if  $v \in T_{i-1}(v_j)$  and  $v \notin F_{i-1}(v_j)$  then
14:       $d[v] = a - s$ 
15:       $T_i(v_j) = T_{i-1}(v_j) \setminus \{v\}$ 
16:       $F_i(v_j) = F_{i-1}(v_j) \cup \{v\}$ 
17:    else  $T_i(v_j) = T_{i-1}(v_j)$ 
18:    for each  $e = (v, w, t, \lambda) \in \mathbf{E}^I$  do
19:      if  $l.a \leq e.t$  then
20:        if  $l.s = 0$  then  $l' = (w, t, t + \lambda)$ 
21:        else  $l' = (w, l.s, t + \lambda)$ 
22:        remove dominated labels from  $\Pi[w]$  and  $\mathcal{Q}$ 
23:        if  $l'$  is not dominated then
24:           $\mathcal{Q}.insert(l')$  and  $\Pi[w].add(l')$ 
25:        if  $w \notin F_i(v_j)$  then
26:           $T_i(v_j) = T_i(v_j) \cup \{w\}$ 
27:      update  $\tilde{c}^I(v_j)$ 
28:      if  $\tilde{c}^I(v_j) < \text{smallest value in top } k \text{ values}$  then
29:        stop computation for vertex  $v_j$ 
30:       $i \leftarrow i + 1$ 
31:    update top- $k$  results with  $v_j$  and  $c^I(v_j) = \tilde{c}^I(v_j)$ 
32: return top- $k$  values and corresponding vertices

```

Altogether, we have the following upper bound.

Theorem 2 In Algorithm 2, the temporal closeness $c^I(u)$ in iteration i of the while loop is less or equal to

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

Finally, we discuss the running time of the algorithm.

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

Proof Ordering the vertices by decreasing out-degree takes $\mathcal{O}(|V| \log |V|)$ time. Computing λ_{min} is done during the removal of vertices that are not in the time interval I , i.e., this together takes $\mathcal{O}(|\mathbf{E}|)$ time. We can compute Δ by checking for each edge $(u, v, t, \lambda) \in \mathbf{E}^I$ the time stamps of the outgoing edges at v in $\mathcal{O}(|\mathbf{E}^I| \delta_m^+)$. Initialization is done in $\mathcal{O}(|\mathbf{E}|)$. The outer for

loop (line 4) is called for each vertex, and the inner loop is an adaption of Algorithm 1. The updating of the upper bound can be done in constant time. Therefore, we have $|V|$ times the running time of Algorithm 1. Keeping track of the top- k results is possible in $\mathcal{O}(|V| \log k)$. $\square$

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 $\bar{c}(v_i)$.

4.3 Heuristic modifications

We propose two different modifications for our fastest path algorithm and obtain heuristic versions of our temporal closeness algorithms.

Heuristic 1: We propose the following heuristic approach. The idea is to limit the size of the label lists at the vertices. Let $h \in \mathbb{N}$ be the maximal size of each label list at $v \in V$. At each vertex $v \in V$, we then have maximal h labels at any point during the computation of the temporal closeness, i.e., $|\Pi[v]| \leq h$. We achieve this by adding a check before adding a new label to the list in line 24. If the size of $\Pi[v]$ is already h at this point, we discard the new label.

Theorem 4 Let δ_m^+ be the maximal out-degree and $\xi = \max\{h\delta_m^+, \log(|\mathbf{E}^I|)\}$. The running time of the temporal top- k closeness algorithm using Heuristic 1 is in $\mathcal{O}(|V|^2 + |V||\mathbf{E}| \cdot \hat{\tau}_+^I \xi)$.

Proof Let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph, $u \in V$, and I a time interval. The fastest path algorithm using the first heuristic returns estimations of the durations of the fastest (u, v) -paths for all $v \in V$ in $\mathbf{G}^I$ in running time $\mathcal{O}(|V| + |\mathbf{E}| \cdot |\tau_+^I(u)| \cdot \xi)$. The difference for the running time happens during the inner for loop when a label is created and compared to all labels at vertex w for the dominance check. The number of labels at a vertex w is now less or equal to the size of $\Pi[w]$, i.e., h , and the domination checks for all outgoing edges (line 9 ff.) can be done in $h\delta_m^+$. Because we call the adapted fastest paths algorithm for each $v \in V$ the result follows. $\square$

Heuristic 2: We can further reduce the running time in the case of $h = 1$ by ignoring vertices after they are visited for the first time. We only need to store a single label at each vertex and do not need the label lists at the vertices. Even though this further reduces the possibilities to find the shortest duration path, our experimental evaluation shows that this approach performs well on real-world data sets. 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, λ) with $a \leq t$. All further labels (v, s', a') that are returned from the priority queue in the following iterations are ignored, even if $a' < a$. This means that we only have to store the label with the minimum duration found so far at each vertex $v \in V$ because this is the label that will determine the final duration $d(u, v)$. Consequently, this approach leads to a Dijkstra-like algorithm. As we argued already in Sect. 3.1, we cannot guarantee to find the fastest paths because, in general, fastest paths do not consist of fastest sub-paths.

Theorem 5 The running time of temporal top- k closeness algorithm using the second heuristic is in $\mathcal{O}(|V|^2 + |V||\mathbf{E}| \cdot \log(|\mathbf{E}^I|))$.

Proof The fastest path algorithm using the second heuristic has a running time in $\mathcal{O}(|V| + |\mathbf{E}| \cdot \log(|\mathbf{E}|^I))$. The dominance checks are unnecessary, and the total number of labels generated is less or equal to $|\mathbf{E}|^I$. Because we do not visit any vertex more than once, for each edge at most one label is added to the priority queue. We call the adapted fastest paths algorithm for each $v \in V$, and therefore, the result follows. $\square$

4.4 The number of reachable vertices

The top- k temporal closeness algorithms need as one of its inputs the number $r^I(u)$ of reachable vertices in $\mathbf{G}^I$ 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. [19]. If the adjacent edges are 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 [52] with $\mathcal{O}(|V|^2 + |V||\mathbf{E}|)$ running time. In Sect. 5.1, we present an approximation for speeding up the computation of the number of reachable vertices.

4.5 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. [52] 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 δ_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.

5 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. [8]. The second one approximates the temporal closeness for all vertices and adapts the undirected static closeness approximation from Wang et al. [13]. 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.

Fig. 3 Example for a temporal graph $\mathbf{G}$ and its temporal transpose $\mathcal{I}(\mathbf{G})$. All edges of $\mathbf{G}$ and $\mathcal{I}(\mathbf{G})$ have unit transition times and $t_{\max} = 9$ due to edge $(w, z, 8, 1)$

First, we introduce the concept of the temporal transpose of a temporal graph, which corresponds to reversing the edges in a conventional static graph and allows the search of incoming temporal paths.

Definition 5 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 $\mathbf{G}$.

Figure 3 shows an example for a temporal graph $\mathbf{G}$ and its temporal transpose $\mathcal{I}(\mathbf{G})$. The following lemma is the basis for the approximations.

Lemma 4 Let $\mathbf{G} = (V, \mathbf{E})$ be a temporal graph in which all edges have equal transition times, $\mathcal{I}(\mathbf{G})$ its temporal transpose, and $u, v \in V$. 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.

Proof We use $t(e)$ to denote the availability time of edge $e \in \mathbf{E}$. First notice that every (non-temporal) (u, v) -path P in $\mathbf{G}$ corresponds to a (non-temporal) (v, u) -path in $\mathcal{I}(\mathbf{G})$ that visits the vertices in reverse order of P . Therefore, having a temporal path in $\mathbf{G}$ (or $\mathcal{I}(\mathbf{G})$) there exists a sequence of vertices and edges in $\mathcal{I}(\mathbf{G})$ (or $\mathbf{G}$, respectively) for which we have to show that the time stamps at the edges allow for a temporal path. Let $P = (v_1, e_1, v_2, \dots, e_\ell, v_{\ell+1})$ be a temporal path in $\mathbf{G}$. Now, consider the sequence $Q = (v_{\ell+1}, e'_\ell, v_\ell, \dots, e'_1, v_1)$ in $\mathcal{I}(\mathbf{G})$, with $e'_i = (v_{i+1}, v_i, t_{\max} - t, \lambda)$ for each $e_i = (v_i, v_{i+1}, t, \lambda)$ and $i \in \{1, \dots, \ell\}$. From $t(e_i) + \lambda \leq t(e_{i+1})$ follows $t_{\max} - t(e_i) \geq t_{\max} - t(e_{i+1}) + \lambda$. Therefore, the sequence Q is a temporal path in $\mathcal{I}(\mathbf{G})$. The other direction is symmetric. Furthermore, we have

$$\begin{aligned}
 d(P) &= a(P) - s(P) \\
 &= \lambda + t(e_\ell) - t(e_1) \\
 &= \lambda + t(e_\ell) - t(e_1) + t_{\max} - t_{\max} \\
 &= \lambda + (t_{\max} - t(e_1)) - (t_{\max} - t(e_\ell)) \\
 &= \lambda + t(e'_1) - t(e'_\ell) \\
 &= a(Q) - s(Q) = d(Q).
 \end{aligned}$$

□

Lemma 4 allows us to apply our temporal closeness algorithms to the transposed temporal graph in order to consider the shortest durations of incoming instead of outgoing temporal paths. This way, we can compute the temporal in-closeness.

Theorem 6 Let $\mathbf{G}$ be a temporal graph in which all edges have equal transition times, $\mathcal{I}(\mathbf{G})$ its temporal transpose. The temporal closeness $c(u)$ of a vertex $u \in V$ in $\mathcal{I}(\mathbf{G})$ is equal to the temporal in-closeness $c_{in}(u)$ in $\mathbf{G}$.

Furthermore, Lemma 4 shows that the set of vertices $R(u)$ reachable by u in the transposed graph $\mathcal{I}(\mathbf{G})$ is equal to the set of vertices in $\mathbf{G}$ that can reach u .

5.1 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. [8]. The non-weighted version proceeds the following way. For two sets X and Y let $S : Y \rightarrow 2^X$. Furthermore, let L be an oracle that if it is presented a random permutation $r : X \rightarrow \{1, \dots, |X|\}$ it returns a mapping $l : Y \rightarrow X$ 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 $l_i : Y \rightarrow X$. For an estimator $\hat{s}(y) = \frac{h}{\sum_{i=1}^h \mathcal{R}_i(l_i(y))} - 1$ the following holds.

Theorem 7 (Cohen et al. [8]) 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 $(u, v, t, \lambda) \in \mathbf{E}$ that start or arrive during 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 $\mathbf{G} = (V, \mathbf{E})$, $h \in \mathbb{N}$, time interval I

Output: Estimates of $\hat{r}^I(u)$ for all $u \in V$

```

1: Calculate  $\mathcal{I}(\mathbf{G}^I)$ 
2: for  $j = 1, \dots, h$  do
3:    $\tilde{\mathbf{G}} \leftarrow \mathcal{I}(\mathbf{G}^I)$ 
4:   assign  $\mathcal{R}(v)$  for all  $v \in V$  uniformly from  $[0, 1]$ 
5:   sort  $v_1, \dots, v_n$  by increasing rank
6:   for  $i = 1, \dots, n$  do
7:     compute  $R(v_i)$  and remove used edges in  $\tilde{\mathbf{G}}$ 
8:     set  $l_j(v) = \mathcal{R}(v_i)$  for all non-marked  $v \in R(v_i)$ 
9:     mark all  $v \in R(v_i)$ 
10: for  $v \in V$  do
11:    $\hat{r}^I(v) = \frac{h}{\sum_{j=0}^h l_j(v)} - 1$ 

```

Theorem 8 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

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

Proof After assigning $l_j(v) = \mathcal{R}(v_i)$, we do not need to reach v again using the edges that we took on the path from v_i to v . All possible extensions outgoing from v will be explored. Then, the result follows from Lemma 4 and Theorem 7. $\square$

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 $\mathcal{O}(|V| \log |V|)$, and computing $R(v)$ takes $\mathcal{O}(|\mathbf{E}^I| + |V|)$ for each $v \in V$, however, each edge is at most processed once in each of the h rounds.

Theorem 9 *The running time of Algorithm 3 is in $\mathcal{O}(|\mathbf{E}| + h \cdot (|\mathbf{E}^I| + |V|^2))$.*

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 early for a vertex with a top- k temporal closeness value.

5.2 Approximation of the temporal closeness

We lift the randomized algorithm for undirected, static graphs from Wang et al. [13] 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 4, we can estimate the closeness of a vertex u in $\mathbf{G}^I$ by averaging the distances $d(v, u)$ found in $\mathcal{I}(\mathbf{G}^I)$.

Algorithm 4

Input: Temporal graph $\mathbf{G} = (V, \mathbf{E})$, $h \in \mathbb{N}$, time interval I

Output: Estimates $\hat{c}_n^I(u)$ for $u \in V$

- 1: Calculate $\mathcal{I}(\mathbf{G}^I)$
 - 2: Sample $v_1, \dots, v_h$ from V
 - 3: **for** $i = 1 \dots h$ **do**
 - 4: Compute $d(v_i, w)$ for all $w \in V$ in $\mathcal{I}(\mathbf{G}^I)$
 - 5: **for** $u \in V$ **do**
 - 6: Let $\hat{c}_n^I(u) = \frac{1}{h} \sum_{i=1}^h \frac{1}{d(v_i, u)}$
-

Theorem 10 *Let $\mathbf{G} = (V, \mathbf{E})$ 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*

$h = \log |V| \cdot \epsilon^{-2}$ the probability

$$P(|\hat{c}_n^I(u) - c_n^I(u)| \geq \epsilon) \leq \frac{2}{|V|^2}.$$

The running time is in $\mathcal{O}(|\mathbf{E}| + h \cdot (|V| + |\mathbf{E}^I|))$.

Proof Due to Lemma 4, we know that there is a one-to-one mapping between the temporal paths in $\mathbf{G}^I$ and $\mathcal{I}(\mathbf{G}^I)$ in which corresponding paths have the same duration. Next, we use Hoeffdings inequality [16]. Let $x_1, \dots, x_h$ be independent random variables that are bounded by $0 \leq x_i \leq 1$ for $i \in \{1, \dots, h\}$. Furthermore, let $S = \sum_{i=1}^h x_i$ and $\mu = \mathbb{E}[S/h]$ the expected mean. Then the following inequality holds.

$$P(|S/h - \mu| \geq \epsilon) \leq 2e^{-2h\epsilon^2}.$$

Now, for $x_i = \frac{1}{d(v_i, u)}$ and $\mu = E\left[\sum_{i=1}^h \frac{1}{d(v_i, u)} \cdot \frac{1}{h}\right]$ we have

$$P\left(\left|1/h \cdot \sum_{i=1}^h \frac{1}{d(v_i, u)} - \mu\right| \geq \epsilon\right) \leq 2e^{-2h\epsilon^2}.$$

Then with $h = \log |V| \cdot \epsilon^{-2}$ it follows

$$P(|\hat{c}_n^I(u) - c_n^I(u)| \geq \epsilon) \leq 2e^{-2(\log |V| \cdot \epsilon^{-2})\epsilon^2} = \frac{2}{|V|^2}.$$

□

6 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 does using the heuristics for computing the fastest paths affect 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 temporal closeness compare to static closeness, degree centrality, and reachability? How do the temporal closeness and the temporal in-closeness compare to each other?

6.1 Data sets

We used the following real-world temporal graph data sets:

- *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* [20].
- *Arxiv* An authors collaboration network from the *arXiv's High Energy Physics - Phenomenology (hep-ph)* section [30]. Vertices represent authors and edges collaborations. The time stamp of an edge is the publication date.
- *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 [51].
- *Prosper* A network based on a personal loan website. Vertices represent persons, and each edge a loan from one person to another person [42].
- *WikipediaSG* The network is based on the *Wikipedia* network. Each vertex represents a *Wikipedia* page, and each edge a hyperlink between two pages [35].
- *WikiTalk* Vertices represent users and edges edits of a user's talk page by another user [29,47].
- *Digg* and *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 [17,34].

¹ <http://www.sociopatterns.org>

Table 1 Statistics and properties of the data sets, with δ_m^- (resp. δ_m^+) being the maximal in-degree (resp. out-degree), and $|E_s|$ the number of edges in the aggregated graph

Data set	Property		δ_m^+	δ_m^-	$\hat{\tau}_+$	$\hat{\tau}_-$	$ \mathcal{T}(\mathbf{G}) $	$ E_s $
	$ V $	$ E $						
Infectious	10 972	415 912	457	544	310	307	76 943	52 761
Arxiv	28 093	4 596 803	9 301	3 962	321	284	2 337	3 148 447
Facebook	63 731	817 035	998	219	412	86	333 924	817 035
Prosper	89 269	3 394 978	9 436	1 071	412	8	1 259	3 330 224
WikipediaSG	208 142	810 702	1 574	4 939	153	223	223	810 702
WikiTalk	225 749	1 554 698	113 867	36 413	75 045	36 410	1 389 632	565 476
Digg	279 630	1 731 653	995	12 038	982	11 506	6 864	1 731 653
FlickrSG	323 181	1 577 469	3 153	2 128	20	20	20	1 577 469

For *WikipediaSG* and *FlickrSG*, we chose a random time interval in size of ten percent of the total time span. For the other data sets, the time interval spans over all edges. Table 1 gives an overview of the statistics of the networks (for *WikipediaSG* and *FlickrSG* the statistics are for the subgraph $\mathbf{G}^I$). The transition times are one for all edges in all data sets.

6.2 Experimental protocol and algorithms

All experiments were conducted on a workstation with an AMD EPYC 7402P 24-Core Processor with 3.35 GHz and 256 GB of RAM running Ubuntu 18.04.3 LTS. We used GNU CC Compiler 9.3.0 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 [52].

For TC- TOP- k and TC- ALL, we also implemented the variants using the heuristics HEURISTIC1 and HEURISTIC2 described in Sect. 4.3. Our source code and the data sets are available at <https://gitlab.com/tgpublic/tgcloseness>.

6.3 Results and discussion

Q1 Table 2 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. The reported running times include the time spend for computing the reachability. 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 4 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

Table 2 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	1 076.83	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	1 622.04
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 3 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 $\pm$ 0.01	1.45 $\pm$ 0.01	1.88 $\pm$ 0.01
Arxiv	117.25 $\pm$ 0.34	310.36 $\pm$ 0.53	1 075.13 $\pm$ 5.10
Facebook	33.81 $\pm$ 2.83	41.14 $\pm$ 4.68	77.32 $\pm$ 5.39
Prosper	224.33 $\pm$ 3.09	227.56 $\pm$ 0.48	246.65 $\pm$ 1.49
WikipediaSG	346.95 $\pm$ 25.20	340.39 $\pm$ 2.31	369.37 $\pm$ 2.43
WikiTalk	1 010.12 $\pm$ 5.86	1 472.38 $\pm$ 10.63	2 678.99 $\pm$ 4.31
Digg	957.80 $\pm$ 6.19	1 056.20 $\pm$ 5.55	2 353.76 $\pm$ 2.78
FlickrSG	965.75 $\pm$ 2.03	1 014.69 $\pm$ 1.78	1 510.15 $\pm$ 2.73

Table 4 Running times in seconds for the first heuristic with $h = 2$

Data set	Algorithms using Heuristic1		TC-Top-100	TC-Top-1000	TC-All
	TC-Top-1	TC-Top-10			
Infectious	0.62	0.64	0.68	0.89	0.90
Arxiv	112.43	117.53	157.86	381.49	717.44
Facebook	94.35	95.30	99.46	126.84	168.43
Prosper	88.01	88.67	90.34	103.66	178.69
WikipediaSG	282.07	280.49	282.81	298.79	352.15
WikiTalk	891.24	902.72	940.52	1 245.50	2 033.96
Digg	2 023.51	2 027.85	2 054.38	2 376.91	6 548.97
FlickrSG	3 987.14	3 987.44	4 008.83	4 235.94	9 457.16

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

Table 5 Running times in seconds for the second heuristic

Data set	Algorithms using HEURISTIC2		TC-Top-100	TC-Top-1000	TC-All
	TC-Top-1	TC-Top-10			
Infectious	0.53	0.54	0.56	0.66	0.61
Arxiv	115.21	116.97	131.37	218.09	309.34
Facebook	93.27	93.60	95.63	109.96	101.13
Prosper	83.62	83.63	85.25	93.08	130.55
WikipediaSG	266.17	266.06	267.61	277.92	242.35
WikiTalk	858.12	863.55	875.96	989.35	1 149.01
Digg	2 000.59	2 000.63	2 008.89	2 089.17	2 971.84
FlickrSG	3 934.20	3 935.52	3 942.18	4 011.21	4 444.82

Fig. 4 Decrease in running time of the top- k algorithms in percent compared to baseline EDGESTR

times faster, and for the PROSPER 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 3 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 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 9), 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 all data sets and all $k \in \{10, 100, 1000\}$, we obtained top- k vertex sets with similarities of one, i.e., the algorithm found the exact top- k sets in all ten rounds.

Fig. 5 Decrease of the running times in percent of the algorithms using the heuristics compared to the exact algorithms

Q3 We ran the temporal closeness algorithms using HEURISTIC1 with $h = 2$. The running times are shown in Table 4. They are lower than those of our exact algorithms for all data sets. Figure 5a shows the decrease in running time compared to the exact algorithms. For *Infectious*, the running time is decreased by around 50% for all algorithms. In case of *Arxiv* the heuristic decreases the running time between 61% (TC- ALL) and 71% (TC- TOP- 100). Similar results are achieved for the *Prosper* data set with decreases between 61% (TC- ALL) and 66% (TC- TOP- 1, TC- TOP- 10 and TC- TOP- 100). For *WikipediaSG* and *WikiTalk*, the decrease in running time is between 25% and 57%, and for *Digg* between 13% and 35%. The lowest decrease in running time is achieved for *Facebook* and *FlickrSG* with values between 11% – 21% and 5% – 33%, respectively. We measured the average relative deviation from the exact closeness values and the Jaccard similarity between the computed top- k sets and the exact solutions. For *Arxiv*, *Prosper*, *Digg* and *FlickrSG* the exact top- k sets are computed for all $k \in \{1, 10, 100, 1000\}$, i.e., the Jaccard similarity is one. Furthermore, for *Arxiv* and *Prosper* the average relative deviation from the exact closeness is less than $2 \cdot 10^{-6}$. For *Digg* and *FlickrSG*, the deviations are less than 0.0003 and $7 \cdot 10^{-5}$. The Jaccard similarities for all remaining data sets are one in case of $k = 1$ and $k = 10$. For *Facebook*, *WikipediaSG* and *WikiTalk*, the Jaccard similarity is one for $k = 100$ and 0.99 for $k = 1000$. The similarities for *Infectious* are 0.96 for both $k = 100$ and $k = 1000$. The average relative deviation from the exact closeness is less than $3 \cdot 10^{-5}$ for *Facebook*, *WikipediaSG* and *WikiTalk*. For the *Infectious* data set, the deviation is less than 0.007.

Next, we ran the temporal closeness algorithms using the second heuristic (HEURISTIC2). Table 5 shows the running times. As expected, the running times are lower than those of our exact algorithms for all data sets and lower than the running times of the algorithms using the first heuristic in almost all cases. Figure Fig. 5b shows the decrease in running time compared to the exact algorithms. For *Infectious*, *Arxiv* and *Prosper* the second heuristic leads to large gains in performance for all algorithms of at least 55%, 66%, and 68%, respectively. In case of *WikipediaSG* and *WikiTalk*, the decrease is at least 27% and 32%, respectively. Notice that for TC- ALL the decrease is at least 52% (*Facebook*) and up to 83% (*Arxiv*). All data sets but *Infectious* have a Jaccard similarity of one for $k \in \{1, 10, 100\}$. *Arxiv*, *Facebook* and *Prosper* have a Jaccard similarity of one for $k = 1000$, and their average relative deviation from the exact closeness is less than $2 \cdot 10^{-6}$. The *Infectious* data set has a Jaccard similarity of 0.96 for both $k = 100$ and $k = 1000$. The average relative deviation is 0.008. The remaining data

sets have a Jaccard similarity of 0.99 for $k = 1000$ and an average relative deviation of 0.003 (*FlickrSG*), $5 \cdot 10^{-5}$ (*WikiTalk*), $3 \cdot 10^{-5}$ (*Digg*), and $1 \cdot 10^{-6}$ (*WikipediaSG*). The heuristics provide very good results with small errors in the computed closeness values. For small k , the correct top- k sets are found. For larger k , the Jaccard similarity is still very high.

Q4 For a fixed error ϵ , the sample size h grows logarithmically in $|V|$, and the approximation is most suitable for data sets with a high number of vertices. For example, if we want to achieve an error of $\epsilon \leq 0.001$ (with high probability) for a data set with $|V| = 10^6$ vertices it follows $h = \log 10^6 \cdot 0.001^{-2} \approx 1.38 \cdot 10^7$. In this case, the needed sample size exceeds the number of vertices by a factor greater than ten, and the exact computation of the normalized temporal closeness will be faster. However, for a data set with $|V| = 10^8$ vertices, we have $h \approx 1.84 \cdot 10^7$, and more drastically for $|V| = 10^{100}$ it follows $h \approx 2.3 \cdot 10^8$. Choosing a large enough h to achieve a low error for data sets with few vertices may lead to too high running times. Hence, we ran the Algorithm 4 ten times for each data set with the sampling sizes $h = p \cdot |V|$ with $p \in \{0.1, 0.2, 0.5\}$. Table 6 reports the average running times and the standard deviations. Theorem 10 shows that TC- APPROX with high probability computes closeness values with a small additive error. 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 6 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. The standard deviation of the mean approximation error over the ten rounds is below 0.01 for all data sets.

Q5 We measured the Jaccard similarity between the temporal and static top- k closeness vertex sets. Table 7 shows that the similarity is very low in most cases, and for *Facebook* and *Digg* even zero. In case of *Arxiv* and $k = 10$, the similarity is highest with 0.81. Furthermore, Table 7 shows the Jaccard similarity between the temporal top- k closeness vertices and the vertices with the largest number of outgoing temporal edges, i.e., the out-degree centrality. For all chosen k , the Jaccard similarity for *Infectious* is low. For *Facebook*, *WikipediaSG*, *FlickrSG* the sets of the top-10 temporal closeness vertices and degree centrality vertices are equal. However, the similarity declines fast for each of these data sets and is very low for $k = 1000$. We also compared the Jaccard similarity between the top- k temporal closeness vertices and the top- k vertex sets of vertices with the highest reachability values, i.e., vertices u for which $r(u)$ is highest (Table 7). The values are small for all data sets, with 0.23 being the highest similarity for *Arxiv* and $k = 1000$. For $k = 10$, all similarities are zero with exception of the *Facebook* and *FlickrSG* data sets with similarities of 0.17 and 0.05. The Jaccard similarities between the top- k temporal closeness and the temporal in-closeness vertices are very low for all data sets but *WikiTalk* and *FlickrSG*. These low Jaccard similarities are expected due to the missing symmetries of the directed temporal edges and the temporal restrictions. The vertices with a high temporal closeness are different from the vertices that have a high temporal in-closeness.

To further investigate the relationships between the vertex rankings obtained using temporal closeness, degree centrality, and reachability, we measured the correlations of the rankings using the *Kendall rank correlation coefficient* [23]. Because vertices in a graph can have equal centrality values and rank, we use the version accounting for possible ties in the rankings (Kendall's τ_b coefficient) [24]. The Kendall rank correlation coefficient is commonly used for analyzing and comparing vertex rankings given by centrality measures [14]. Let $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$ be two sequences, i.e., the ranked vertices. A pair of (x_i, y_i) and (x_j, y_j) with $1 \leq i < j \leq n$ is *concordant* if $x_i < x_j$ and $y_i < y_j$ or $x_i > x_j$ and $y_i > y_j$. The pair is *discordant* if $x_i > x_j$ and $y_i < y_j$ or $x_i < x_j$ and $y_i > y_j$. The Kendall rank correlation is defined as the number of concordant pairs minus the number

Table 6 The average running times and standard deviations in seconds and the mean approximation error for TC- APPROX with sampling sizes $h = p \cdot |V|$

Data set	TC- APPROX Running time $p = 0.1$	Approximation error		
		$p = 0.2$	$p = 0.5$	$p = 0.1$
Infectious	0.25 ± 0.01	0.47 ± 0.01	1.12 ± 0.01	0.83
Arxiv	410.78 ± 5.16	862.46 ± 10.71	2 067.93 ± 7.52	0.78
Facebook	28.22 ± 0.37	56.61 ± 1.00	131.58 ± 1.20	0.63
Prosper	48.83 ± 0.94	98.00 ± 0.62	228.96 ± 0.48	0.59
WikipediaSG	84.51 ± 0.67	170.38 ± 2.92	353.54 ± 4.23	0.41
WikiTalk	1 573.23 ± 10.06	3 173.40 ± 27.47	7 744.86 ± 35.24	0.08
Digg	1 682.70 ± 27.82	3 356.85 ± 42.60	8 259.91 ± 62.70	0.23
FlickrSG	1 656.13 ± 7.58	3 288.93 ± 27.47	6 946.62 ± 44.30	0.66

Table 7 Jaccard similarity between the top- k temporal closeness vertices and the temporal in-closeness, static closeness vertices in the aggregated graph, vertices with the highest number of outgoing temporal edges, and vertices with the highest reachability

Data set	Static closeness		Degree centrality		Reachability			Temporal in-closeness		
	$k = 10$	$k = 100$	$k = 1000$	$k = 10$	$k = 100$	$k = 1000$	$k = 10$	$k = 100$	$k = 1000$	$k = 1000$
Infectious	0.25	0.18	0.26	0.00	0.11	0.09	0.00	0.00	0.03	0.15
Arxiv	0.81	0.41	0.41	0.25	0.54	0.63	0.00	0.00	0.04	0.10
Facebook	0.00	0.00	0.01	1.00	0.74	0.01	0.17	0.00	0.03	0.09
Prosper	0.00	0.45	0.41	0.81	0.90	0.01	0.00	0.00	0.00	0.00
WikipediaSG	0.00	0.15	0.39	1.00	0.70	0.005	0.00	0.00	0.00	0.07
WikiTalk	0.42	0.45	0.63	0.53	0.57	0.004	0.00	0.17	0.50	0.66
Digg	0.00	0.00	0.00	0.83	0.56	0.004	0.00	0.00	0.00	0.13
FlickrSG	0.42	0.34	0.25	1.00	0.95	0.003	0.05	0.33	0.26	0.33

Fig. 6 The correlation matrices show the Kendall rank correlation between the temporal closeness, degree centrality, reachability centrality, and static closeness vertex rankings

of discordant pairs, normalized by the number of all possible pairs (accounting for ties) in the rankings. The correlation coefficient takes on values between -1 and 1 , where values close to one indicate similar rankings, close to zero no correlation, and close to minus one a strong negative correlation. Figure 6 shows the correlation matrices for all data sets. The correlation between the rankings by degree and temporal closeness is very strong for most data sets. For the *Infectious* and *WikiTalk* data sets, it is 0.53 and 0.78 , and for the other data sets, it is between 0.94 and one. The reason for the high correlation is that, in the case of unit traversal times, each direct neighbor adds a value of one to the temporal closeness. For the reachability, the correlation varies between 0.56 for the *Infectious* and 0.97 for the *WikiTalk* data set, with an average value of 0.82 . There is only little correlation between temporal and static closeness for most data sets.

Finally, we compare different centrality measures as a heuristic for selecting initial vertices for distributing information in a temporal network. The problem of choosing optimal initial vertices is an instance of the influence maximization problem in temporal networks. Kempe et al. [22] have shown that selecting the most influential vertices is NP-hard in static networks. Here, 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. We simulated the dissemination over the time spanned by the temporal graph. At the start of the simulation, only a set of $k = 100$ seed vertices possess the information. We chose the seed vertices with the top- k temporal closeness, the top- k highest degree, top- k static closeness, and k randomly chosen vertices, respectively. The static closeness was computed on the aggregated graph $A(G)$. Next, we evaluated the spread of the information over time. Each vertex, i.e., person, that has the information propagates it with a probability of 10% . Hence, each time a temporal edge $(u, v, t, 1)$ is available at time t , the vertex u gives the information to vertex v with a probability of 10% . The information is then available to vertex v for further distribution at time $t + 1$. We repeated the experiment ten times and measured the mean number of persons who obtained the information after processing the last edge. Figure 7 shows the results. When using the top- k temporal closeness vertices as seeds, the information is distributed to the highest number of people. Using the vertices with the highest

Fig. 7 Comparing different centrality measures for choosing the seed of a dissemination process

degree as seeds leads to the second-best results. The static closeness results cannot compete because the temporal restrictions are not respected during the computation of the top- k static closeness vertices. The randomly chosen vertices perform better than static closeness. In conclusion, temporal closeness can be a helpful heuristic for selecting influential vertices in temporal graphs.

7 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 on its own. Our top- k temporal closeness algorithms improved the running times for all real-world data sets. We introduced simple yet strong heuristic modifications that significantly reduced the running times and led to only small errors for all 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.

Acknowledgements This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813. We thank the anonymous reviewer for valuable feedback on this paper.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory

regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bavelas A (1950) Communication patterns in task-oriented groups. *The J Acoust Soc Am* 22(6):725–730
2. Béres F, Pálovics R, Oláh A, Benczúr András A (2018) Temporal walk based centrality metric for graph streams. *Appl Netw Sci* 3(1):32:1-32:26
3. Bergamini E, Borassi M, Crescenzi P, Marino A, Meyerhenke H (2019) Computing top-k closeness centrality faster in unweighted graphs. *ACM Trans Knowl Discov Data* 13(5):53:1-53:40
4. Bisenius P, Bergamini E, Angriman E, and Meyerhenke H (2018) Computing top-k closeness centrality in fully-dynamic graphs. In: *Proceedings of the twentieth workshop on algorithm engineering and experiments, ALENEX*, pp. 21–35. SIAM
5. Braha D and Bar-Yam Y (2009) Time-dependent complex networks: dynamic centrality, dynamic motifs, and cycles of social interactions. In: *Adaptive Networks*, Springer, pp. 39–50
6. Brandes Ulrik (2001) A faster algorithm for betweenness centrality. *The J Math Sociol* 25(2):163–177
7. Bui-Xuan B-M, Ferreira A, Jarry A (2003) Computing shortest, fastest, and foremost journeys in dynamic networks. *Int J Found Comput Sci* 14(02):267–285
8. Cohen E (1997) Size-estimation framework with applications to transitive closure and reachability. *J Comput Syst Sci* 55(3):441–453
9. Cohen E, Dellling D, Pajor T, and Werneck RF (2014) Computing classic closeness centrality, at scale. In: *Proceedings of the second ACM conference on Online social networks*, pp. 37–50
10. Cooke KL, Halsey E (1966) The shortest route through a network with time-dependent internodal transit times. *J Math Anal Appl* 14(3):493–498
11. Crescenzi P, Magnien Clémence, Marino A (2020) Finding top-k nodes for temporal closeness in large temporal graphs. *Algorithms* 13(9):211
12. Das Kousik, Samanta S, Pal M (2018) Study on centrality measures in social networks: a survey. *Soc Netw Anal Min* 8(1):13
13. Eppstein D, Wang J (2001) Fast approximation of centrality. In: *Proceedings of the twelfth annual symposium on discrete algorithms*, January 7–9, 2001, Washington, DC, USA. ACM/SIAM, pp. 228–229
14. Grando F, Noble D, and Lamb LC (2016) An analysis of centrality measures for complex and social networks. In: *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. IEEE
15. Harary F, Gupta G (1997) Dynamic graph models. *Math Comput Model* 25(7):79–87
16. Hoeffding W (1994) Probability inequalities for sums of bounded random variables. In: *The collected works of Wassily Hoeffding*, pp. 409–426. Springer: New York, NY
17. Hogg T, Lerman Kristina (2012) Social dynamics of digg. *EPI Data Sci* 1(1):5
18. Holme Petter (2015) Modern temporal network theory: a colloquium. *The Eur Phys J B* 88(9):234
19. Huang S, Cheng J, and Wu H (2014) Temporal graph traversals: definitions, algorithms, and applications. *CoRR*, abs/1401.1919
20. Isella L, Stehlé J, Barrat A, Cattuto C, Pinton J-F, Van den Broeck W (2011) What's in a crowd? Analysis of face-to-face behavioral networks. *J Theor Biol* 271(1):166–180
21. Kempe D, Kleinberg JM, Kumar A (2002) Connectivity and inference problems for temporal networks. *J Comput Syst Sci* 64(4):820–842
22. Kempe D, Kleinberg JM, Tardos É (2015) Maximizing the spread of influence through a social network. *Theory Comput* 11:105–147
23. Kendall MG (1938) A new measure of rank correlation. *Biometrika* 30(1/2):81–93
24. Kendall MG (1945) The treatment of ties in ranking problems. *Biometrika* 33(3):239–251
25. Kim H, Anderson R (2012) Temporal node centrality in complex networks. *Phys Rev E* 85(2):026107
26. Kossinets Gueorgi, Watts DJ (2006) Empirical analysis of an evolving social network. *Science* 311(5757):88–90
27. Landherr A, Friedl B, Heidemann J (2010) A critical review of centrality measures in social networks. *Bus Inform Syst Eng* 2(6):371–385
28. 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):61:1-61:29
29. Leskovec J, Huttenlocher D, and Kleinberg J (2010) Governance in social media: a case study of the wikipedia promotion process. In: *Fourth International AAAI conference on weblogs and social media*
30. Leskovec J, Kleinberg J, and Faloutsos C (2007) Graph evolution: densification and shrinking diameters. *ACM Trans Knowl Discov Data*, 1(1):2–es

31. Marchiori M, Latora Vito (2000) Harmony in the small-world. *Physica A Stati Mech Appl* 285(3–4):539–546
32. Masuda N, Holme Petter (2019) Detecting sequences of system states in temporal networks. *Sci Rep* 9(1):1–11
33. Michail Othon (2016) An introduction to temporal graphs: an algorithmic perspective. *Internet Math.* 12(4):239–280
34. Mislove A, Koppula HS, Gummadi KP, Druschel P, and Bhattacharjee B (2008) Growth of the flickr social network. In: *Proceedings of the first workshop on Online social networks*, pp. 25–30
35. Mislove AE (2009) Online social networks: measurement, analysis, and applications to distributed information systems. PhD thesis, Rice University
36. Mutzel P and Oettershagen L (2019) On the enumeration of bicriteria temporal paths. In: *Theory and applications of models of computation, TAMC*, volume 11436 of *lecture notes in computer science*, Springer, pp. 518–535
37. Nicosia V, Tang J, Mascolo C, Musolesi M, Russo G, Latora V (2013) Graph metrics for temporal networks. In: *Temporal Networks*, Springer, pp. 15–40
38. Oettershagen L and Mutzel P (2020) Efficient top-k temporal closeness calculation in temporal networks. In: *2020 IEEE international conference on data mining, ICDM*, pp. 402–411. IEEE
39. Okamoto K, Chen W, and Li X-Y (2008) Ranking of closeness centrality for large-scale social networks. In: *Frontiers in algorithmics, second annual international workshop, FAW*. Springer, pp. 186–195
40. Palla G, Barabási A-L, Vicsek T (2007) Quantifying social group evolution. *Nature* 446(7136):664–667
41. Pan RK, Saramäki J (2011) Path lengths, correlations, and centrality in temporal networks. *Phys Rev E* 84(1):016105
42. Redmond U, Cunningham P (2013) A temporal network analysis reveals the unprofitability of arbitrage in the prosper marketplace. *Exp Syst Appl* 40(9):3715–3721
43. Rodrigues FA (2019) Network centrality: an introduction. In: *A mathematical modeling approach from nonlinear dynamics to complex systems*, Springer, pp. 177–196
44. Rozenshtein P and Gionis A (2016) Temporal pagerank. In: *Machine learning and knowledge discovery in databases - European Conference, ECML PKDD*, volume 9852 of *lecture notes in computer science*, Springer, pp. 674–689
45. Santoro N, Quattrociocchi W, Flocchini P, Casteigts A, Amblard F (2011) Time-varying graphs and social network analysis: temporal indicators and metrics. <https://arxiv.org/abs/1102.0629>
46. Saxena A and Iyengar S (2020) Centrality measures in complex networks: a survey. *CoRR*, abs/2011.07190
47. Sun J, Kunegis J, and Staab S (2016) Predicting user roles in social networks using transfer learning with feature transformation. In: *IEEE international conference on data mining workshops*, pp. 128–135. IEEE Computer Society
48. Tang J, Leontiadis I, Scellato S, Nicosia V, Mascolo C, Musolesi M, and Latora V (2013) Applications of temporal graph metrics to real-world networks. In: *Temporal Networks*. Springer, pp. 135–159
49. Tang J, Musolesi M, Mascolo C, Latora V and Nicosia V (2010) Analysing information flows and key mediators through temporal centrality metrics. In: *Proc 3rd Workshop on Social Network Systems*, pp. 1–6
50. Tsalouchidou I, Baeza-Yates R, Bonchi F, Liao K, and Sellis T (2019) Temporal betweenness centrality in dynamic graphs. *Int J Data Sci Anal*, pp. 1–16
51. Viswanath B, Mislove A, Cha M, and Gummadi KP (2009) On the evolution of user interaction in facebook. In: *Proceedings of the 2nd ACM workshop on Online social networks*, pp. 37–42
52. Huanhuan W, Cheng J, Huang S, Ke Y, Yi L, Yanyan X (2014) Path problems in temporal graphs. *Proc VLDB Endow* 7(9):721–732
53. Yan E, Ding Y (2009) Applying centrality measures to impact analysis: a coauthorship network analysis. *J Am Soc Inform Sci Technol* 60(10):2107–2118
54. Zhan J, Gurung S, Parsa Sai Phani Krishna (2017) Identification of top-k nodes in large networks using Katz centrality. *J Big Data* 4(1):1–19

Lutz Oettershagen received his M.Sc. degree in computer science from TU Dortmund University in Germany. He is currently working toward a Ph.D. degree at the University of Bonn in Germany. His research interests include (temporal) graphs and data mining.

Petra Mutzel is a full professor for Computational Analytics in the Computer Science Department at Bonn University in Germany. She is also the Scientific Director of the High Performance Computing and Analytics Lab of the University. Currently, she is Associate Editor of the ACM Journal on Experimental Algorithmics, the Journal of Graph Algorithms and Applications (JGAA), and Mathematical Programming Computation (MPC). Her research focuses on algorithmic data analysis, algorithm engineering, and combinatorial optimization on graphs and networks. Currently, the main application areas are in Cheminformatics, social and biological network analysis, geodesy, network design, graph drawing, quantum computing, and statistical physics.