Rozenshtein, Polina; Tatti, Nikolaj; Gionis, Aristides

The network-untangling problem

Published in:
Data Mining and Knowledge Discovery

DOI:
10.1007/s10618-020-00717-5

Published: 01/01/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Rozenshtein, P., Tatti, N., & Gionis, A. (2021). The network-untangling problem: from interactions to activity timelines. *Data Mining and Knowledge Discovery*, *35*(1), 213-247. https://doi.org/10.1007/s10618-020-00717-5

Noname manuscript No.
(will be inserted by the editor)

The network-untangling problem: From interactions to activity timelines

Polina Rozenshtein · Nikolaj Tatti · Aristides Gionis

Received: date / Accepted: date

Abstract In this paper we study a problem of determining when entities are active based on their interactions with each other. We consider a set of entities V and a sequence of time-stamped edges E among the entities. Each edge $(u, v, t) \in E$ denotes an interaction between entities u and v at time t. We assume an activity model where each entity is active during at most k time intervals. An interaction (u, v, t) can be *explained* if at least one of u or v are active at time t. Our goal is to reconstruct the *activity intervals* for all entities in the network, so as to explain the observed interactions. This problem, the *network-untangling problem*, can be applied to discover event timelines from complex entity interactions.

We provide two formulations of the network-untangling problem: (*i*) minimizing the total interval length over all entities (SUM version), and (*ii*) minimizing the maximum interval length (MAX version). We study separately the two problems for $k = 1$ and $k > 1$ activity intervals per entity. For the case $k = 1$, we show that the SUM problem is **NP**-hard, while the MAX problem can be solved optimally in linear time. For the SUM problem we provide efficient algorithms motivated by realistic assumptions. For the case of $k > 1$, we show that both formulations are inapproximable. However, we propose efficient algorithms based on alternative optimization. We complement our study with

P. Rozenshtein
Institute of Data Science, National University of Singapore, Singapore, and
Helsinki Institute for Information Technology, Aalto University, Espoo, Finland
E-mail: idspoli@nus.edu.sg

N. Tatti
Helsinki Institute for Information Technology, University of Helsinki, and
F-Secure Corp., Helsinki, Finland

A. Gionis
KTH Royal Institute of Technology, Sweden, and
Helsinki Institute for Information Technology, Aalto University, Espoo, Finland

an evaluation on synthetic and real-world datasets, which demonstrates the validity of our concepts and the good performance of our algorithms.

Keywords Temporal networks · complex networks · timeline reconstruction · vertex cover · linear programming · 2-SAT

1 Introduction

New data abstractions emerging from modern applications require new definitions for data-summarization and synthesis tasks. In particular, for many data that are typically modeled as networks, temporal information is nowadays readily available, leading to *temporal networks* (Holme and Saramäki, 2012; Michail, 2016). In a temporal network $G = (V, E)$, edges describe interactions over a set of entities V. For each edge $(u, v, t) \in E$, the time of interaction t, between entities $u, v \in V$ is also available.

In this paper we introduce a new problem formulation for *summarizing temporal networks*. Network summarization is a well-established problem with applications to data compression, visualization, interactive analysis, and noise elimination. However, temporal network summarization is a rather novel and challenging topic, because of the large variety of temporal network summaries being proposed. An extensive survey for both static and temporal network summarization was compiled by Liu et al. (2018). Many of the temporal summaries (such as temporal motifs (Kovanen et al., 2013; Paranjape et al., 2017a), temporal graphlets (Hulovatyy et al., 2015; Lahiri and Berger-Wolf, 2008), vocabulary-based summaries (Shah et al., 2015), evolutionary patterns (Wackersreuther et al., 2010; Berlingerio et al., 2009), community evolution (Pietilänen and Diot, 2012; He and Chen, 2015)) are rather complex and may be hard to interpret. Here we propose a simple and intuitive model for activity summarization. Our main idea is to introduce an *activity model* where entities can be *active* over *latent time intervals*. An edge (interaction) between two entities can be explained if, at the time of the interaction, *at least one* of the two entities is active. Our summarization task is to find the latent activity intervals for all entities. The output of this process yields an *activity timeline* for the whole network.

Intuitively, an active entity can help "explaining" interactions of that entity with other entities. Finding short time intervals and corresponding active entities, which explain the observed interactions in the temporal network, corresponds to finding the most salient events. To further motivate our summarization task, consider the following example.

Example. Consider a stream of tweets related to global news in the first half of June 2018: the *G7 summit* takes place in Canada on June 8-9, where US president *D. Trump* spars with other world leaders about import *tariffs*, argues with the prime minister of Canada *J. Trudeau*, while many fear an imminent *trade war*. A few days later, on June 15, president *Trump* announces 25% *tariffs* on Chinese goods, prompting *China* to retaliate, and increasing *trade war* concerns, while analysts still discuss the *G7 summit* aftermath.

A standard approach to analyze such data is to create a *co-occurrence graph* of the key entities, e.g., frequent Twitter hashtags, mentioned in the news stories. A toy instance of such a hashtag co-occurrence graph for the above example is shown in the left side of Fig. 1. In this paper we consider retaining temporal information for each pair of co-occurring entities and representing the data as a *temporal network*, as shown in the right side of Fig. 1. This is clearly a richer representation providing more data-analysis opportunities. In this paper we aim to find event timelines, represented as a succinct set of (entity, time-interval) pairs that explain the observed data. The timelines for the data in Fig. 1 are shown in Fig. 2. We see two main events, corresponding to the *G7 summit* and the *tariffs on Chinese goods* announcement, described by a key entity each and corresponding time intervals. We see how the key entities explain the occurrence of the other entities, many of which are common. □

Motivated by the previous example we introduce the *network-untangling problem*, where the goal is to reconstruct an activity timeline from a temporal network. We consider a simple model in which we assume that each entity can be *active* during at most k time intervals. We say that a temporal edge (u, v, t) is *covered* if at least one of u and v is active at time t. The objective is to find a set of activity intervals, k for each entity, so that all temporal edges are covered, and the length of the activity intervals is minimized. We consider two definitions for interval length: total length (SUM) and maximum length (MAX).

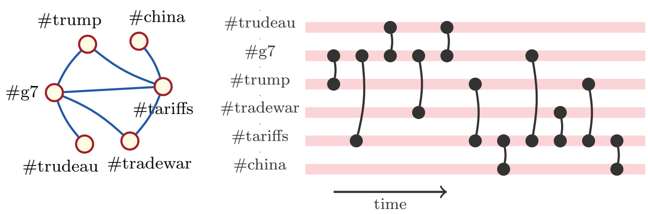

Fig. 1 A toy example motivating our problem definition. Left: co-occurrence graph of hashtags as they may appear on a social-media platform, such as Twitter. Right: more fine-grained representation of the co-occurrence hashtag graph as a temporal network; edges now include time information

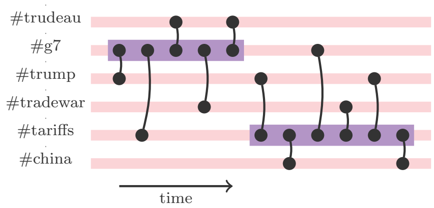

Fig. 2 The solution to the timeline discovery problem, defined in this paper, for the temporal network of Fig. 1. A timeline of events that explain all temporal edges is identified. The timeline consists of intervals during which certain hashtags are *active*

Table 1 Common notation used throughout the paper

$G=(V,E)$	network G with V nodes and E edges
n, m	number of nodes, number of edges
$t(e)$	time stamp of edge e
$E(v)$	edges adjacent to vertex v
$A(v)$	edge indices adjacent to vertex v
$T(v)$	time stamps of edges adjacent to vertex v
$\mathcal{T}$	set of activity time intervals
$I_v=[s_v,e_v]$	active time interval for v
$\sigma(I_v)$	duration of a time interval
$S(\mathcal{T})$	sum of durations activity time intervals
$\Delta(\mathcal{T})$	max of durations activity time intervals

When there is only one activity interval per node, i.e., $k=1$, we show that the MAX problem can be mapped to 2-SAT, and solved optimally in linear time. On the other hand, the SUM problem is **NP**-hard. In the general case, $k>1$, we show that both problem variants — MAX and SUM — are not only **NP**-hard but also inapproximable. We approach these problems by offering four iterative algorithms that rely on subproblems that can be solved approximately or optimally. In all cases the subproblems can be solved by linear-time algorithms, yielding overall very practical and efficient methods.

We complement our theoretical results with an experimental evaluation, where we demonstrate that our methods can find ground-truth activity intervals planted on synthetic datasets. Additionally we conduct a case study where it is shown that the discovered intervals match the timeline of real-world events and related sub-events.

The rest of the paper is organized as follows. In Section 2 we formally define the problems we study. Sections 3 and 4 are dedicated to optimizing the maximum activity interval length with $k=1$ and $k>1$. Whereas Sections 5 and 6 are describing optimizing the total activity interval length with $k=1$ and $k>1$. In all sections we establish the computational complexity of the different problem variants along with presenting our solutions. In Section 7 we discuss the related work and in Section 8 we present our experimental evaluation. Finally, Section 9 is a short conclusion.

An earlier version of this work appeared in the ECML PKDD 2017 conference (Rozenshtein et al., 2017). The conference version addressed only the single activity-interval case ($k=1$). The current version extends the problem definition and algorithms to the general case ($k>1$).

2 Preliminaries and problem definition

Let $G=(V,E)$ be a temporal network, where V is a set of vertices and E is a set of *time-stamped* edges. The edges in E are triples of the form (u,v,t), where $u,v\in V$ and t is a time stamp indicating the time that an interaction between vertices u and v takes place. The edges are undirected. We do not preclude the case that two vertices u and v interact multiple times. As it is customary, we denote by n the number of vertices in the graph, and by m the number of edges. For our algorithms we assume that the edges are given in chronological order, if not, they can be sorted in additional $\mathcal{O}(m \log m)$ time. If there are edges with the same time stamp, then any tie-breaking order among those edges will suffice.

Given a vertex $u \in V$, we write $E(u)$ to denote the set of edges adjacent to u, i.e., $E(u) = \{(u, v, t) \in E\}$. We write $N(u) = \{v \mid (u, v, t) \in E\}$ to represent the set of vertices adjacent to u, and $T(u) = \{t \mid (u, v, t) \in E\}$ to represent the set of time stamps of the edges containing u. Finally, we write $t(e)$ to denote the time stamp of an edge $e \in E$.

We denote by $A(v)$ the indices of edges that are adjacent to v. Given an edge e_i with a index i, we will often write $t(i)$ to mean $t(e_i)$.

Given a vertex $u \in V$ and two numbers s_u and e_u, we consider the interval $I_u = [s_u, e_u]$, where s_u is a start time and e_u is an end time. We refer to I_u as the *activity interval* of vertex u. Intuitively, we think of I_u as the time interval in which the vertex u has been *active*. A set of activity intervals $\mathcal{T} = \{I_u\}_{u \in V}$, one interval for each vertex $u \in V$, is an *activity timeline* for the temporal network G.

Given a temporal network $G = (V, E)$ and an activity timeline $\mathcal{T} = \{I_u\}_{u \in V}$, we say that $\mathcal{T}$ *covers* G if for each edge $(u, v, t) \in E$, we have $t \in I_u$ or $t \in I_v$, that is, for each edge in the network at least one of its endpoints is active.

Note that there is a trivial timeline that provides a cover. Such a timeline, defined by $I_u = [\min T(u), \max T(u)]$, may have unnecessarily long intervals. Instead, we aim at finding a timeline that has as compact intervals as possible. We measure the quality of a timeline by the total duration of all activity intervals in it. More formally, we define the *total span*, or *sum-span*, of a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ by

$$S(\mathcal{T}) = \sum_{u \in V} \sigma(I_u),$$

where $\sigma(I_u) = e_u - s_u$ is the duration of a single interval. An alternative way to measure the compactness of a timeline is by the duration of its longest interval,

$$\Delta(\mathcal{T}) = \max_{u \in V} \sigma(I_u).$$

We refer to $\Delta(\mathcal{T})$ as the *max-span* of the timeline $\mathcal{T}$.

For the two quality measures, sum-span and max-span, we define corresponding problem variants.

Problem 1 (MinTimeline$_+$) Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and minimizes the sum-span $S(\mathcal{T})$.

Problem 2 (MinTimeline$_\infty$) Given a temporal network $G = (V, E)$ find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and minimizes the max-span $\Delta(\mathcal{T})$.

Multiple intervals. Additionally, we extend our problem definitions to allow k active intervals per vertex. We define a k-activity timeline as a set of activity intervals $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$. Note that we allow empty intervals, so in practice some vertices may have less than k intervals.

We define the *k-sum-span* of a k-activity timeline $\mathcal{T}$ by

$$S(\mathcal{T}) = \sum_{j \in [1,k]} \sum_{u \in V} \sigma(I_{uj}),$$

where $\sigma(I_{uj}) = e_{uj} - s_{uj}$ is the duration of the j-th activity interval of vertex u.

The *max-span* of the timeline $\mathcal{T}$ is defined similarly as the duration of the longest interval,

$$\Delta(\mathcal{T}) = \max_{j \in [1,k]} \max_{u \in V} \sigma(I_{uj}).$$

We can now extend Problems MinTimeline$_+$ and MinTimeline$_\infty$ to the case of k activity intervals per vertex.

Problem 3 (k-MinTimeline$_+$) Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$ that covers G and minimizes the sum-span $S(\mathcal{T})$.

Problem 4 (k-MinTimeline$_\infty$) Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$ that covers G and minimizes the max-span $\Delta(\mathcal{T})$.

The choice between problems MinTimeline$_+$ and MinTimeline$_\infty$ depends largely on the application setting we are working with. With problem MinTimeline$_\infty$ (or k-MinTimeline$_\infty$) we obtain a worst-case bound for the length of all activity intervals. This property can be useful in scenarios where we anticipate all activity intervals to be of comparable length, for example, for creating a timeline of events with a daily cycle. On the other hand, problems MinTimeline$_\infty$ and k-MinTimeline$_\infty$ are sensitive to outliers: the presence of a single long interval, which may be caused due to noise in the data, is sufficient to lead to solutions with unreasonably large cost, despite all other intervals being sufficiently short. In this case, it is more appropriate to use problems MinTimeline$_+$ and k-MinTimeline$_+$, since they provide greater flexibility and can accomodate intervals of varying lengths. Thus, the use of problems MinTimeline$_+$ and k-MinTimeline$_+$ is recommended when there is high variability in the length of the events that we expect to discover in the activity timeline.

3 Exact algorithm solving MinTimeline$_\infty$

In this section we provide an algorithm solving MinTimeline$_\infty$ in $\mathcal{O}(n \log n)$ time. This is done by solving a budget version of the problem, and then using binary search to find the optimal cover.

3.1 Binary search method for MinTimeline$_\infty$

Our algorithm for MinTimeline$_\infty$ relies on the idea of using a subproblem that is easier to solve.

In this case, we consider as subproblem an instance in which, in addition to the temporal network G, we are also given a set of budgets $\{b_v\}$ of interval durations; one budget b_v for each vertex v. The goal is to find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers the temporal network G and the length of each activity interval I_v is at most b_v. We refer to this problem as Budget.

Problem 5 (Budget) Given a temporal network $G = (V, E)$ and a set of budgets $\{b_v\}_{v \in V}$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and satisfies $\sigma(I_v) \leq b_v$ for each $v \in V$.

Surprisingly, the Budget problem can be solved *optimally* in *linear time*. The algorithm is presented in Section 3.2. Note that this result is compatible with the **NP**-hardness of MinTimeline$_+$, since here we know the budgets for *individual* intervals, while in MinTimeline$_+$ there is one budget for the total interval length.

We can now use binary search to find the optimal value $\Delta(\mathcal{T})$. We call this algorithm `Budget`.

To guarantee a small number of iterations during binary search, some attention is required: let $T = t_1, \ldots, t_m$ be all the time stamps, sorted. Assume that we have L, the largest known infeasible budget and U, the smallest known feasible budget. To define a new candidate budget, we consider $W(i) = \{t_j - t_i \mid L < t_j - t_i < U\}$. The optimal budget is either U or one of the numbers in $W(i)$. If every $W(i)$ is empty, then the answer is U. Otherwise, we compute $m(i)$ to be the median of $W(i)$, ignoring any empty $W(i)$, and we test the median of all $m(i)$ (weighted by $|W(i)|$) as a new budget. We can show that at each iteration $\sum |W(i)|$ is reduced by 1/4, that is, only $\mathcal{O}(\log m)$ iterations are needed. We can determine the medians $m(i)$ and the sizes $|W(i)|$ in linear time since T is sorted, and we can determine the weighted median in linear time by using a modified median-of-medians algorithm. This leads to running time yielding an $\mathcal{O}(m \log m)$ algorithm. In our experiments we use a straightforward binary search by testing $(U + L)/2$ as a budget.

3.2 Exact algorithm for Budget

We develop a linear-time algorithm for problem Budget. We are given a temporal network G, and a set of budgets $\{b_v\}$, and all activity intervals should satisfy $\sigma(I_v) \leq b_v$.

The idea is to map Budget into 2-sat. To do that we introduce a boolean variable x_{vt} for each vertex v and for each timestamp $t \in T(v)$. To guarantee the solution will cover each edge (u, v, t) we add a clause $(x_{vt} \vee x_{ut})$. To make sure that we do not exceed the budget we require that for each vertex v and each pair of time stamps $s, t \in T(v)$ such that $|s - t| > b_v$ either x_{vs} is false or

x_{vt} is false, that is, we add a clause $(\neg x_{vs} \vee \neg x_{vt})$. It follows immediately, that BUDGET has a solution if and only if 2-SAT has a solution. The solution for BUDGET can be obtained from the 2-SAT solution by taking the time intervals that contain all boolean variables set to true. Since 2-SAT is a polynomially-time solvable problem (Aspvall et al., 1979), we have the following.

Proposition 1 *Problem* BUDGET *can be solved in polynomial time.*

To see this, first we recall that solving 2-SAT can be done in linear-time with respect to the number of clauses (Aspvall et al., 1979). However, in our case we may have $\mathcal{O}(m^2)$ clauses. Fortunately, the 2-SAT instances created with our mapping have enough structure to be solvable in $\mathcal{O}(m)$ time. This speed-up is described in the remainder of the section.

Let us first review the algorithm by Aspvall et al. (1979) for solving 2-SAT. The algorithm starts with constructing an *implication graph* $H = (W, A)$. The graph H is directed and its vertex set $W = P \cup Q$ has a vertex p_i in P and a vertex q_i in Q for each boolean variable x_i. The edges A are as follows: a clause $(x_i \vee x_j)$ induces two edges $(q_i \to p_j)$ and $(q_j \to p_i)$, a clause $(\neg x_i \vee \neg x_j)$ induces two edges $(p_i \to q_j)$ and $(p_j \to q_i)$, and a clause $(x_i \vee \neg x_j)$ induces two edges $(q_i \to q_j)$ and $(p_j \to p_i)$.

In our case, the edges A are divided to two groups A_1 and A_2. The set A_1 contains two directed edges $(q_{vt} \to p_{ut})$ and $(q_{ut} \to p_{vt})$ for each edge $e = (u, v, t) \in E$. The set A_2 contains two directed edges $(p_{vt} \to q_{vs})$ and $(p_{vs} \to q_{vt})$ for each vertex v and each pair of time stamps $s, t \in T(v)$ such that $|s - t| > b_v$. Note that A_1 goes from Q to P and A_2 goes from P to Q. Moreover, $|A_1| \in \mathcal{O}(m)$ and $|A_2| \in \mathcal{O}(m^2)$.

Next, we decompose H in strongly connected components (SCC), and order them topologically. If any strongly connected component contains both p_{vt} and q_{vt}, then we know that 2-SAT is not solvable. Otherwise, to obtain the solution, we start enumerating over the components, children first: if the boolean variables corresponding to the vertices in the component do not have a truth assignment,[1] then we set x_{vt} to be true if p_{vt} is in the component, and x_{vt} to be false if q_{vt} is in the component

The bottleneck of this method is the SCC decomposition, which requires time $\mathcal{O}(|W| + |A|)$, and the remaining steps can be done in $\mathcal{O}(|W|)$ time. Since $|W| \in \mathcal{O}(m)$, we need to be able to perform the SCC decomposition in time $\mathcal{O}(m)$. We will use the algorithm by Kosajaru (see Hopcroft and Ullman (1983)), which consists of two depth-first search (DFS) computations, performing constant-time operations on each visited vertex. Thus, we need to only optimize the DFS.

To speed-up the DFS computation, we design an oracle such that given a vertex $p \in P$ it returns an *unvisited* neighboring vertex $q \in Q$ in *constant* time. Since $|Q| \in \mathcal{O}(m)$, DFS spends at most $\mathcal{O}(m)$ time processing vertices $p \in P$. On the other hand, if we are at $q \in Q$, then we can use the standard

[1] Due to the property of implication graph, either all or none variables will be set in the component.

DFS to find the neighboring vertex $p \in P$. Since $|A_1| \in \mathcal{O}(m)$, this guarantees that DFS spends at most $\mathcal{O}(m)$ time processing vertices $q \in Q$.

Next, we describe the oracle: first we keep the unvisited vertices Q in lists $\ell[v] = (q_{vt} \in Q; q_{vt}$ is not visited) sorted chronologically. Assume that we are at $p_{vt} \in P$. We retrieve the first vertex in $\ell[v]$, say q_{vs}, and check if $|s - t| > b_v$. If true, then q_{vs} is a neighbor of p_{vt}, so we return q_{vs}. Naturally, we delete q_{vs} from $\ell[v]$ the moment we visit q_{vs}. If $|s - t| \leq b_v$, then test similarly the *last* vertex in $\ell[v]$, say $q_{vs'}$. If both $q_{vs'}$ and q_{vs} are non-neighbors of p_{vt}, then, since $\ell[v]$ is sorted chronologically, we can conclude that $\ell[v]$ does not have unvisited neighbors of p_{vt}. Since p_{vt} does not have any neighbors outside $\ell[v]$, we conclude that p_{vt} does not have any unvisited neighbors.

Using this oracle we can now perform DFS in $\mathcal{O}(m)$ time, which in turns allows us to do the SCC decomposition in $\mathcal{O}(m)$ time, which solves Budget in $\mathcal{O}(m)$ time.

4 Algorithm for k-MinTimeline$_\infty$

Next, we consider a k-extension of MinTimeline$_\infty$. Unlike the simpler previous problem, this extension is not only computationally infeasible but also inapproximable.

Proposition 2 *k-MinTimeline$_\infty$ and k-MinTimeline$_+$ are inapproximable, unless $\boldsymbol{P} = \boldsymbol{NP}$.*

Proof The proof can be found in Appendix A.

4.1 Iterative method with budgets for k-MinTimeline$_\infty$

As a heuristic we consider two nested subproblems. In the first one, k-Partition, we assume that for each node we are given a set of $k - 1$ points $\{g_{vi}\}_{v \in V, i \in [1,k-1]}$, which belong to the gaps between k activity intervals (i.e., *inactive points*).

More formally, given a timeline $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$, we say that $\{g_{vi}\}$ *interleaves* with $\mathcal{T}$ if g_{vi} is between I_{vi} and $I_{v(i+1)}$, where we do not allow g_{vi} to "touch" the border of the intervals. Among these timelines, we look for the ones that cover all interactions and minimize the maximum length of an interval.

Problem 6 (k-Partition) Given a temporal network $G = (V, E)$ and a set of inactive points $\{g_{vi}\}_{v \in V, i \in [1,k-1]}$, find a timeline $\mathcal{T} = \{I_{uj}\}_{u \in V, j \in [1,k]}$ that covers G, interleaves with $\{g_{vi}\}$, and minimizes the max-span $\Delta(\mathcal{T})$.

Problem k-Partition can be solved in polynomial time through iteration of Problem k-Budget, which sets a budget for each interval.

Problem 7 (k-Budget) Given a temporal network $G = (V, E)$, a set of budgets $\{b_{vj}\}_{v \in V, j \in [1,k]}$, and a set of inactive points $\{g_{vi}\}_{v \in V, i \in [1,k-1]}$, find a timeline $\mathcal{T} = \{I_{uj}\}_{u \in V, j \in [1,k]}$ that covers G, interleaves with $\{g_{vi}\}$, and $\sigma(I_{vj}) \leq b_{vj}$ for each $v \in V$ and $j \in [1, k]$.

Given an algorithm for k-Budget we use binary search for budgets to find the optimal value $\Delta(\mathcal{T})$. To solve k-MinTimeline$_\infty$ we can apply an iterative heuristic similar to `k-Inner`: we guess $k-1$ initial inactive points for each node $u \in V$, solve k-Partition optimally by binary search over the given budgets, update the set of inactive points and repeat until there is no improvement.

In the experiments we use the following simple and natural strategy to initialize and update the inactive points: to initialize the inactive points for a fixed node $u \in V$ we find the $k-1$ largest intervals between consecutive time stamps of interactions of u. Then g_{vi} is set to the mean point of the i-th interval. Once k-Partition outputs the timeline $\mathcal{T} = \{I_{uj}\}_{u \in V, j \in [1,k]}$, we update g_{vi} as the mean between two consecutive activity intervals: $g_{vi} = (s_{I_{vi}} + e_{I_{v(i+1)}})/2$. We iterate the process until there is no improvement. In our experiments, we refer to this algorithm as `k-Budget`.

4.2 Exact algorithm for k-Budget

We extend the previous approach to k-MinTimeline$_\infty$.

We again map k-Budget to 2-sat. We introduce a boolean variable x_{vt} for each vertex v and for each timestamp $t \in T(v)$ and add a clause $(x_{vt} \vee x_{ut})$. Then for each vertex v and each pair of time stamps $t_1, t_2 \in T(v)$, which lie between the two consecutive gap points g_{vi} and $g_{v(i+1)}$ (or between the last/first gap point and corresponding end point of the time-series) and have $|t_1 - t_2| > b_{vi}$, we add a clause $(\neg x_{vt_1} \vee \neg x_{vt_2})$.

Similarly to Budget, we solve this instance of 2-sat using speed-up oracles for DFS. The only difference is that DFS oracle for Q vertices now keeps k chronologically ordered lists of unvisited points for each node $v \in V$, each list $l_i[v]$ with $i = [1, \ldots, k]$ contains points q_{vt} between two consecutive gap points with timestamps $t(g_{v(i-1)})$ and $t(g_{vi})$ (or between a border point of the timeseries and a gap point in cases of $i = 1$ and $i = k$). We keep additional index to identify in constant time to which list l_i point p_{vt} belongs and thus total complexity remains $\mathcal{O}(m)$.

5 Algorithm for MinTimeline$_+$

We next consider MinTimeline$_+$. Unlike, MinTimeline$_\infty$ this problem is **NP**-hard.

Proposition 3 *The decision version of the* MinTimeline$_+$ *problem is **NP**-complete. Namely, given a temporal network $G = (V, E)$ and a budget ℓ, it is*

***NP**-complete to decide whether there is timeline $\mathcal{T}^* = \{I_u\}_{u \in V}$ that covers G and has sum-span $S(\mathcal{T}^*) \leq \ell$.*

Proof The proof can be found in Appendix A.

5.1 Inner point iterative method for MinTimeline$_+$

We now turn our attention to algorithm for solving MinTimeline$_+$. Our approach is to consider a meaningful subproblem. Assume that we are given a temporal network $G = (V, E)$ and a set of time points $\{m_v\}_{v \in V}$, i.e., one time point m_v for each vertex $v \in V$, and we ask whether we can find an optimal activity timeline $\mathcal{T} = \{I_u\}_{u \in V}$ so that each interval I_v contains the corresponding time point m_v, i.e., $m_v \in I_v$, for each $v \in V$. This problem definition is useful when we know one time point that each vertex was active, and we want to extend this to an optimal timeline. We refer to this problem as Coalesce.

Problem 8 (Coalesce) Given a temporal network $G = (V, E)$ and a set of inner time points $\{m_v\}_{v \in V}$, find a timeline $\mathcal{T} = \{I_v\}_{v \in V}$ that covers G, satisfies $m_v \in I_v$ for each $v \in V$, and minimizes the sum-span $S(\mathcal{T})$.

Interestingly, we can show that the Coalesce problem can be approximated within a factor of 2 in *linear time*. This 2-approximation algorithm is presented in Section 5.2.

Motivated by Coalesce, we propose an algorithm for MinTimeline$_+$, which uses Coalesce as a subroutine: initialize $m_v = (\min T(v) + \max T(v))/2$ to be an inner time point for vertex v. We then use our approximation algorithm for Coalesce to obtain a set of intervals $\{I_v\} = \{[s_v, e_v]\}_{v \in V}$. We use these intervals to set the new inner points, $m_v = (s_v + e_v)/2$, and repeat until the score no longer improves. We call this algorithm `Inner`.

5.2 Algorithm for Coalesce

As noted in Problem 8, the input to the Coalesce problem is a temporal network $G = (V, E)$ and a set of interior time points $\{m_v\}_{v \in V}$.

Consider a vertex v and the corresponding interior point m_v. For an edge index i we define the *peripheral indices* $p(i; v)$ to be the indices that are on the other side of i than m_v (see Figure 3 for example),

$$p(i; v) = \begin{cases} \{j \mid j \in A(v)\,,\ j \geq i\} & \text{if } t(i) \geq m_v, \\ \{j \mid j \in A(v)\,,\ j \leq i\} & \text{if } t(i) < m_v. \end{cases}$$

Our next step is to express Coalesce as an integer linear program. Let $x_{vi} \in \{0, 1\}$ be a variable for each vertex $v \in V$ and index $i \in A(v)$. Instead of going

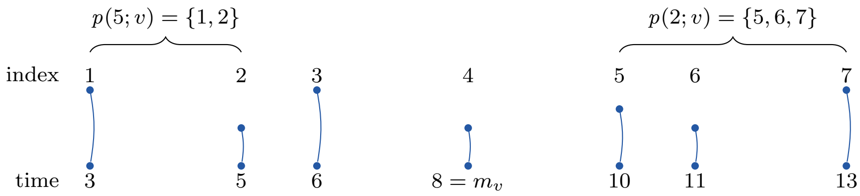

Fig. 3 Toy example of peripheral indices used in a linear program solving COALESCE.

for the obvious construction, where $x_{vi} = 1$ indicates that v is active at time i, we follow a different formulation: in our program $x_{vi} = 1$ indicates that $t(i)$ is either the *beginning* or the *end* of the active region of v. It follows that the integer program

$$\begin{aligned} &\min \sum_{v \in V, i \in A(v)} |t(i) - m_v| x_{vi}, \\ &\text{such that} \sum_{j \in p(i;v)} x_{vj} + \sum_{j \in p(i;u)} x_{uj} \geq 1, \ \text{for all } e_i = (u,v,t) \in E \end{aligned}$$

solves COALESCE. Minimizing the first sum corresponds to minimizing the sum-span of the timeline, while the constraint on the second sum ensures that the resulting timeline covers the temporal network. Note that we do not require that each vertex should have exactly one beginning and one end. However, the minimality of the optimal solution ensures that this constraint will be satisfied, too.

Relaxing the integrality constraint and considering the program as linear program, allows us to write the dual. The variables in the dual can be viewed as positive weights α_e on the edges, with the goal of maximizing the total sum of these weights. Let us write α_i for α_{e_i}.

To express the constraints on the dual, let us define an auxiliary function $h(v;i)$ as the sum of the weights of adjacent edges between i and m_v,

$$h(v;i) = \begin{cases} \sum \{\alpha_j \mid j \in A(v)\,,\ j \leq i, t(j) \geq m_v\} & \text{if } t(i) \geq m_v, \\ \sum \{\alpha_j \mid j \in A(v)\,,\ j \geq i, t(j) < m_v\} & \text{if } t(i) < m_v. \end{cases}$$

The dual can now be formulated as

$$\begin{aligned} &\max \sum_{e_j \in E} \alpha_j, \\ &\text{such that } \ h(v;i) \leq |t(i) - m_v|, \ \text{for all } v \in V, \ i \in A(v)\,, \end{aligned}$$

that is, we maximize the total weight of edges such that for each vertex v and for each index i, the sum of adjacent edges is bounded by $|t(i) - m_v|$.

We say that the solution to dual is *maximal* if we cannot increase any edge weight α_e without violating the constraints. An optimal solution is maximal but a maximal solution is not necessarily optimal.

Algorithm 1: `Maximal`, yields 2-approximation to Coalesce.

$b[v] \leftarrow \infty$ for $v \in V$;
$a[v] \leftarrow 0$ for $v \in V$;
foreach $e = (u, v, t) \in E$ in chronological order **do**
  $\alpha_e \leftarrow \min\{z(u), z(v)\}$; {see Eq. (1)}
  if $t < m_v$ **then** $b[v] \leftarrow \min\{b[v] - \alpha_e, m_v - t - \alpha_e\}$;
  else $a[v] \leftarrow a[v] + \alpha_e$;
  if $t < m_u$ **then** $b[u] \leftarrow \min\{b[u] - \alpha_e, m_u - t - \alpha_e\}$;
  else $a[u] \leftarrow a[u] + \alpha_e$;

Our next result shows that a maximal solution can be used to obtain a 2-approximation cover.

Proposition 4 *Consider a maximal solution* $\{\alpha_e\}_{e \in E}$ *to the dual program. Define a set of intervals* $\mathcal{T} = \{I_v\}$ *by* $I_v = [\min\{t(i) \mid i \in X_v\}, \max\{t(i) \mid i \in X_v\}]$*, where*

$$X_v = \{i \in A(v) \mid h(v; i) = |t(i) - m_v|\}.$$

Then $\mathcal{T}$ *is a 2-approximation solution for the problem* Coalesce.

Proof The proof can be found in Appendix B.

We have established that as long as we can obtain a maximal solution for the dual, we can extract a timeline that is 2-approximation.

Next, we introduce a linear-time algorithm that computes a maximal dual solution. The algorithm visits each edge e in chronological order and increases α_e as much as possible without violating the dual constraints. To obtain a linear-time complexity we need to determine in *constant* time by how much we can increase α_e. The pseudo-code is given in Algorithm 1, and the remaining section is used to prove the correctness of the algorithm.

As the algorithm goes over the edges, we maintain two counters per each vertex, $a[v]$ and $b[v]$. Let $e_j = (u, v, t)$ be the current edge. The counter $a[v]$ is maintained only if $t \geq m_v$, and the counter $b[v]$ is maintained if $t < m_v$. Our invariant for maintaining the counters $a[v]$ and $b[v]$ is that at the beginning of the j-th round they are equal to

$$a[v] = h(v; j) \quad \text{and} \quad b[v] = \min_{i<j}\{t(i) - m_v - h(v; i)\},$$

Moreover, we have weights $\alpha_i = 0$, for $i \geq j$. The following lemma tells us how to update α_j using $a[v]$ and $b[v]$.

Lemma 1 *Assume that we are processing edge* $e_j = (u, v, t)$*. We can increase* α_j *by at most* $\min\{z(u), z(v)\}$,

$$\textit{where} \quad z(w) = \begin{cases} t - m_w - a[w] & \textit{if } t(j) \geq m_w, \\ \min\{m_w - t, b[w]\} & \textit{if } t(j) < m_w. \end{cases} \tag{1}$$

Proof The proof can be found in Appendix B.

Our final step is to how to maintain $a[v]$ and $b[v]$. Maintaining $a[v]$ is trivial: we simply add α_j to $a[v]$. The new $b[v]$ is equal to

$$\min_{i \leq j} \{t(i) - m_v - h(v; i)\} = \min\{b[v] - \alpha_j, m_v - t(j) - \alpha_j\}.$$

Clearly the counters $a[v]$ and $b[v]$ and the dual variables α_e can be maintained in constant time per edge processed, making `Maximal` a linear-time algorithm.

6 Algorithm for k-MinTimeline$_+$

Our final algorithm is for k-MinTimeline$_+$. We have already showed that k-MinTimeline$_+$ is inapproximable in Proposition 2. Hence, we propose an iterative method, similar to `Inner`.

6.1 Inner point iterative method for k-MinTimeline$_+$

As with MinTimeline$_+$ we consider a subproblem that can be solved efficiently with an approximation guarantee. In particular, we assume that we are given a set of k time points for each node, $\{m_{vj}\}_{v \in V, j \in [1,k]}$. We ask to find an optimal activity timeline $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$ so that the interval I_{vj} of vertex v contains the corresponding time point m_{vj}, that is, $m_{vj} \in I_{vj}$, for each $v \in V$ and $j \in [1, k]$. These inner points can be located *anywhere* within the interval. We refer to this problem as k-Coalesce.

Problem 9 (k-Coalesce) Given a temporal network $G = (V, E)$ and a set of time points $\{m_{vj}\}_{v \in V, j \in [1,k]}$, find a timeline $\mathcal{T} = \{I_{vj}\}_{v \in V, j \in [1,k]}$ that covers G, satisfies $m_{vj} \in I_{vj}$ for each $v \in V$ and $j \in [1, k]$, and minimizes the sum-span $S(\mathcal{T})$.

Below we show how to extend our approach for Problem Coalesce and design a 2-approximation linear time algorithm. Given an algorithm for k-Coalesce, we can guess initial k active time points for each node and iterate solving k-Coalesce and updating the inner points.

In the experiments, we initialize $\{m_{vj}\}_{v \in V, j \in [1,k]}$ by using the centroids of a k-clustering algorithm, performed on the time-stamps of the edges that contain vertex v. In particular, we start with a clustering that minimizes the total diameter of the clusters. Such a clustering can be obtained efficiently by locating the $k - 1$ largest intervals between two consecutive interactions of v in E. Then m_{vj} is set to be the mean of the cluster interval j. After solving k-Coalesce we update m_{vj}'s as the middle points of the new activity interval and we iterate until there is improvement in the solution. We call this algorithm `k-Inner`.

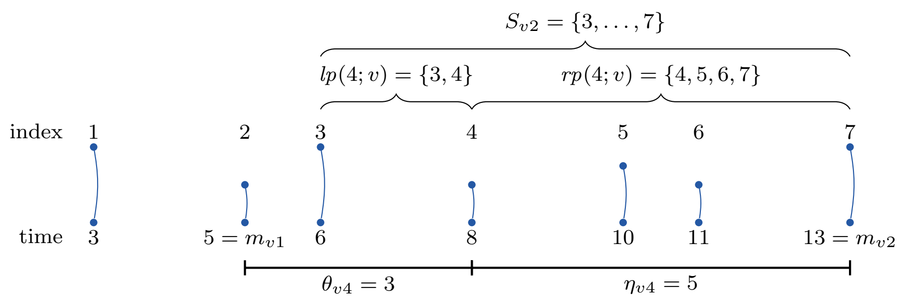

Fig. 4 Toy example of variables used by the linear program for solving k-Coalesce. Note that S_{v2} does not contain edge index 2 but contains 13. This is done due to notational convenience when proving the correctness of `k-Maximal`.

6.2 Algorithm for k-Coalesce

Next, we show how to solve k-Coalesce. Assume we are given k inner points for each vertex, denoted by m_{vi}. Let us write $m_{v0} = -\infty$ and $m_{v(k+1)} = \infty$.

The middle points divide the timeline of each vertex in $k + 1$ segments,

$$S_{v\ell} = \left\{ i \in A(v) \mid m_{v(\ell-1)} < t(i) \leq m_{v\ell} \right\}.$$

We extend the definition of *peripheral time stamps* to handle k inner points. Next we define *peripheral time stamps*. In order to handle multiple inner points, we will explicitly differentiate left time stamps and right time stamps. Given an edge index i that belongs to $S_{v\ell}$ we define

$$lp(i; v) = \{ j \in S_{v\ell} \mid j \leq i \}, \quad \text{and} \quad rp(i; v) = \{ j \in S_{v\ell} \mid j \geq i \},$$

that is we split $S_{v\ell}$ in two halves at index i. See Figure 4 for a toy example.

For the integer linear programming we define two variables x_{vi}, y_{vi} for each vertex $v \in V$, and an edge index $i \in A(v)$ The assignment $x_{vi} = 1$ indicates that $t(i)$ is the *end* of an active interval while $y_{vi} = 1$ indicates that $t(i)$ is the *beginning* of an active interval.

It follows that the integer program with $x_{vj}, y_{vj} \in \{0, 1\}$

$$\begin{aligned} \min \quad & \sum_{v, i \in A(v)} (m_{v\ell} - t(i)) x_{vi} + (t(i) - m_{v(\ell-1)}) y_{vi}, \\ \text{such that} \quad & \sum_{j \in lp(i;v)} x_{vj} + \sum_{j \in rp(i;v)} y_{vj} + \sum_{j \in lp(i;u)} x_{uj} + \sum_{j \in rp(i;u)} y_{uj} \geq 1, \\ & \text{for all } e_i = (u, v, t) \in E \end{aligned}$$

solves k-Coalesce.

We relax the integrality constraint and write the dual. The variables in the dual can be viewed as positive weights α_e on the edges, with the goal of maximizing the total sum of these weights.

As before, we write α_i for α_{e_i}. Given an edgex index i, we define two auxiliary functions summing the weights eihter from the left side or from the right side,

$$lh(v;i) = \sum_{j \in lp(i;v)} \alpha_j, \quad \text{and} \quad rh(v;i) = \sum_{j \in rp(i;v)} \alpha_j.$$

Let $e_i = (u, v, t)$ be an edge. Let ℓ be the index such that $i \in S_{v\ell}$. We define the distances of t to the inner points as

$$\theta_{vi} = t - m_{v(\ell-1)} \quad \text{and} \quad \eta_{vi} = m_{v\ell} - t.$$

See Figure 4 for a toy example.

The dual can now be formulated as

$$\begin{aligned} \max \quad & \sum_{e \in E} \alpha_e, \\ \text{such that} \quad & lh(v;i) \leq \theta_{vi} \text{ and } rh(v;i) \leq \eta_{vi}, \quad \text{for all } v \in V, i \in A(v)\,. \end{aligned}$$

Recall that the solution to dual is *maximal* if we cannot increase any edge weight α_e without violating the constraints. However, unlike in the previous section, this will not be enough for us, and we need a stronger condition.

Assume that $\{\alpha_e\}$ is a maximal solution. Let $e_i = (u, v, t)$ be an edge. We say that e_i is *left-maximal* if at least one of the four following cases hold

(i) $rh(v;j) = \eta_{vj}$, for some $j \in lp(i;v)$,
(ii) $rh(u;j) = \eta_{uj}$, for some $j \in lp(i;u)$,
(iii) $lh(v;i) = \theta_{vi}$,
(iv) $lh(u;i) = \theta_{ui}$.

Note that the last two cases are more strict than the regular maximality. If all edges are left-maximal, then we say that $\{\alpha_e\}$ is left-maximal. The next proportion shows how to obtain a solution for k-Coalesce given a left-maximal solution for the dual.

Proposition 5 *Let $\{\alpha_i\}$ be a left-maximal solution. Define*

$$X_{v\ell} = \{t(i) \mid i \in S_{v\ell}, rh(v;i) = \eta_{vi}\}\,,$$

and let $x_{v\ell} = \min(X_{v\ell} \cup \{m_{v\ell}\})$. Define also

$$Y_{v\ell} = \{t(i) \mid i \in S_{v\ell}, lh(v;i) = \theta_{vi}, t(i) < x_{v\ell}\}\,,$$

and $y_{v\ell} = \max(Y_{v\ell} \cup \left\{m_{v(\ell-1)}\right\})$. Define a set of intervals

$$\mathcal{T} = \left\{[x_{v\ell}, y_{v(\ell+1)}]\right\}_{v \in V, \ell = 1, \ldots, k}\,.$$

Then $\mathcal{T}$ is a 2-approximation solution for k-Coalesce.

Proof The proof can be found in Appendix C.

Algorithm 2: `k-Maximal`, produces a left-maximal dual solution, yielding 2-approximation to k-Coalesce.

$b[v, \ell] \leftarrow \infty$ for $v \in V$;
$a[v, \ell] \leftarrow 0$ for $v \in V$;
foreach $e_i = (u, v, t) \in E$ in chronological order **do**
  $\ell \leftarrow$ smallest index with $m_{v\ell} \geq t$;
  $r \leftarrow$ smallest index with $m_{ur} \geq t$;
  $z_1 \leftarrow \min(\theta_{vi} - a[v, \ell], \eta_{vi}, b[v, \ell])$;
  $z_2 \leftarrow \min(\theta_{ui} - a[u, r], \eta_{ui}, b[u, r])$;
  $\alpha_e \leftarrow \min(z_1, z_2)$;
  $a[v, \ell] \leftarrow a[v, \ell] + \alpha_e$;
  $a[u, r] \leftarrow a[u, r] + \alpha_e$;
  $b[v, \ell] \leftarrow \min\{b[v, \ell] - \alpha_e, \eta_{vi} - \alpha_e\}$;
  $b[u, r] \leftarrow \min\{b[u, r] - \alpha_e, \eta_{ui} - \alpha_e\}$;

We now move to describe the algorithm for obtaining a left-maximal solution. The pseudo-code, given in Algorithm 2, is an extension of `Maximal`. Similarly, it visits each edge $e = (u, v, t)$ in chronological order and increases α_e as much as possible without violating the dual constraints. Since we process edges in chronological order, we also ensure the left-maximality. Time complexity is linear as we determine in *constant* time by how much we can increase α_e.

As the algorithm goes over the edges, we maintain $2(k + 1)$ counters per each vertex, $a[v, \ell]$ and $b[v, \ell]$. These counters are only maintained when we are processing $S_{v\ell}$. Note that at the extreme segments, we only need one counter but for notational simplicity we keep updating both.

Finally, we establish that `k-Maximal` yields 2-approximation to k-Coalesce.

Proposition 6 *The solution* $\{\alpha_i\}$ *provided by* `k-Maximal` *is feasible and left-maximal.*

Proof The proof can be found in Appendix C.

7 Related work

To the best of our knowledge, the problem we consider in this paper has not been studied before in the literature. In this section we review briefly the lines of work that are most closely related to our setting.

Vertex cover. Our problem definition can also be considered a temporal version of the classic vertex-cover problem, one of 21 original **NP**-complete problems in Karp's seminal paper (Karp, 1972). A factor-2 approximation is available for vertex cover, by taking all vertices of a maximal matching (Hartmanis, 1982). Slightly improved approximations exist for special cases of the problem, while assuming that the unique games conjecture is true, the minimum vertex cover cannot be approximated within any constant factor better than 2 (Khot and Regev, 2008). Nevertheless, our formulation cannot be mapped directly to the static vertex-cover problem, thus, the proposed solutions need to be tailor-made for the temporal setting.

Modeling and discovering burstiness on sequential data. Modeling and discovering bursts in time sequences is a very well-studied topic in data mining. In a seminal work, Kleinberg (2003) discovered burstiness using an exponential model over the delays between the events. Ihler et al. (2006) proposed an alternative approach by modelling event counts in a sliding window with a Poisson process. Similarly, Fung et al. (2005) used a Binomial distribution. Zhu and Shasha (2003) used wavelet analysis to detect bursts for multiple windows. Vlachos et al. (2004) defined a distance between two bursts that can be used to query similar bursts. He and Parker (2010) modelled burst events as dynamic physical systems. Finally, Lappas et al. (2009) used the notion of discrepancy to mine maximal bursts.

A highly related approach for discovering bursty events is segmentation, where the goal is to segment a sequence in k coherent pieces, so that each period of high activity occurs in its own segment. If the overall score is additive with respect to the segments, then this problem can be solved in $\mathcal{O}(n^2k)$ time (Bellman, 1961). Moreover, under mild assumptions one can obtain a $(1+\epsilon)$ approximation in linear time (Guha et al., 2006; Tatti, 2019).

The difference of all these works with our setting is that we consider network data, i.e., sequences of interactions among pairs of entities. By assuming that for each interaction only one entity needs to be active, our problem becomes highly combinatorial. In order to counter-balance the increased combinatorial complexity, we consider a simpler burstiness model than previous works. However, even with this simplification the previous methods cannot be applied. Instead, we devise novel combinatorial solutions.

Mining temporal networks. Our work also falls in the broad area of mining *temporal networks* (Holme and Saramäki, 2012; Michail, 2016). In the last few years a lot of research has been devoted in the study and analysis of temporal networks. Areas of interest include work on counting network motifs (Kovanen et al., 2013; Paranjape et al., 2017b), finding temporal communities/temporal clusters (Dakiche et al., 2019; Rossetti and Cazabet, 2018; Hartmann et al., 2016), summarization temporal networks (Liu et al., 2018), developing streaming algorithms for efficient computation over temporal networks (McGregor, 2014), and more. A recent tutorial on mining temporal networks was presented in KDD 2019 (Rozenshtein and Gionis, 2019).

Temporal networks summarization. Our work is related to networks summarization as we aim to construct a compact representation of activity in a temporal network. This topic has been recently extensively studies and an overview of the advances can be found in a survey (Liu et al., 2018). The diversity of models and approaches is vast. Some notable approaches include temporal motif (Kovanen et al., 2013; Paranjape et al., 2017a) and graphlet counting (Hulovatyy et al., 2015; Lahiri and Berger-Wolf, 2008). Others use structural and behavioral vocabulary to describe the network as concise as possible (Shah et al., 2015). Another direction of summarization is searching for temporal backbones (Bogdanov et al., 2011) or evolving communities (Pietilänen and Diot, 2012; He and Chen, 2015) and evolutionary patterns (Wackersreuther et al., 2010; Berlingerio et al., 2009). To the best of our knowledge our timeline summary is not directly related to any of temporal network summarization models from the literature.

In more detail, the network-untangling problem can be considered an event-detection problem, where the goal is to find time intervals and/or sets of nodes with high activity. Typical event-detection methods use text or other meta-data, as they reveal event semantics. One line of work is based on constructing different types of word graphs (Cataldi et al., 2010; Weng and Lee, 2011; Meladianos et al., 2015). Events are detected as clusters in such graphs, however, temporal information is not considered directly.

Another family of methods uses statistical modeling for identify events as trends (Mathioudakis and Koudas, 2010; Becker et al., 2011). Leskovec et al. (2009) and Yang and Leskovec (2011) consider spreading of short quotes in the citation network of social media. These methods rely on clustering "bursty" keywords. Our setting is considerably different as we focus on interactions between entities and explicitly model entity activity by continuous time intervals.

In a different line of work, researchers have considered the problem of identifying state changes in a temporal network, and segmenting the network timeline into a small set of states, e.g., day vs. night, or weekdays vs. weekend. Gauvin et al. (2014) approach this problem via a tensor-factorization approach, while Masuda and Holme (2019) extract high-level features from network snapshots and segment the timeline by a hierarchical clustering algorithm applied on the feature representation. Our work is distinct, as we are not considering global network features and we are not asking to segment the network timeline, instead we aim to *cover* the temporal interactions with vertex-centered time intervals.

Information maps. From an application point-of-view, our work is loosely related with papers that aim to process large amounts of data and create maps that present the available information in a succinct and easy-to-understand manner. Shahaf et al. have considered this problem in the context of news articles (Shahaf et al., 2012b, 2013) and scientific publications (Shahaf et al., 2012a). However, their approach is not comparable to ours, as their input is a set of documents and not a temporal network, and their output is a "metro map" and not an activity timeline.

8 Experimental evaluation

In this section we empirically evaluate the performance of our methods.[2]

[2] The implementation of all algorithms and sample scripts used for the experimental evaluation is available at `https://github.com/polinapolina/the-network-untangling-problem`.

8.1 Setup

We first test the algorithms on synthetic datasets and then present a case study on a real-world social-media dataset.

Single interval. For the case of a single activity interval per vertex ($k = 1$) we generate dataset *Synthetic*. We first generate a static network of $n = 100$ vertices with a power-law degree distribution (we use the configuration model (Newman, 2003) with power-law exponent set to 2.0). Then for every vertex we generate a ground-truth activity interval and add $m = 100$ interactions with random neighbors. These interactions are used to construct an activity interval of length $\ell = 99$ time units: two interactions are placed on the borders of the interval, so that we can ensure the interval length, other interactions are places uniformly at random at any continuous time moment inside the interval.

We combine these intervals with varying degree of overlap. For controlling the overlap we use a parameter $p \in [0, 0.01, 0.02, \ldots, 0.99, 1]$, indicating the proportion of time stamps overlapping between two intervals: we set the starting point of the ith interval to be $1 + m(1 - p)(i - 1)$.

When $p = 0$, all intervals are disjoint and every time stamp is guaranteed to have only one interaction, thus, it should be easy to find the correct activity intervals. E.g., $p = 0.01$ will mean that the second activity interval starts at the same time stamp when the first one ends, so two adjacent activity intervals have one time stamp in common, but the length the interval overlap is 0 time units. $p = 0.02$ will mean that adjacent activity intervals have an overlap of 1 time unit, etc. Note that as p increases above 0.5 non-adjacent intervals also start to overlap, which creates a much more challenging setting. When $p = 1$, all activity intervals have exactly the same position, so there is a large number of solutions whose score is even better than the ground-truth solution. In all cases *Synthetic* has 10 000 interactions in total. Unless specified, we report results averaged over 100 runs and test a fairly complex case of overlap parameter $p = 0.5$.

Multiple intervals. For the problem version with $k > 1$ activity intervals per vertex we generate dataset *Synthetic*$_k$. We use the same construction method as for *Synthetic*, but plant $k = 10$ activity intervals of length $\ell = 9$ time units for each vertex. Each interval has $m = 10$ interactions with random neighbors, 2 of the interactions are fixed to mark the start and the end of an activity interval, the rest are placed uniformly at random inside the interval. To distribute the activity intervals on the timeline, we produce k permutations of the nodes π_r, $r = 1, \ldots, k$. Denote the position of node j in rth permutation as $\pi_r(j)$. We control overlaps using a parameter $p \in [0, 0.1, 0.2, \ldots, 0.9, 1]$: we set the starting point of the ith interval for node j to be $1 + m(1 - p)(\pi_i(j) - 1) + mn(1 - p)(i - 1)$. *Synthetic*$_k$ has also 10 000 interactions in total.

Baselines. We compare our algorithms against simple greedy baselines. Before introducing the baselines, we extend the notations as following. Given a subset of temporal edges $E' \subseteq E$ and a node $u \in V$, denote $E'(u)$ a set of edges in

Algorithm 3: `Baseline`, produces a baseline timeline $\mathcal{T}_{BL}$ for $G = (E, V)$

$E' \leftarrow E$;
$\mathcal{T}_{BL} \leftarrow \emptyset$;
while $E' \neq \emptyset$ **do**
  $u = \arg\min_{v \in V(E')} S\big(I_k^{min}(u \mid E')\big) / |E'(u)|$;
  $\mathcal{T}_{BL} = \mathcal{T}_{BL} \cup \{I_m(u \mid E')\}$;
  $E' = E' \setminus E'(u)$;
return $\mathcal{T}_{BL}$

E' adjacent to u: $E'(u) = \{(u, v, t) \in E'\}$. Denote a set of nodes, which participate in the interactions E' as $V(E')$. Denote a set of $k \geq 1$ intervals of minimum total length, which covers all interactions $E'(u)$, as $I_k^{min}(u \mid E')$ and their total length as $S\big(I_k^{min}(u \mid E')\big)$. For $k = 1$, interval $I_1^{min}(u \mid E')$ is simply an interval between the first and the last interactions of u in E'. For $k > 1$, these intervals can be computed by finding $k - 1$ longest intervals between adjacent uncovered interactions of u, and setting them to be the endpoints of the activity intervals.

Algorithm `Baseline` produces a timeline $\mathcal{T}_{BL}$ with at most $k \geq 1$ activity intervals per node. Starting from an empty set of activity intervals, the algorithm iteratively picks a node u and adds to the timeline $\mathcal{T}_{BL}$ k intervals $I_k^{min}(u \mid E')$, which cover *all uncovered* interactions of u. Node u is picked based on its relative cost: the minimum length of k intervals, which cover all uncovered interactions of u, divided by the number of uncovered interactions it participates in.

This strategy prevents us from selecting too long and sparse intervals, as we select the densest I_k^{min} with respect to currently uncovered interactions. The strategy is inspired by a classic greedy algorithm for weighted set cover (Chvatal, 1979). If we were to solve a minimum total length timeline construction problem which can include only either activity intervals covering *all* interactions $E(u)$ of some node u or empty intervals, the greedy leads to a $\log(|E|)$-approximation.

For $k = 1$ we refer to the baseline as `1-Baseline` and for $k > 1$ to as `k-Baseline`.

Evaluation metrics. To evaluate the quality of the discovered activity intervals we compare the set of discovered intervals with the ground-truth intervals. For each vertex u we define precision $P_u = \frac{|TP_u|}{|F_u|}$, where TP_u is the set of correctly identified timestamps of $T(u)$ when u was active (a set of interactions, which are correctly attributed to the activity of u), and F_u is the set of all discovered active timestamps of $T(u)$. Similarly, we define the recall for vertex u as $R_u = \frac{|TP_u|}{|A_u|}$, where A_u is the set of true activity timestamps in $T(u)$. We calculate the average precision and recall: $P = \frac{1}{|V|} \sum_{u \in V} P_u$ and $R = \frac{1}{|V|} \sum_{u \in V} R_u$; and report the F-measure $F = \frac{2 \cdot P \cdot R}{P + R}$.

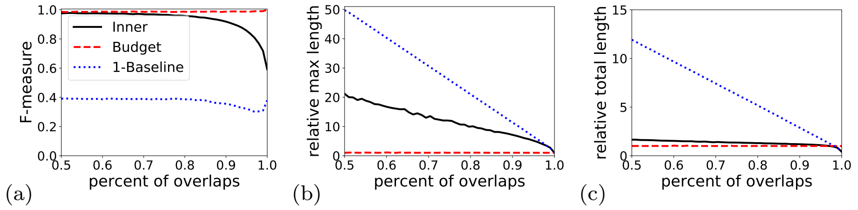

Fig. 5 Output of `Inner`, `Budget`, and `1-Baseline` for different overlaps p in the ground truth activity intervals. (a) F-measure of correctly identified active time-stamped vertices; (b) M: maximum activity interval length divided by true maximum activity interval length; (c) L: total activity interval length divided by true total activity interval length.

In addition to F-measure, we calculate the relative total length L and the relative maximum length M. Here, L is the total length of the discovered intervals divided by the ground-truth total length of the activity intervals. Similarly, M is the maximum length of the discovered intervals divided by the true maximum length of activity intervals.

We also test the sensitivity of the algorithms with respect to initialization. This is more interesting for the setting of $k > 1$ intervals, where we need to guess inner- or gap points. Denote by initialization I_0 the set of gap/inner points, taken from the ground truth, and $A(I_0)$ the solution obtained by our algorithm when initialized with I_0. We introduce a distortion parameter $\delta \in [0, 1]$, which captures the fraction of initialization points from I_0 that are substituted by randomly selected points. Denote the distorted initialization by I_δ and the corresponding solution obtained by our algorithm by $A(I_\delta)$. We experiment with different values of δ and report the normalized Hamming distance H between the solutions $A(I_0)$ and $A(I_\delta)$. The value $H(A(I_0), A(I_\delta))$ is equal to the fraction of timestamps that are classified in the same way (as active or inactive) in both solutions.

Please note that the y-scale is different across the figures reporting the same metrics. This is done to ensure the plots readability. Each figure corresponds to a different experiment and the algorithms performance varies at a different scale.

Case study. For the case study we use a *Twitter* dataset, which records activity of Twitter users in Helsinki between the period of December 2008 to May 2014. We consider only tweets with more than one hashtag (666 487 tweets) and build the co-occurrence network of these hashtags: vertices correspond to hashtags and time-stamped edges correspond to a tweet that mentions both hashtags. The temporal network contains 304 573 vertices and 3 292 699 edges.

8.2 Results for the single interval setting

We test both algorithms on dataset *Synthetic* with varying overlap parameter p. The results are shown in Figure 5. Note that since in *Synthetic* all intervals

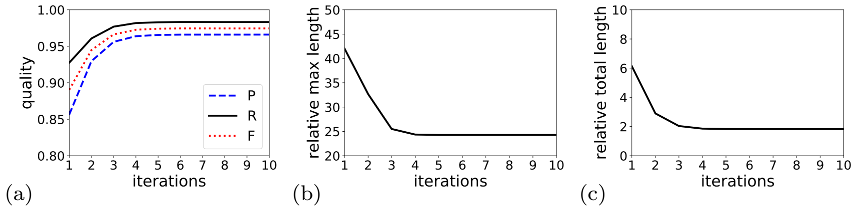

Fig. 6 Convergence of `Maximal` algorithm. Overlap p is set to 0.5. (a) Precision, Recall, and F-measure; (b) M: relative length of the maximum interval; (c) L: relative total length.

have the same length, if during the binary search the correct value of budget is found, then all vertices receive the correct budget.

Figure 5(a) demonstrates that for algorithm `Inner` the F-measure is typically high for all values of the overlap parameter, but drops, when p increases. Compared to the optimal cost, algorithm `Inner` finds good solutions with respect to the total length (shown in Figure 5(c)), but not with respect to the maximum interval length (shown in Figure 5(b)), as it is not designed for this measure. On the other hand, `Budget` performs very well with respect to all measures, for all values of the overlap parameter p. `1-Baseline` performs poorly in terms of all three metrics. It is naive and tends to allocate incorrectly large intervals for the most active nodes on the first steps and to ignore nodes with a lower activity rate. The F-measure value of the baseline algorithm in Figure 5(a) slightly increases after a monotonic decrease when $p > 0.9$. This happens due the way we calculate precision and recall (please see evaluation metrics defined above) and how the baseline operates. In the case of a very large overlap, such as $p > 0.9$, every ground truth activity interval spans almost the entire timeline. The baseline tends to select dense activity intervals, but at this degree of overlap the entire timeline is dense. In this case, the assigned activity intervals are likely to span a very large portion or the entire timeline. This results in a higher true positive count TP_u: even if a particular interaction of u was not planted as a part of activity of a node u, since it falls into its ground truth activity interval, it must be counted as correct. As the numbers $|A_u|$ and $|F_u|$ are not affected, a higher TP_u results in a higher F-measure.

In Figure 6 we see the behavior of algorithm `Inner` as a function of the number of iterations. After a couple of iterations the value and quality (F-measure, precision and recall) of the solution improve dramatically. The method converges in less than 10 iterations.

8.3 Results for the k-interval setting, with $k > 1$

We first report the quality of the solution obtained by algorithms `k-Inner` and `k-Budget` as a function of iterations. Results are presented on Figures 7 and

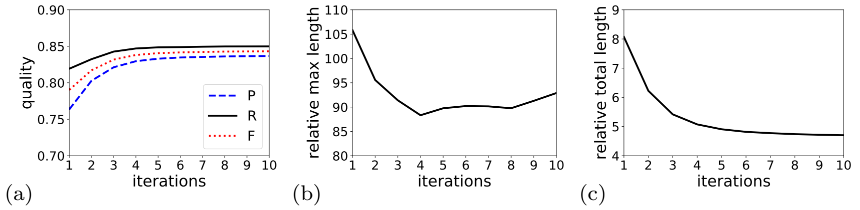

Fig. 7 Convergence of `k-Inner` algorithm from initial active points, set to be mean of the detected k clusters. (a) Precision, Recall, and F-measure; (b) M: relative length of the maximum interval; (c) L: relative total length.

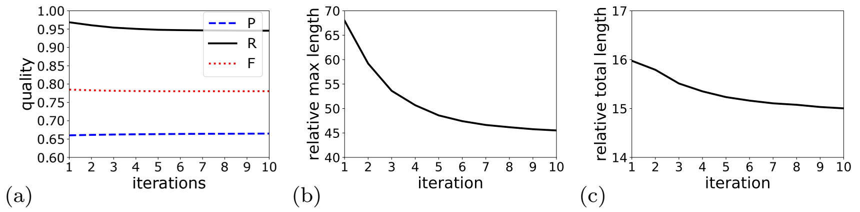

Fig. 8 Convergence of `k-Budget` Algorithm from initial inactive points, set to be mean of the largest gaps. (a) Precision, Recall, and F-measure; (b) M: relative length of the maximum interval; (c) L: relative total length.

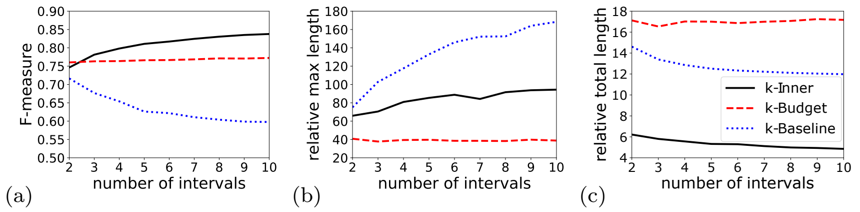

Fig. 9 Effect of the number of intervals on output of `k-Inner`, `k-Budget`, and `k-Baseline`. Initialization is based on clustering. (a) F-measure; (b) M: relative length of the maximum interval; (c) L: relative total length.

8, respectively. Both algorithms consistently improve their objectives during iterations, and both algorithms achieve high F-measure values.

Next, we test how the number of intervals k affects the quality of the solution. Figure 9 shows that `k-Budget` produces stable results with respect to all measures. Results of `k-Inner` are stable in terms of its cost function (total length, Figure 9(c)), while the relative maximum length (Figure 9(b)) grows with k. Longer intervals allow `k-Inner` to compensate for possible errors in the input inner points and achieve larger F-measure values (Figure 9(a)). `k-Baseline` has extremely low F-measure values and high relative maximum length. However, by design `k-Baseline` avoids including large gaps in the activity intervals. This allows achieving lower relative total length than `k-Budget`.

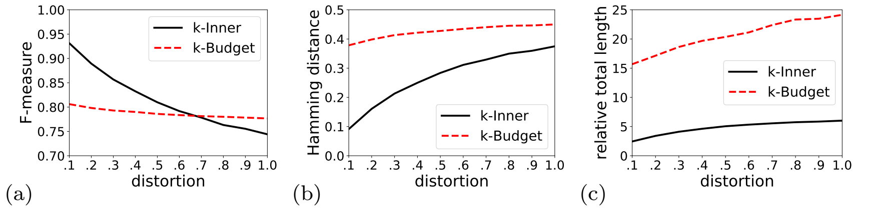

Fig. 10 Sensitivity to initialization. The x-axis shows the percent of initialization points, which are selected randomly: distortion $\delta = 1$ means that all initialization points were chosen randomly. (a) F-measure; (b) normalized Hamming distance $H(A(I_0), A(I_\delta))$ with solution for $\delta = 0$; (c) L: relative total length.

On Figure 10 we evaluate the sensitivity to initialization. We experiment with different initialization distortion parameters δ and report the F-measure, the normalized Hamming distance $H(A(I_0), A(I_\delta))$, and the relative total length of the solution. `k-Budget` finds solutions with similar H-distance and F-measure for all distortions, while `k-Inner` is more sensitive to the initialization. However, even with distortion $\delta = 1.0$ (completely random initialization) the resulting F-measure is quite high. In addition, the cost of the solution (Figure 10(c)) stays approximately constant.

Scalability. Both methods `Budget` and `Inner` (and their extensions `k-Budget` and `k-Inner`) use linear-time algorithms in their inner loops and converge in a small number of iterations. This makes our methods scalable. Indicatively, we are able to run `Maximal` on a power-law network with $\gamma = 2$ of 1 million vertices and 1 billion interactions in 15 minutes, despite using using a non-optimized Python implementation. More running time results are shown in Figure 11, where we run `k-Inner`, `k-Budget`, and `k-Baseline` with $k = 10$ on the first T interactions of the same synthetic temporal graph. The figure shows that `k-Budget` and `k-Baseline` are not as scalable `k-Inner`. Both `k-Budget` and `k-Inner` use linear time core subroutines. `k-Inner` performs iterative search for inner points of activity intervals. `k-Budget` does the similar search for inner points of *inactive* intervals, but has to run additional logarithmic binary search for the budgets. Furthermore, we empirically observe that `k-Budget` needs a higher number of iterations of inactive points updating. Typically inactive intervals are much longer than active intervals. Thus, specifying one inactive point per interval does not contain much informations for a faster convergence.

Case study. Next we present our results on the *Twitter* dataset. For this case study we used algorithm `Inner` since it is faster then `Budget` and scales better for real-world data. In Figure 12 we show a subset of hashtags from tweets posted in November 2013. We also depict the activity intervals for those hashtags, as discovered by algorithm `Inner`. Note that for not cluttering the image, we depict only a subset of all relevant hashtags. In particular, we pick 3 seed hashtags: `#slush13`, `#mtvema` and `#nokiaegm` and the set of hashtags that co-occur with the seeds. Each of the seeds corresponds to a known event: `#slush13` corresponds to Slush'13 — a startup and tech event organized in

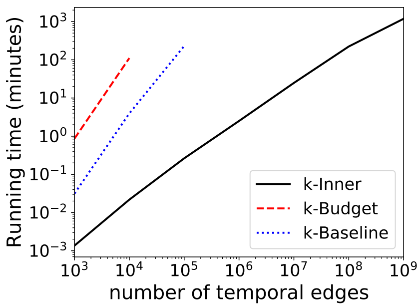

Fig. 11 Running time in minutes for `k-Inner`, `k-Budget`, and `k-Baseline` for $k = 10$.

Helsinki in November 13-14, 2013. `#mtvema` is dedicated to MTV Europe Music Awards, held on 10 November, 2013. `#nokiaegm` is Extraordinary General Meeting (EGM) of Nokia Corporation, held in Helsinki in November 19, 2013.

For each hashtag we depict its activity intervals in blue. All hashtag's mentions are shown as small circles on the timeline. A circle is colored blue if it falls into the hashtag's discovered activity interval and orange otherwise. We draw arched edges for interactions (co-occurrences) of two hashtags only if at the moment of interaction one hashtag is active and another one is not.

Figure 12 shows that the tag `#slush13` becomes active exactly at the starting date of the event. During its activity this tag covers many technical tags, e.g., `#zenrobotics` (Helsinki-based automation company), `#younited` (personal cloud service by local company) and `#walkbase` (local software company). Then on 19 November, the tag `#nokiaegm` becomes active: this event is very narrow and covers mentions of Microsoft executive Stephen Elop. Another large event is occurring around 10 November with active tags `#emazing`, `#ema2013` and `#mtvema`. They cover `#bestpop`, `#bestvideo` and other related tags.

Many events have recurrent nature. For example, Slush is an annual event. In Figure 13 we show a subset of hashtags from tweets posted from January 2011 till December 2013. We run `k-Inner` with $k = 3$ and depict the activity intervals for some hashtags co-occuring with `#slush`.

Figure 13 shows that, although each year has its own tag for Slush (`#slush11`, `#slush12`, `#slush13` and variants), tag `#slush` becomes active every November, when the event takes place. As before, it covers many company names and tech-related hashtags (e.g., `#supercell`, `#sailfish`, `#jolla`, `#aller`). On the other hand, hashtags that are always active (such as startup-related hashtags `#startupsauna` and `#aaltoes`) have large activity intervals, which span the whole timeline.

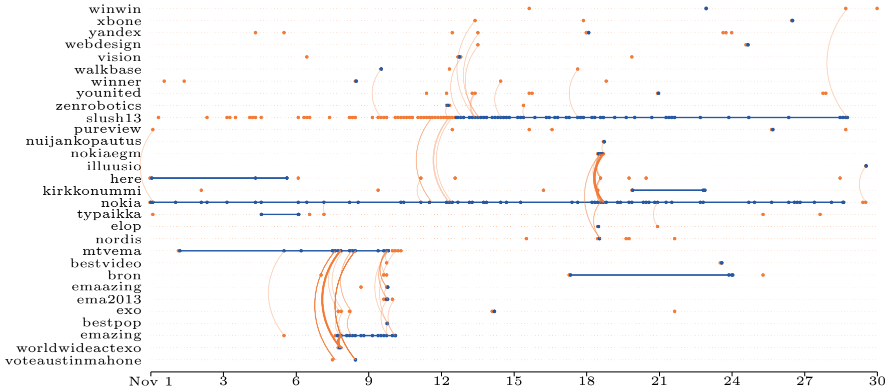

Fig. 12 Part of the output of `Inner` algorithm on Twitter dataset for November'13. Tags, co-occurring with hashtags **`#slush13`**, **`#mtvema`** and **`#nokiaegm`**. Activity intervals and active moments of interactions (hashtags' co-occurrences) are colored blue, inactive moments of interactions are colored orange. Only edges between active and inactive *listed* hashtags are shown.

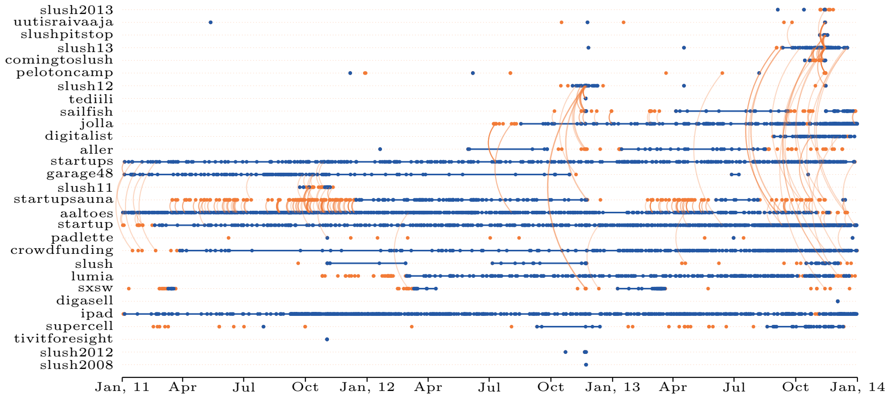

Fig. 13 Part of the output of `k-Inner` algorithm on Twitter dataset years 2011–2013 with $k = 3$. Tags, co-occurring with hashtag **`#slush`**. Activity intervals and active moments of interactions (hashtags' co-occurrences) are colored blue, inactive moments of interactions are colored orange. Only edges between active and inactive *listed* hashtags are shown.

9 Conclusions

In this paper we introduced and studied a new problem, which we called *network untangling* problem, and which provides a *summarization* of a *temporal network*. Given a set of temporal interactions, our goal is to discover *activity time intervals* for the network entities, so as to explain the observed interactions. We considered two settings: MinTimeline$_+$, where we aim to minimize the total sum of interval lengths, and MinTimeline$_\infty$, where we aim to minimize the maximum interval length. We showed that the former problem is

NP-hard and we developed a practical iterative algorithm where one iteration requires only a linear time, while the latter problem is solvable in polynomial time. We also considered a model with k activity intervals, and showed that both problems are inapproximable. We then proposed iterative algorithms `k-Inner` and `k-Budget` for solving k-MinTimeline$_+$ and k-MinTimeline$_\infty$, respectively. Both algorithms are practical: an iteration in `k-Inner` requires linear time while an iteration in `k-Budget` requires $\mathcal{O}(m \log m)$ time.

The proposed approach to temporal network summarization builds a compact summary of nodes activity over time. The resulting summary can be used alone for visualization to show key entities and their activity intervals. The summary can be also used together with the original temporal network to produce a sparsified temporal network, selecting only interactions covered by dense or otherwise interesting activity intervals. Then any further temporal network analysis can be conducted on such "backbone" temporal network, including evolutionary patterns mining, temporal community detection, finding node importance, or information flow. Of course, such intervals summaries do not preserve the whole information about interactions and might be not suitable for fine-grained network analysis.

One limitation of our approach is that we require all temporal edges (interactions) to be covered. This requirement can be too strict, so it is interesting to develop methods that allow more flexibility and give a partial coverage of interactions. Another limitation is that all our methods operate in an off-line fashion. It will be very interesting to develop on-line methods to incrementally update the discovered activity timeline. One more natural extension of the problem is to consider non-binary activity levels or other type of constraints.

The considered problem formulations require the number of intervals k being given as an input parameter. Thus, we assume that the user has some prior knowledge about the data or can find a suitable k value through an experimentation with different values. All our formulations and algorithms can be extended to the case when different k values are specified for different nodes. Such a setting is highly practical, but it is unrealistic to expect the user to specify such a large set of parameters. A practical extension, where an algorithm automatically decides on the number of k activity intervals assigned for each node, is non-trivial and we leave it as a future work.

In our work we assume that all edges are undirected. An interesting variant of the considered problems is where a *portion* of the edges is directed. Here, we could modify the problem by requiring that the out-node must be active for each directed edge. Such a setup leads to additional constraints. `Budget` (and `k-Budget`) can be used directly with these constraints: the constraints translate into preset truth assignments for some of the variables in 2-SAT, resulting clauses of the form $(1 \vee x)$ can be removed and clauses $(0 \vee x)$ can be substituted by $(x \vee x)$. Then `Budget` can be used unchanged. Algorithms for MinTimeline$_+$ and k-MinTimeline$_+$ do require adjustments. For MinTimeline$_+$ it is rather straightforward: if there is a known active point, it must be used as input inner point for Coalesce; if there are more than one active points for a node u, then all interactions of this node between the active points are automatically covered by the active interval u, since there is only one interval per node allowed. These interactions can be excluded from consideration and the cost function of the IP must be adjusted to consider the earliest and the latest known active points for node u. Similar adjustment can be done to solve for k-MinTimeline$_+$.

Declarations

Funding. This work was supported by three Academy of Finland projects (286211, 313927, 317085), the EC H2020 RIA project "SoBigData++" (871042), and the Wallenberg AI, AutonomousSystems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.
Conflicts of interest/Competing interests. All authors declare that they have no conflicting/competing interests.
Availability of data and material. The scripts used to generate the *Synthetic* and *Synthetic*$_k$ datasets are available at `https://github.com/polinapolina/the-network-untangling-problem`. The *Twitter* dataset used for the case study is not publicly available.
Code availability. The implementation of all algorithms and sample scripts used for the experimental evaluation is available at `https://github.com/polinapolina/the-network-untangling-problem`.

References

Aspvall B, Plass MF, Tarjan RE (1979) A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3):121–123

Becker H, Naaman M, Gravano L (2011) Beyond trending topics: Real-world event identification on twitter. vol 11

Bellman R (1961) On the approximation of curves by line segments using dynamic programming. Communications of the ACM 4(6):284

Berlingerio M, Bonchi F, Bringmann B, Gionis A (2009) Mining graph evolution rules. In: joint European conference on machine learning and knowledge discovery in databases, Springer, pp 115–130

Bogdanov P, Mongiovì M, Singh AK (2011) Mining heavy subgraphs in time-evolving networks. In: 2011 IEEE 11th International Conference on Data Mining, IEEE, pp 81–90

Cataldi M, Di Caro L, Schifanella C (2010) Emerging topic detection on twitter based on temporal and social terms evaluation. In: Proceedings of the tenth international workshop on multimedia data mining, pp 1–10

Chvatal V (1979) A greedy heuristic for the set-covering problem. Mathematics of operations research 4(3):233–235

Dakiche N, Tayeb FBS, Slimani Y, Benatchba K (2019) Tracking community evolution in social networks: A survey. Information Processing & Management 56(3):1084–1102

Fung GPC, Yu JX, Yu PS, Lu H (2005) Parameter free bursty events detection in text streams. In: Proceedings of the 31st international conference on Very large data bases, pp 181–192

Gauvin L, Panisson A, Cattuto C (2014) Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. PloS one 9(1):e86028

Guha S, Koudas N, Shim K (2006) Approximation and streaming algorithms for histogram construction problems. ACM Transactions on Database Systems (TODS) 31(1):396–438

Hartmanis J (1982) Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). Siam Review 24(1):90

Hartmann T, Kappes A, Wagner D (2016) Clustering evolving networks. In: Algorithm Engineering, Springer, pp 280–329

He D, Parker DS (2010) Topic dynamics: an alternative model of bursts in streams of topics. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 443–452

He J, Chen D (2015) A fast algorithm for community detection in temporal network. Physica A: Statistical Mechanics and its Applications 429:87–94

Holme P, Saramäki J (2012) Temporal networks. Physics reports 519(3):97–125

Hopcroft JE, Ullman JD (1983) Data structures and algorithms, vol 175. Addison-Wesley Boston, MA, USA

Hulovatyy Y, Chen H, Milenković T (2015) Exploring the structure and function of temporal networks with dynamic graphlets. Bioinformatics 31(12):i171–i180

Ihler A, Hutchins J, Smyth P (2006) Adaptive event detection with time-varying poisson processes. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 207–216

Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations, Springer, pp 85–103

Khot S, Regev O (2008) Vertex cover might be hard to approximate to within 2- ε. Journal of Computer and System Sciences 74(3):335–349

Kleinberg J (2003) Bursty and hierarchical structure in streams. Data Mining and Knowledge Discovery 7(4):373–397

Kovanen L, Karsai M, Kaski K, Kertész J, Saramäki J (2013) Temporal motifs. In: Temporal Networks, Springer, pp 119–133

Lahiri M, Berger-Wolf TY (2008) Mining periodic behavior in dynamic social networks. In: 2008 Eighth IEEE International Conference on Data Mining, IEEE, pp 373–382

Lappas T, Arai B, Platakis M, Kotsakos D, Gunopulos D (2009) On burstiness-aware search for document sequences. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 477–486

Leskovec J, Backstrom L, Kleinberg J (2009) Meme-tracking and the dynamics of the news cycle. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 497–506

Liu Y, Safavi T, Dighe A, Koutra D (2018) Graph summarization methods and applications: A survey. ACM Computing Surveys (CSUR) 51(3):1–34
Masuda N, Holme P (2019) Detecting sequences of system states in temporal networks. Scientific reports 9(1):1–11
Mathioudakis M, Koudas N (2010) Twittermonitor: trend detection over the twitter stream. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp 1155–1158
McGregor A (2014) Graph stream algorithms: a survey. ACM SIGMOD Record 43(1):9–20
Meladianos P, Nikolentzos G, Rousseau F, Stavrakas Y, Vazirgiannis M (2015) Degeneracy-based real-time sub-event detection in twitter stream. The International AAAI Conference on Web and Social Media (ICWSM) 15:248–257
Michail O (2016) An introduction to temporal graphs: An algorithmic perspective. Internet Mathematics 12(4):239–280
Newman ME (2003) The structure and function of complex networks. SIAM review 45(2):167–256
Paranjape A, Benson AR, Leskovec J (2017a) Motifs in temporal networks. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp 601–610
Paranjape A, Benson AR, Leskovec J (2017b) Motifs in temporal networks. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp 601–610
Pietilänen AK, Diot C (2012) Dissemination in opportunistic social networks: the role of temporal communities. In: Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing, pp 165–174
Rossetti G, Cazabet R (2018) Community discovery in dynamic networks: a survey. ACM Computing Surveys (CSUR) 51(2):1–37
Rozenshtein P, Gionis A (2019) Mining temporal networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 3225–3226
Rozenshtein P, Tatti N, Gionis A (2017) The network-untangling problem: From interactions to activity timelines. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 701–716
Shah N, Koutra D, Zou T, Gallagher B, Faloutsos C (2015) Timecrunch: Interpretable dynamic graph summarization. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 1055–1064
Shahaf D, Guestrin C, Horvitz E (2012a) Metro maps of science. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1122–1130
Shahaf D, Guestrin C, Horvitz E (2012b) Trains of thought: Generating information maps. In: Proceedings of the 21st international conference on World Wide Web, pp 899–908

Shahaf D, Yang J, Suen C, Jacobs J, Wang H, Leskovec J (2013) Information cartography: creating zoomable, large-scale maps of information. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1097–1105
Tatti N (2019) Strongly polynomial efficient approximation scheme for segmentation. Information Processing Letters 142:1–8
Vlachos M, Meek C, Vagena Z, Gunopulos D (2004) Identifying similarities, periodicities and bursts for online search queries. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp 131–142
Wackersreuther B, Wackersreuther P, Oswald A, Böhm C, Borgwardt KM (2010) Frequent subgraph discovery in dynamic networks. In: Proceedings of the Eighth Workshop on Mining and Learning with Graphs, pp 155–162
Weng J, Lee BS (2011) Event detection in twitter. In: The International AAAI Conference on Web and Social Media (ICWSM)
Yang J, Leskovec J (2011) Patterns of temporal variation in online media. In: Proceedings of the fourth ACM international conference on Web search and data mining, pp 177–186
Zhu Y, Shasha D (2003) Efficient elastic burst detection in data streams. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp 336–345

A Proofs regarding computational complexity

Proof (of Proposition 2) To prove the result we provide a reduction from VertexCover. Assume that we are given a graph H with n vertices and an integer k. We construct a temporal network G as follows: We place H at time stamp 0. We then add k fully-connected graphs $C_1, \ldots, C_k$ with n vertices at time stamps $1, \ldots, k$. Figure 14 shows an example.

We claim that G has k-interval cover with *zero* cost if and only if H can be covered with k vertices. This shows that solving whether there is a zero-cost solution for either k-MinTimeline$_+$ or k-MinTimeline$_\infty$ is **NP**-complete, which also automatically implies that these problems are inapproximable.

To prove the claim, first assume that H can be covered with k vertices, say $w_1, \ldots, w_k$. Construct a k-activity timeline by first covering $w_1, \ldots, w_k$ at time stamp 0 with zero-length intervals. Similarly, cover every vertex v at every time stamp $t = 1, \ldots, k$ with a zero-length intervals, except if $v = w_t$. Figure 14 shows an example. By definition H is covered, and $n - 1$ vertices in each C_i are also covered. Consequently, the timeline covers G. Since each vertex uses exactly k intervals, we have proven the first direction.

To prove the other direction, assume that there is a zero-solution for G. Since each C_i is fully-connected, we must have at least $n - 1$ vertices covered in each C_i. This means that we have at most k spare intervals to cover H. The vertices that these intervals cover form a k-vertex cover, proving the claim. □

Proof (of Proposition 3) We will prove the hardness by reducing VertexCover to MinTimeline$_+$. Assume that we are given a (static) network $H = (W, A)$ with n vertices $W = \{w_1, \ldots, w_n\}$ and a budget ℓ. In the VertexCover problem we are asked to decide whether there exists a subset $U \subseteq W$ of at most ℓ vertices ($|U| \leq \ell$) covering all edges in A.

We map an instance of VertexCover to an instance of MinTimeline$_+$ by creating a temporal network $G = (V, E)$, as follows. The vertices V consist of $2n$ vertices: for each

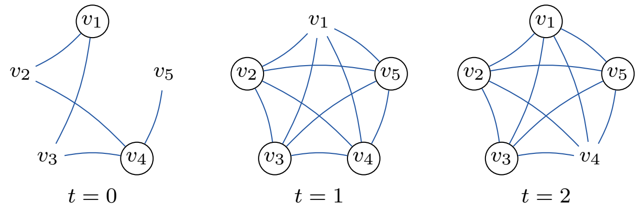

Fig. 14 Example of a temporal network used in proof of Proposition 2. Circled nodes are active at the corresponding time points

$w_i \in W$, we add vertices v_i and u_i. The edges are as follows: For each edge $(w_i, w_j) \in A$, we add a temporal edge $(v_i, v_j, 0)$ to E. For each vertex $w_i \in W$, we add two temporal edges $(v_i, u_i, 1)$ and $(v_i, u_i, n+2)$ to E. Figure 15 shows an example.

Let $\mathcal{T}^*$ be an optimal timeline covering G. We claim that $S(\mathcal{T}^*) \leq \ell$ if and only if there is a vertex cover of H with ℓ vertices.

To prove the *if* direction, consider a vertex cover of H, say U, with ℓ vertices. Consider the following coverage: cover each u_i at $n+2$, and each v_i at 1. For each $w_i \in U$, cover v_i at 0. Figure 15 shows an example. Now every vertex u_i has a 0-length activity interval $[n+2, n+2]$. Vertices v_i, which correspond to $w_i \notin U$, also have 0-length activity intervals $[1, 1]$. Only vertices v_i, which correspond to $w_i \in U$, have 1-length activity intervals $[0, 1]$. All vertices in V have activity intervals of total length ℓ. Every interaction in G belongs to some activity interval: by construction, interactions at $t = n+2$ are spanned by activity intervals of $\{u_i\}$, interactions at $t = 1$ are spanned by activity intervals of $\{v_i\}$, and interactions at $t = 0$ are spanned by activity intervals ℓ of vertices from $\{v_i\}$. The resulting intervals are indeed forming a timeline with a total span of ℓ.

To prove the other direction, first note that if we cover each v_i by an interval $[0, 1]$ and each u_i by an interval $[n+2, n+2]$, then this yields a timeline $\mathcal{T}$ covering G. The total span of intervals in $\mathcal{T}$ is n. Thus, $S(\mathcal{T}^*) \leq S(\mathcal{T}) = n$. This guarantees that if $0 \in I_{v_i}$, then $n+2 \notin I_{v_i}$, so $n+2 \in I_{u_i}$. Otherwise $\mathcal{T}^*$ contains an $(n+2)$-length interval, this contradicts to $S(\mathcal{T}^*) \leq n$. By the same argument, $1 \notin I_{u_i}$ and so $1 \in I_{v_i}$. In summary, if $0 \in I_{v_i}$, then $\sigma(I_{v_i}) = 1$. This implies that if $S(\mathcal{T}^*) \leq \ell$, then we have at most ℓ active vertices at 0. Let U be the set of those vertices. Since $\mathcal{T}^*$ is a timeline covering G, then U is a vertex cover for H. □

B Proofs regarding `Maximal`

Proof (of Proposition 4) We first show that a maximal dual solution is a feasible timeline. Let $e_i = (u, v, t)$ be a temporal edge. If $p(i; v) \cap X_v = \emptyset$ and $p(i; u) \cap X_u = \emptyset$, then we can increase the value of α_i without violating the dual constraints, so the solution is not maximal. Thus $t \in I_v \cup I_u$, making $\mathcal{T}$ a feasible timeline.

Next we show that the resulting solution $\mathcal{T}$ is a 2-approximation to Coalesce. Write $x_v = \min(X_v)$ and $y_v = \max(X_v)$. Let $\mathcal{T}^*$ be the optimal solution. Then

$$
\begin{aligned}
S(\mathcal{T}) &= \sum_{v \in V} |t(x_v) - m_v| + |t(y_v) - m_v| \\
&= \sum_{v \in V} h(v; x_v) + h(v; y_v) \\
&\leq \sum_{v \in V} \sum_{e \in E(v)} \alpha_e = 2 \sum_{e \in E} \alpha_e \leq 2S(\mathcal{T}^*),
\end{aligned}
$$

where the second equality follows from the definition of X_v, the first inequality follows from the fact that $\alpha_e \geq 0$, and the last inequality follows from primal-dual theory. □

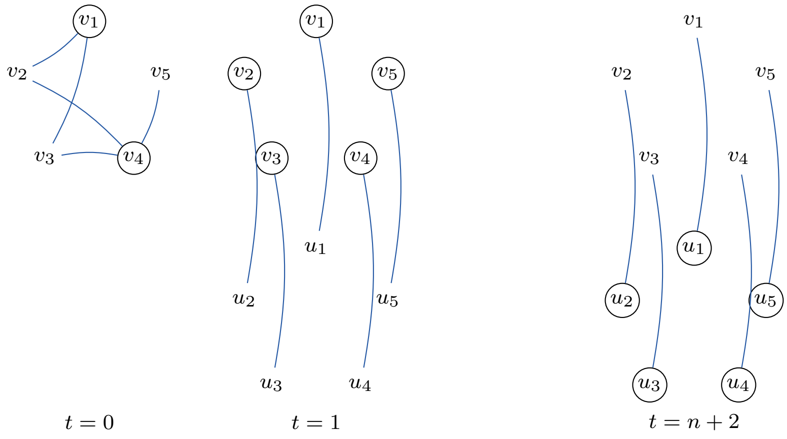

Fig. 15 Example of a temporal network used in proof of Proposition 3. Circled nodes are active at the corresponding time points

Proof (of Lemma 1) We will prove this result by showing that $\alpha_j \leq z(v)$ if and only if all dual constraints related to v are valid. Since the same holds also for u the lemma follows. We consider two cases.

First case: $t(j) < m_v$. In this case we have

$$z(v) = \min \{m_w - t, b[w]\} = \min_{i \leq j} \{t(i) - m_v - h(v; i)\},$$

before increasing α_j. This guarantees that if $\alpha_j \leq z(v)$, then $h(v;i) \leq |t(i) - m_v|$, for every $i \leq j$. Moreover, when $\alpha_j = z(v)$ one of these constraints becomes tight. Since these are the only constraints containing α_j, we have proven the first case.

Second case: $t(j) \geq m_v$. If $i < j$, the sum $h(v;i)$ does not contain α_j, so the corresponding constraint remains valid. If $i \geq j$, then the corresponding constraint is valid if and only if $h(v;j) \leq |t(j) - m_v|$. This is because $\alpha_\ell = 0$ for all $\ell > j$. But $z(v) = t - m_v - a[v]$ corresponds exactly to the amount we can increase α_j so that $h(v;j) = |t - m_v|$. □

C Proofs regarding `k-Maximal`

Proof (of Proposition 5) Let us first prove the feasibility of $\mathcal{T}$, that is, show that every edge is covered. Let $e_i = (u, v, t)$ be an edge, and let ℓ be an index such that $i \in S_{v\ell}$.

At least, one of the four cases of left-maximality must hold. Assume Case (*i*), that is, $rh(v;j) = \eta_{vj}$ for some $j \in lp(i;v)$. Then $t(j) \in X_{v\ell}$ and $x_{v\ell} \leq t(i) \leq m_{v\ell}$, so e_i is covered. Case (*ii*) is similar.

Assume that Cases (*i*) and (*ii*) do not hold. Then Case (*iii*) or Case (*iv*) holds. Assume Case (*iii*). If $x_{v\ell} \leq t(i)$, then e_i is covered. Assume that $t(i) < x_{v\ell}$. Then, by definition, $t(i) \in Y_{v\ell}$. Thus $m_{v(\ell-1)} < t(i) \leq y_{v\ell}$, so e_i is covered. Case (*iv*) is similar.

We have shown that $\mathcal{T}$ is feasible. Next we show that the resulting solution $\mathcal{T}$ is a 2-approximation to k-Coalesce. Let $\mathcal{T}^*$ be the optimal solution. Then

$$\begin{aligned} S(\mathcal{T}) &= \sum_{\ell=1}^{k} \sum_{v \in V} |t(x_{v\ell}) - m_{v\ell}| + |t\big(y_{v(\ell+1)}\big) - m_{v\ell}| \\ &= \sum_{\ell=1}^{k} \sum_{v \in V} rh(v; x_{v\ell}) + lh(v; y_{v\ell}) \\ &\leq \sum_{v \in V} \sum_{e \in E(v)} \alpha_e = 2 \sum_{e \in E} \alpha_e \leq 2S(\mathcal{T}^*)\,, \end{aligned}$$

where the second equality follows from the definition of X_{vi} and Y_{vi}, the first inequality follows from the fact that $\alpha_e \geq 0$ and the intervals in $\mathcal{T}$ do not intersect, and the last inequality follows from primal-dual theory. □

We will prove Proposition 6 in a sequence of lemmas.

Let us write $a_i[v, \ell]$ and $b_i[v, \ell]$ to be the values of these counters at the beginning of the ith iteration. We maintain the following invariants. The first counter, $a_{i+1}[v, \ell]$ matches $lh(v; i)$. The second counter, $b_i[v, \ell]$ is how much we can afford to increase α_i without violating $rh(v; s) \leq \eta_{vs}$, where $s \leq i$. This is formalized in the next lemma.

Lemma 2 *Let v be a vertex and $\ell = 1, \ldots, k+1$ be an integer. Shorten $S = S_{v\ell}$ and write $f(j, i) = \sum_{o \in S, j \leq o \leq i} \alpha_o$.*

Then for any $i \in S$,

$$a_{i+1}[v, \ell] = lh(v; i) \tag{2}$$

and

$$b_{i+1}[v, \ell] = \min_{j \in S, j \leq i} (\eta_{vj} - f(j, i))\,. \tag{3}$$

Proof We prove this claim by induction. Let i be the first index in S. Then $a_i[v, \ell] = 0$ and $a_{i+1}[v, \ell] = \alpha_i$, and $b_i[v, \ell] = \infty$ and $b_{i+1}[v, \ell] = \eta_{vi} - \alpha_{vi}$.

Assume that i is not the first index in S, and let $j \in S$ be the previous index. Since $a_i[v, \ell] = a_{j+1}[v, \ell]$, we have

$$a_{i+1}[v, \ell] = a_i[v, \ell] + \alpha_i = lh(v; j) + \alpha_i = lh(v; i)\,.$$

Also, $b_i[v, \ell] = b_{j+1}[v, \ell]$, and

$$\begin{aligned} b_{i+1}[v, \ell] &= \min(b_i[v, \ell] - \alpha_i, \eta_{vi} - \alpha_i) \\ &= \min\left(\min_{s \in S, s \leq j} (\eta_{vs} - f(s, i))\,, \eta_{vi} - \alpha_i\right) \\ &= \min_{s \in S, s \leq i} (\eta_{vs} - f(s, i))\,, \end{aligned}$$

proving the lemma. □

Our next step is to prove the feasibility of the output of k-`Maximal`. In order to do that, we first bound the counters.

Lemma 3 *For each vertex v, index $\ell = 1, \ldots, k$, and $i \in S_{v\ell}$,*

$$a_{i+1}[v, \ell] \leq \theta_{vi} \tag{4}$$

and

$$b_{i+1}[v, \ell] \geq 0. \tag{5}$$

Proof Since, $\alpha_i \leq \theta_{vi} - a_i[v, \ell]$ we have

$$a_{i+1}[v, \ell] = a_i[v, \ell] + \alpha_i \leq \theta_{vi}.$$

Also since, $\alpha_i \leq \min(b_i[v, \ell], \eta_{vi})$ we have

$$b_{i+1}[v, \ell] = \min(b_i[v, \ell], \eta_{vi}) - \alpha_i \geq 0.$$

This proves the claim. □

To prove the feasibility, we first show that $\alpha_i \geq 0$.

Lemma 4 *$\alpha_i \geq 0$, for all i.*

Proof Let $(u, v, t) = e_i$, and let ℓ such that $i \in S_{v\ell}$. Let z_1, and z_2 be as defined by the algorithm in the ith round.

If i is the first index in $S_{v\ell}$, then $a_i[v, \ell] = 0$ and $b_i[v, \ell] = \infty$. Then $z_1 \geq 0$, and similarly $z_2 \geq 0$. Consequently, $\alpha_i \geq 0$.

Assume that i is not the first index in $S_{v\ell}$, and let j be the previous edge index. Then Eq. 4 implies

$$a_i[v, \ell] = a_{j+1}[v, \ell] \leq \theta_{vj} \leq \theta_{vi}.$$

In addition, Eq. 5 implies $b_i[v, \ell] \geq 0$, so $z_1 \geq 0$. Similarly, $z_2 \geq 0$. Consequently, $\alpha_i \geq 0$. □

Next lemma shows that $\{\alpha_i\}$ satisfies the constraints, making the dual solution feasible.

Lemma 5 *Let $v \in V$, $\ell = 1, \ldots, k+1$, and $i \in S_{v\ell}$. Then $rh(v; i) \leq \eta_{vi}$ and $lh(v; i) \leq \theta_{vi}$.*

Proof Eq. 2 and Eq. 4 give us

$$lh(v; i) = a_{i+1}[v, \ell] \leq \theta_{vi}.$$

Moreover, $lh(v; i)$ remains constant in the later rounds.

Let j be the last index in $S_{v\ell}$. Then Eq. 3 and Eq. 5 state that

$$0 \leq b_{j+1}[v, \ell] \leq \eta_{vi} - rh(v; i)\,.$$

Moreover, the sum $rh(v; i)$ remains constant in the later rounds. Thus, $\{\alpha_i\}$ is a feasible dual solution. □

Proof (of Proposition 6) Let $e_i = (u, v, t)$ be an edge, and let ℓ and r be the indices such that $i \in S_{v\ell}$ and $i \in S_{ur}$. We need to show that e_i is left-maximal.

If $b_{i+1}[v, \ell] = 0$, then Eq. 3 states that there is $j \in X_{v\ell}$ with $j \leq i$ such that $\eta_{vj} = rh(v; j)$. This is the definition of Case (*i*) of left-maximality. Similarly, $b_{i+1}[u, r] = 0$ leads to Case (*ii*).

Assume $b_{i+1}[v, \ell] > 0$ and $b_{i+1}[u, r] > 0$. This can only happen if

$$\alpha_e < \min(b_i[v, \ell], \eta_{vi}) \quad \text{and} \quad \alpha_e < \min(b_i[u, r], \eta_{ui}).$$

Consequently, $\alpha_i = \theta_{vi} - a_i[v, \ell]$ or $\alpha_i = \theta_{ui} - a_i[u, r]$. If the former, then Eq. 2 states $lh(v; i) = a_{i+1}[v, \ell] = \theta_{vi}$, leading to Case (*iii*). Similarly, the latter case leads to Case (*iv*). Thus, e_i is left-maximal. □