The network-untangling problem: From interactions to activity timelines

Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis

HIIT, Aalto University, Espoo, Finland
firstname.lastname@aalto.fi

Abstract. In this paper we study a problem of determining when entities are active based on their interactions with each other. More formally, 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 that takes place at time t. We view this input as a *temporal network*. We then assume a simple *activity model* in which each entity is *active* during a short time interval. 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, which we refer to as the *network-untangling problem*, can be applied to discover timelines of events from complex interactions among entities.
We provide two formulations for the network-untangling problem: (i) minimizing the total interval length over all entities, and (ii) minimizing the maximum interval length. We show that the sum problem is **NP**-hard, while, surprisingly, the max problem can be solved optimally in linear time, using a mapping to 2-SAT. For the sum problem we provide efficient and effective algorithms based on realistic assumptions. Furthermore, we complement our study with 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

Data increase in volume and complexity. A major challenge that arises in many applications is to process efficiently large amounts of data in order to synthesize the available bits of information into a concise but meaningful picture.

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* [9, 19]. In temporal networks $G = (V, E)$, 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 for summarizing temporal networks. The main idea is to consider that the entities of the network are *active* over presumably short time intervals. Edges (interactions) of the temporal network between two entities can be explained by *at least one* of the two entities being active at the time of the interaction. Our summarization task is to process the available temporal edges (interactions) and infer the latent activity intervals for all entities. In this way, we can infer an *activity timeline* for the whole network. To motivate the summarization task studied in this paper, consider the following application scenario.

Example. Consider a news story unfolding over the period of several months, or years, such as **Brexit**. There is a sequence of intertwined events (e.g., UK referendum, prime minister resigns, appointment of new prime minister, supreme court decision, invoking article 50, etc.) as well as a roster of key characters who participate in the events (e.g., Cameron, Johnson, May, Tusk, etc.). Consider now a stream of Brexit-related tweets, as events unfold, and hashtags mentioned in those tweets (e.g., `#brexit`, `#remain`, `#ukip`, `#indyref2`, etc.). For our purposes, we view the twitter stream as a temporal network: a tweet mentioning two hashtags h_1 and h_2 and posted at time t is seen as a temporal edge (h_1, h_2, t). A typical situation is that a hashtag *bursts* during a time interval that is associated with a *main event*, while it may also appear outside the time interval in a connection with other *secondary events*. For instance, the peak activity for `#remain` may have been during the weeks leading to the referendum, but the same hashtag may also appear later, say, in reference to invoking article 50, by a user who wished that EU had not voted for Brexit. The question that we ask in this paper is whether it is possible to process the temporal network of entity interactions and reconstruct the latent activity intervals for each entity (hashtags, in this example), and thus, infer the complete timeline of the news story.

Motivated by the previous example, and similar application scenarios, we introduce the *network-untangling problem*, where the goal is to reconstruct an activity timeline from a temporal network. Our formulation uses a simple model in which we assume that each network entity is *active* during a time interval. An temporal edge (u, v, t) is *covered* if at least one of u or v are active at time t. The algorithmic objective is to find a set of activity intervals, one 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 and maximum length.

We show that the problem of minimizing the maximum length over all activity intervals can be mapped to 2-SAT, and be solved optimally and in linear time On the other hand, minimizing the total interval length is an **NP**-hard problem. To confront this challenge we offer two iterative algorithms that rely on the fact that certain subproblems can be solved approximately or optimally. In both 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 are capable on finding 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.

2 Preliminaries and problem definition

Our input is a temporal network $G = (V, E)$, where V is a set of vertices and E is a set of time-stamped edges. The edges of the temporal network 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. In our setting 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.

Given a vertex $u \in V$, we will write $E(u)$ to be the set of edges adjacent to vertex u, i.e., $E(u) = \{(u, v, t) \in E\}$. We will also 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 $e \in E$.

Given a vertex $u \in V$ and two real 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 the timeline $\mathcal{T}$ *covers* the network G if for each edge $(u, v, t) \in E$, we have $t \in I_u$ or $t \in I_v$, that is, when each network edge occurs at least one of its endpoints is active.

Note that each temporal network has 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 finding an activity timeline that have 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 $\mathcal{T}$.

Associated with the above compactness measures we define the following two problems that we consider in this paper.

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 $\Delta(\mathcal{T})$.

3 Computational complexity and algorithms

Surprisingly, while MinTimeline is an **NP**-hard problem, MinTimeline$_\infty$ can be solved optimally efficiently. The optimality of MinTimeline$_\infty$ is a result of the algorithm presented in Section 5. In this section we establish the complexity of MinTimeline, and we present two efficient algorithms for MinTimeline and MinTimeline$_\infty$.

Proposition 1. *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 $S(\mathcal{T}^*) \leq \ell$.*

Proof. 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 consists of $2n$ vertices: for each $w_i \in W$, we add vertex 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 $(v_i, u_i, 2n + 1)$ to E.

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 $2n + 1$, and each v_i at 1. For each $w_i \in U$, cover v_i at 0. 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 $[2n + 1, 2n + 1]$, then this yields a timeline $\mathcal{T}^*$ covering G. The total span intervals $\mathcal{T}^*$ is n. Thus, $S(\mathcal{T}^*) \leq n$. This guarantees that if $0 \in I_{v_i}$, then $2n+1 \notin I_{v_i}$, so $2n+1 \in I_{u_i}$. This implies that $1 \notin I_{u_i}$ 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 vertices covered at 0. Let U be the set of those vertices. Since $\mathcal{T}^*$ is timeline covering G, then U is a vertex cover for H. □

3.1 Iterative method based on inner points

As we saw, MinTimeline is an **NP**-hard problem. The next logical question is whether we can approximate this problem. Unfortunately, there is evidence that such an algorithm would be highly non-trivial: we can show that if we extend our problem definition to hyper-edges—the coverage then means that one vertex needs to be covered per edge—then such a problem is inapproximable. This suggests that an approximation algorithm would have to rely on the fact that we are dealing with edges and not hyper-edges.

Luckily, we can consider meaningful subproblems. Assume that we are given a temporal network $G = (V, E)$ and we also given a set of time point $\{m_v\}_{v \in V}$, i.e., one time point m_v for each vertex $v \in V$, and we are asked whether we can find an optimal activity timeline $\mathcal{T} = \{I_u\}_{u \in V}$ so that the interval I_v of vertex v contains the corresponding time point m_v, i.e., $m_v \in I_v$, for each $v \in V$. Note that these inner points can be located *anywhere* within the interval (not just, say, in the center of the interval). 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 $\textsc{MinTimeline}_m$.

Problem 3. ($\textsc{MinTimeline}_m$) Given a temporal network $G = (V, E)$ and a set of inner time points $\{m_v\}_{v \in V}$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ 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 $\textsc{MinTimeline}_m$ problem can be solved approximately, in *linear time*, within a factor of 2 of the optimal solution. The 2-approximation algorithm is presented in Section 4.

Being able to solve $\textsc{MinTimeline}_m$, motivates the following algorithm for $\textsc{MinTimeline}$, which uses $\textsc{MinTimeline}_m$ as a subroutine: initialize $m_v = (\min T(v) + \max T(v))/2$ to be an inner time point for vertex v; recall that $T(v)$ are the time stamps of the edges containing v. We then use our approximation algorithm for $\textsc{MinTimeline}_m$ 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`.

3.2 Iterative method based on budgets

Our algorithm for $\textsc{MinTimeline}_\infty$ also 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 $\textsc{MinTimeline}_b$.

Problem 4. ($\textsc{MinTimeline}_b$) 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 $\textsc{MinTimeline}_b$ problem can be solved *optimally* in *linear time.* The algorithm is presented in Section 5. Note that this result is compatible with the **NP**-hardness of $\textsc{MinTimeline}$, since here we know the budgets for *individual* intervals, and thus, there are an exponential number of ways that we can distribute the total budget among the individual intervals.

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 binary steps, 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 first define $W(i) = \{t_j - t_i \mid L < t_j - t_i < U\}$. 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 $W(i)$, ignore any empty $W(i)$. Finally, we test the weighted 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 is 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 a $\mathcal{O}(m \log m)$ running time. However, in our experimental evaluation, we use a straightforward binary search by testing $(U + L)/2$ as a budget.

4 Approximation algorithm for MinTimeline$_m$

In this section we design a 2-approximation linear-time algorithm for the MinTimeline$_m$ problem. As defined in Problem 3, our input is a temporal network $G = (V, E)$ and a set of interior time points $\{m_v\}_{v \in V}$. As before, $T(v)$ denotes the set of time stamps of the edges containing vertex v.

Consider a vertex v and the corresponding interior point m_v. For a time point t we define the *peripheral time stamps* $p(t; v)$ to be the time stamps that are on the other side of t than m_v,

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

Our next step is to express MinTimeline$_m$ as an integer linear program. To do that we will define a variable x_{vt} for each vertex $v \in V$ and time stamp $t \in T(v)$. Instead of going for the obvious construction, where $x_{vt} = 1$ indicates that v is active at t, we will do a different formulation: in our program $x_{vt} = 1$ indicates that t is either the *beginning* or the *end* of the active region of v. It follows that the integer program

$$\begin{aligned} \min \quad & \sum_{v,t} |t - m_v| x_{vt}, \\ \text{such that} \quad & \sum_{s \in p(v;t)} x_{vs} + \sum_{s \in p(u;t)} x_{us} \geq 1, \ \text{ for all } (u, v, t) \in E \end{aligned}$$

solves MinTimeline$_m$. Naturally, here we also require that $x_{vt} \in \{0, 1\}$. 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.

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

$$h(v,t,s) = \sum \left\{ \alpha_e \mid e \in E(v)\,,\ t(e) \text{ is between } s \text{ and } t \right\},$$

where, recall that, $E(v)$ denotes the edges adjacent to v and $t(e)$ denotes the time stamp of edge $e \in E$. The dual can now be formulated as

$$\max \sum_{e \in E} \alpha_e, \quad \text{such that} \quad h(v, t, m_v) \leq |t - m_v|, \text{ for all } v \in V,\ t \in T(v)\,,$$

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

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.

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

Proposition 2. *Consider a maximal solution α_e to the dual program. Define a set of intervals $\mathcal{T} = \{I_v\}$ by $I_v = [\min X_v, \max X_v]$, where*

$$X_v = \{m_v\} \cup \{t \in T(v) \mid h(v, t, m_v) = |t - m_v|\}\,.$$

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

Proof. We first show that a maximal dual solution is a feasible timeline. Let $e = (u, v, t)$ be a temporal edge. If $p(t; v) \cap X_v = \emptyset$ and $p(t; u) \cap X_u = \emptyset$, then we can increase the value of α_e without violating the 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 MinTimeline$_m$. Write $x_v = \min\{X_v\}$ and $y_v = \max\{X_v\}$. Let $\mathcal{T}^*$ be the optimal solution. Then

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

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. This proves the claim. □

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. We will now introduce a linear-time algorithm that computes a maximal dual solution. The algorithm visits each edge $e = (u, v, t)$ 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.

Algorithm 1: `Maximal`, yields 2-approximation to $\textsc{MinTimeline}_m$.

```
$b[v] \leftarrow \infty$ 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. (2)}
    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 $a[u] \leftarrow a[u] + \alpha_e$ ;
```

Let us enumerate the edges chronologically by writing e_i for the i-th edge, and let us write α_i to mean α_{e_i}. We will also write t_i for the time stamp of e_i. Finally, let us define k_v to be the smallest index of an edge (u, v, t) with $t \geq m_v$, and o_v to be the largest index of an edge (u, v, t) with $t \leq m_v$.[1]

For simplicity, we rewrite the dual constrains using indices instead of time stamps. Given two indices $i \leq j$, we slightly overload the notation and we write

$$h(v, i, j) = \sum \{\alpha_\ell \mid e_\ell \in E(v)\,,\ \ell \text{ is between } i \text{ and } j\}\,.$$

The dual constraints can be written as

$$h(v, i, o_v) \leq |t_i - m_v|, \text{ if } i < k_v, \quad \text{and} \quad h(v, i, k_v) \leq |t_i - m_v|, \text{ if } i \geq k_v. \quad (1)$$

Each dual constraint is included in these constraints. Eq. (1) may also contain some additional constraints but they are redundant, so the dual constraints hold if and only if constraints in Eq. (1) hold.

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 j-th round they are equal to

$$a[v] = h(v, k_v, j) \quad \text{and} \quad b[v] = \min_{\ell < j}\{t_\ell - m_v - h(v, \ell, j - 1)\}.$$

The following lemma tells us how to update α_j using $a[v]$ and $b[v]$.

[1] If there is an edge exactly at m_v, then $k_v = o_v$.

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

$$\min\{z(u), z(v)\}, \quad \textit{where} \quad z(w) = \begin{cases} t - m_w - a[w] & \textit{if } j \geq k_v, \\ \min\{m_w - t, b[w]\} & \textit{if } j < k_v. \end{cases} \tag{2}$$

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

First case: $j < k_v$. In this case we have $z(v) = \min\{m_w - t, b[w]\} = \min_{\ell \leq j}\{t_\ell - m_v - h(v, \ell, o_v)\}$, before increasing α_j. This guarantees that if $\alpha_j \leq z(v)$, then $h(v, \ell, o_v) \leq |t_\ell - m_v|$, for every $\ell \leq j$. 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: $j \geq k_v$. If $\ell < j$, the sum $h(v, \ell, k_v)$ does not contain α_j, so the corresponding constraint remains valid. If $\ell \geq j$, then the corresponding constraint is valid if and only if $h(v, j, k_v) \leq |t_j - m_v|$. This is because $\alpha_\ell = 0$ for all $\ell > j$. But $z(v)$ corresponds exactly to the amount we can increase α_i so that $h(v, j, k_v) = |t_j - m_v|$. This proves the second case. □

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

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

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.

5 Exact algorithm for MinTimeline$_b$

In this section we develop a linear-time algorithm for the problem MinTimeline$_b$. Here we are given a temporal network G, and a set of budgets $\{b_v\}$ of interval durations, and all activity intervals should satisfy $\sigma(I_v) \leq b_v$.

The idea for this optimal algorithm is to map MinTimeline$_b$ 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 MinTimeline$_b$ has a solution if and only if 2-SAT has a solution. The solution for MinTimeline$_b$ 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 [1], we have the following.

Proposition 3. MinTimeline$_b$ *can be solved in a polynomial time.*

Solving 2-SAT can be done in linear-time with respect to the number of clauses [1]. However, in our case we may have $\mathcal{O}\big(m^2\big)$ 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 [1] 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. Then, for each clause $(x_i \vee x_j)$, there are two edges in A: $(q_i \to p_j)$ and $(q_j \to p_i)$; The negations are handled similarly.

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 $|A_2| \in \mathcal{O}\big(m^2\big)$.

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 enumerate over the components, children first: if the boolean variables corresponding to the vertices in the component do not have truth assignment,[2] 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 $\mathcal{O}(|W| + |A|)$ time, and the remaining steps can be done in $\mathcal{O}(|W|)$ time. Since $|W| \in \mathcal{O}(m)$, we need to optimize the SCC decomposition to perform in $\mathcal{O}(m)$ time. We will use the algorithm by Kosajaru (see [10]) for the SCC decomposition. This algorithm consists of two depth-first searches, performing constant-time operations on each visited node. Thus, we need to only optimize the DFS.

To speed-up the DFS, we need to design an oracle such that given a vertex $p \in P$ it will return an *unvisited* neighboring vertex $q \in Q$ in *constant* time. Since $|Q| \in \mathcal{O}(m)$, this guarantees that 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 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 $\ell[v] = (q_{vt} \in Q; q_{vt} \text{ is not visited})$ sorted chronologically. Assume that we are at $p_{vt} \in P$. We retrieve the first vertex in $\ell[v]$, say q_{vs}, and compare 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.

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

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 then allows us to solve MinTimeline$_b$ in $\mathcal{O}(m)$ time.

6 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 [12]. A factor-2 approximation is available for vertex cover, by taking all vertices of a maximal matching [6]. 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 [13]. 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 [14] discovered burstiness using an exponential model over the delays between the events. Alternative techniques are based on modeling event counts in a sliding window: Ihler et al [11] modeled such a statistic with Poisson process, while Fung et al [5] used Binomial distribution. Additionally, Zhu and Shasha [26] used wavelet analysis, Vlachos et al [23] applied Fourier analysis, and He and Parker [8] adopted concepts from Mechanics to discover burst events. Finally, Lappas et al [15] propose discovering maximal bursts with large discrepancy.

A highly related problem for discovering bursty events is segmentation. Here the goal is to segment the sequence in k coherent pieces. One should expect that time periods of high activity will occur 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 [3]. Moreover, under some mild assumptions we can obtain a $(1+\epsilon)$ approximation in linear time [7].

The difference of all these works with our setting is that we consider networked 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 this increased combinatorial complexity, we consider a simpler burstiness model than previous works: in particular, we assume that each entity has only one activity interval. Extending our definition to more complex activity models (multiple intervals per entity, or multiple activity levels) is left for future work.

Event detection in temporal data. As the input to our problem is a sequence of temporal edges, our work falls in the broad area of mining *temporal* *networks* [9, 19]. More precisely, 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 of constructing different types of word graphs [4, 18, 24]. The events are detected as clusters or connected components in such graphs and temporal information is not considered directly.

Another family of methods uses statistical modeling for identify events as trends [2, 17]. Leskovec et al. [16] and Yang et al. [25] 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.

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 and co-authors have considered this problem in the context of news articles [21, 22] and scientific publications [20]. However, their approach is not directly 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.

7 Experimental evaluation

In this section we empirically evaluate the performance of our methods. [3]

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

For the *Synthetic* dataset, we start by generating a static background network of $n = 100$ vertices with a power law degree distribution (we use the configuration model with power law exponent set to 2.0). Then for every vertex we generate a ground-truth activity interval and we add 100 interactions with random neighbors. These interactions are placed consequently with unit time distance, and thus each activity interval has length of $\ell = 99$ time units. We place the ground-truth activity intervals on a timeline in an overlapping manner, and we control their temporal overlap using a parameter $p \in [0, 1]$. When $p = 0$, all intervals are disjoint and every timestamp has only one interaction, thus, it should be easy to find the correct activity intervals. When $p = 1$, all intervals are merged into one, and every time stamp has 100 of different interactions, 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.

For the case study we use a dataset collected from *Twitter*. The dataset records activity of Twitter users in Helsinki during 12.2008–05.2014. We consider only tweets with more than one hashtag (666 487 tweets) and build the

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

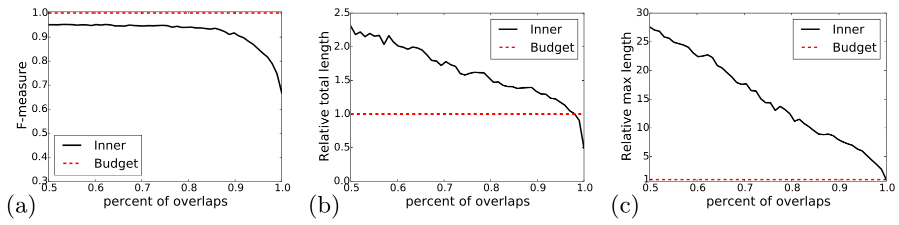

Fig. 1: Output of both algorithms for different overlaps p in the ground truth activity intervals. All values are averaged over 100 runs. (a) F-measure of correctly identifies active time-stamped vertices, (b) L, total activity interval length divided by true total activity interval length, (c) M, maximum activity interval length divided by true maximum activity interval length.

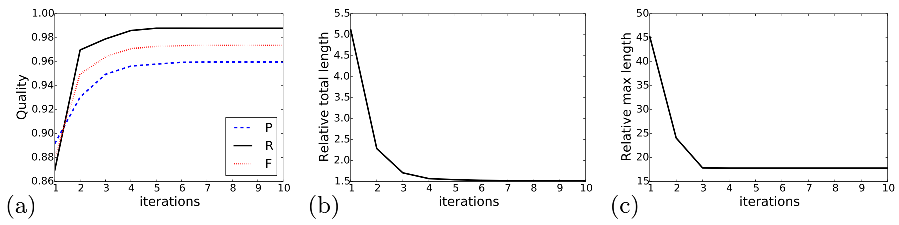

Fig. 2: Convergence of `Maximal` algorithm. Overlap p is set to 0.5, values are averaged over 100 runs. (a) Precision, Recall and F-measure, (b) L, relative total length, (c) M, relative length of the maximum interval.

co-occurrence network of these hashtags: vertices corresponding to hashtags and time-stamped edges corresponding to a tweet in which two hashtags are mentioned. The temporal network contains 304 573 vertices and 3 292 699 edges.

Results from synthetic datasets. To evaluate the quality of the discovered activity intervals we compare the set of discovered intervals with the ground-truth intervals. For every vertex u we define precision $P_u = \frac{|TP_u|}{|F_u|}$, where TP_u is the set of correctly identified moments of activity of u, and F_u is the set of all discovered moments of activity of 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 moments of activity of 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 $F = \frac{2PR}{P+R}$.

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 test both algorithms on the *Synthetic* dataset with varying overlap parameter p. The results are shown in Figure 1. All measures are averaged over

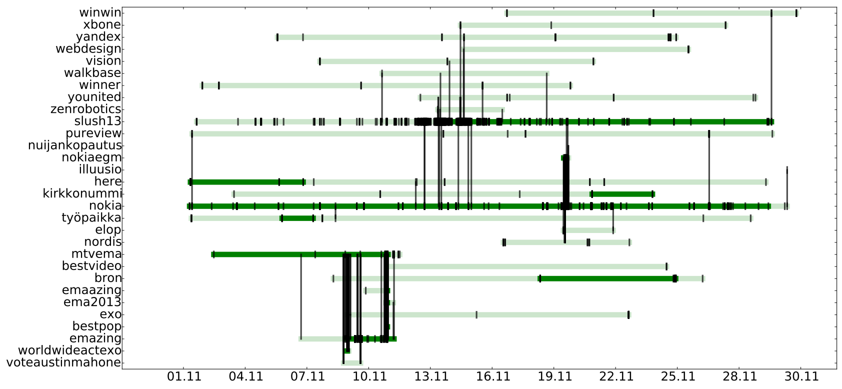

Fig. 3: Part of the output of `Maximal` algorithm on Twitter dataset for November'13. Intervals of activity of co-occurring tags, seeded from hashtags `#slush13`, `#mtvema` and `#nokiaemg`.

100 runs. Note that in the *Synthetic* dataset all activity intervals have the same length, thus, if during binary search the correct value of budget is found, then automatically all vertices receive the correct budget.

Figure 1.a demonstrates that for algorithm `Maximal` the F-measure is typically high for all values of the overlap parameter, but drops, when p increases. On the other hand, Figure 1.b shows that algorithm `Maximal` takes advantage of the overlaps and for large values of p it finds solutions that have better score than the ground truth. This however, leads to decrease in accuracy. As for the maximum interval length, shown in Figure 1.c, algorithm `Maximal` is not designed to optimize it and it typically finds few large intervals, while keeping the total length low. `Budget` finds solutions of correct total and maximum lengths on the *Synthetic* dataset for all values of overlap parameter p.

In Figure 2 we show how the solution of `Maximal` evolves during iterations with re-initialization. After a couple of iterations the value and quality (F-measure, precision and recall) of the solution are improved significantly. During the next iterations the value of the solution does not change, but the quality keeps increasing. The method converges in less than 10 iterations.

Scalability. Both `Budget` and `Inner` use linear-time algorithms in their inner loops and the number of needed outer loop iterations is small. This means that our methods are scalable. To demonstrate this, we were able to run `Maximal` with a network of 1 million vertices and 1 billion interactions in 15 minutes, despite the large constant factor due to the Python implementation.

Case study. Next we present our results on the *Twitter* dataset. In Figure 3 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 `Maximal`. 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 #nokiaemg 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 – the world's leading startup and tech event, organized in Helsinki in November 13-14, 2013. #mtvema is dedicated to MTV Europe Music Awards, held on 10 November, 2013. #nokiaemg is Extraordinary General Meeting (EGM) of Nokia Corporation, held in Helsinki in November 19, 2013.

For each hashtag we plot its entire interval with a light color, and the discovered activity interval with a dark color. For each selected hashtag, we draw interactions (co-occurrence) with other selected hashtags using black vertical lines, while we mark interactions with non-selected hashtags by ticks.

Figure 3 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 #nokiaemg 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.

8 Conclusions

In this paper we introduced and studied a new problem, which we called network untangling. Given a set of temporal undirected interactions, our goal is to discover activity time intervals for the network entities, so as to explain the observed interactions. We consider two settings: MinTimeline, where we aim to minimize the total sum of activity-interval lengths, and MinTimeline$_\infty$, where we aim to minimize the maximum interval length. We show that the former problem is **NP**-hard and we develop efficient iterative algorithms, while the latter problem is solvable in polynomial time.

There are several natural open questions: it is not known whether there is an approximation algorithm for MinTimeline or whether the problem is inapproximable. Second, our model uses one activity interval for each entity. A natural extension of the problem is to consider k intervals per entity, and/or different activity levels.

Acknowledgements. This work was supported by the Tekes project "Re:Know," the Academy of Finland project "Nestor" (286211), and the EC H2020 RIA project "SoBigData" (654024).

References

[1] Aspvall B, Plass MF, Tarjan RE (1982) A linear-time algorithm for testing the truth of certain quantified boolean formulas. IPL 14(4):195

[2] Becker H, Naaman M, Gravano L (2011) Beyond trending topics: Real-world event identification on twitter. In: ICWSM

[3] Bellman R (1961) On the approximation of curves by line segments using dynamic programming. CACM 4(6)
[4] Cataldi M, Di Caro L, Schifanella C (2010) Emerging topic detection on twitter based on temporal and social terms evaluation. In: MDMKDD
[5] Fung GPC, Yu JX, Yu PS, Lu H (2005) Parameter free bursty events detection in text streams. In: VLDB
[6] Gary MR, Johnson DS (1979) Computers and intractability: A guide to the theory of NP-completeness
[7] Guha S, Koudas N, Shim K (2006) Approximation and streaming algorithms for histogram construction problems. TODS 31(1):396–438
[8] He D, Parker DS (2010) Topic dynamics: An alternative model of bursts in streams of topics. In: KDD
[9] Holme P, Saramäki J (2012) Temporal networks. Physics reports 519(3):97–125
[10] Hopcroft JE, Ullman JD (1983) Data structures and algorithms, vol 175. Addison-Wesley Boston, MA, USA:
[11] Ihler A, Hutchins J, Smyth P (2006) Adaptive event detection with time-varying poisson processes. In: KDD
[12] Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations
[13] Khot S, Regev O (2008) Vertex cover might be hard to approximate to within $2 - \varepsilon$. JCSS 74(3)
[14] Kleinberg J (2003) Bursty and hierarchical structure in streams. DMKD 7(4):373–397
[15] Lappas T, Arai B, Platakis M, Kotsakos D, Gunopulos D (2009) On burstiness-aware search for document sequences. In: KDD
[16] Leskovec J, Backstrom L, Kleinberg J (2009) Meme-tracking and the dynamics of the news cycle. In: KDD
[17] Mathioudakis M, Koudas N (2010) Twittermonitor: trend detection over the twitter stream. In: KDD
[18] Meladianos P, Nikolentzos G, Rousseau F, Stavrakas Y, Vazirgiannis M (2015) Degeneracy-based real-time sub-event detection in twitter stream. In: ICWSM
[19] Michail O (2016) An introduction to temporal graphs: An algorithmic perspective. Internet Mathematics 12(4):239–280
[20] Shahaf D, Guestrin C, Horvitz E (2012) Metro maps of science. In: KDD
[21] Shahaf D, Guestrin C, Horvitz E (2012) Trains of thought: Generating information maps. In: WWW
[22] Shahaf D, Yang J, Suen C, Jacobs J, Wang H, Leskovec J (2013) Information cartography: creating zoomable, large-scale maps of information. In: KDD
[23] Vlachos M, Meek C, Vagena Z, Gunopulos D (2004) Identifying similarities, periodicities and bursts for online search queries. In: SIGMOD
[24] Weng J, Lee BS (2011) Event detection in twitter. In: ICWSM
[25] Yang J, Leskovec J (2011) Patterns of temporal variation in online media. In: WSDM
[26] Zhu Y, Shasha D (2003) Efficient elastic burst detection in data streams. In: KDD