Rozenshtein, Polina; Gionis, Aristides; Prakash, B. Aditya; Vreeken, Jilles

Reconstructing an epidemic over time

Published in:
KDD 2016 - Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

DOI:
10.1145/2939672.2939865

Published: 13/08/2016

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

Please cite the original version:
Rozenshtein, P., Gionis, A., Prakash, B. A., & Vreeken, J. (2016). Reconstructing an epidemic over time. In *KDD 2016 - Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Vol. 13-17-August-2016, pp. 1835-1844). ACM. https://doi.org/10.1145/2939672.2939865

Reconstructing an Epidemic Over Time

Polina Rozenshtein◦ Aristides Gionis◦ B. Aditya Prakash• Jilles Vreeken◇
◦ Department of Computer Science, Aalto University, Finland
• Department of Computer Science, Virginia Tech., Blacksburg, USA
◇ Max Planck Institute for Informatics and Saarland University, Saarbrücken, Germany
{polina.rozenshtein, aristides.gionis}@aalto.fi, badityap@cs.vt.edu, jilles@mpi-inf.mpg.de

ABSTRACT

We consider the problem of reconstructing an epidemic over time, or, more general, reconstructing the propagation of an activity in a network. Our input consists of a *temporal network*, which contains information about when two nodes interacted, and a sample of nodes that have been reported as infected. The goal is to recover the flow of the spread, including discovering the starting nodes, and identifying other likely-infected nodes that are not reported. The problem we consider has multiple applications, from public health to social media and viral marketing purposes.

Previous work explicitly factor-in many unrealistic assumptions: it is assumed that (a) the underlying network does not change; (b) we have access to perfect noise-free data; or (c) we know the exact propagation model. In contrast, we avoid these simplifications: we take into account the temporal network, we require only a small sample of reported infections, and we do not make any restrictive assumptions about the propagation model.

We develop CulT, a scalable and effective algorithm to reconstruct epidemics that is also suited for online settings. CulT works by formulating the problem as that of a temporal Steiner-tree computation, for which we design a fast algorithm leveraging the specific problem structure. We demonstrate the efficacy of the proposed approach through extensive experiments on diverse datasets.

1. INTRODUCTION

Rumours, like infections, spread over time. Consider a *temporal network*, which records when two nodes have interacted. Given such a temporal network in which a "virus" has been propagating, and a small sample of infected nodes over time, can we reconstruct the flow of the epidemic? That is, can we automatically identify the starting points of the epidemic, and reliably tell when every node got infected, including for those truly infected nodes that were never reported as such? Moreover, can we do so without having to assume a model for the virus spread? And, can we do this efficiently, with approximation guarantees, and in an online setting? We answer these questions affirmatively.

Reconstructing an epidemic, or a network propagation in general, has many important applications. These include studying how real viruses such as the flu or Ebola spread. This type of analysis is done by the Centre for Disease Control (CDC) with the goal of better understanding of viruses for both preventing and controlling outbreaks. Other examples include the analysis of influence propagation in social networks, as is done by social scientists and marketeers for better understanding and leveraging human behaviour.

Although diffusion processes have been widely studied, the problem of "reverse engineering" network propagations has received relatively little attention. Moreover, existing work only considers few and relatively simple settings [7, 11, 13, 14]. For example, it is generally assumed that the graph over which the contagion spreads is *static*, that we know the model of how the contagion spreads, and that we can obtain complete noise-free observations of the state of the network. In this paper we do not make these unrealistic assumptions.

First of all, we explicitly take into account that influence can only spread when there is an *interaction*; you have to read a status update from a friend to possibly be influenced by it. We therefore consider *temporal networks*, directed and weighted graphs that tell *when* two nodes interact. These highly dynamic graphs can be considered at different resolutions, from micro-seconds, to hours. Our algorithms work regardless of the time-resolution of the graph.

Second, we acknowledge that in reality, we can only obtain a small and noisy sample of the state of the network over time. For real viruses it is obvious that it is infeasible to test the entire population (see for example the surveillance pyramid [17]). Additionally, we face situations in which test results arrive over time, and they may include false positives [15]. To address these considerations, we consider a general setting where over time we receive (possibly noisy) reports of the state of individual nodes. Our algorithm can operate both off-line and *on-line* given a stream of reports and a stream of network interactions.

Third, we acknowledge that in practice we seldom know how influence propagates exactly, and, that parametric models such as Independent Cascade (IC) and Susceptible-Infected (SI) are strong simplifications of reality. We therefore take a non-parametric approach that relies on just a single assumption: shorter paths of infection are more likely.

All together, this allows us to define a cost of a description of an epidemic based on interactions. We show that this formulation is related to the concept of directed Steiner

KDD '16, August 13–17, 2016, San Francisco, CA, USA

ISBN 978-1-4503-4232-2/16/08…$15.00

DOI: http://dx.doi.org/10.1145/2939672.2939865

trees, and that the problem of recovering the flow of an epidemic can be efficiently solved with approximation guarantees. Empirical evaluation shows our method reliably reconstructs epidemics, outperforming meaningful benchmarks, regardless of whether the epidemic was truly generated by multiple fundamental models such as the IC, the SI, or the Forest Fire model.

2. PRELIMINARIES

Let V be a set of nodes. We write (u, v, t) to denote an *interaction* that occurs between nodes u and v at time $t \in \mathbb{R}$; so t is a *time-stamp*. Interactions are *directed*, so (u, v, t) starts in u and ends in v, and can be weighted. For simplicity, however, we consider only unweighted graphs. A *temporal network* $G = (V, E)$ consists of a set of interactions $E = \{(u, v, t) \mid u, v \in V, t \in \mathbb{R}\}$ over the set of nodes V. The number of nodes in the temporal network is denoted by $|V| = n$ and the number of interactions by $|E| = m$. We assume that more than one interaction between two nodes u and v can appear in E with different time-stamps.

For a temporal network $G = (V, E)$ and a node $u \in V$ we write $\mathcal{V}(u)$ to denote the set of all time-stamped copies of u for which u participates in some interaction of E, i.e., $\mathcal{V}(u) = \{(u, t) \mid (u, v, t) \in E \text{ or } (v, u, t) \in E\}$. We write $\mathcal{V} = \bigcup_{u \in V} \mathcal{V}(u)$ for the set of all time-stamped nodes.

Given a node u and a time interval $[t_\mathrm{s}, t_\mathrm{e}]$ we denote by $\mathcal{V}(u, [t_\mathrm{s}, t_\mathrm{e}])$ the subset of nodes of $\mathcal{V}(u)$ whose time-stamp falls in the interval $[t_\mathrm{s}, t_\mathrm{e}]$. Additionally, given an arbitrary set of time-stamped nodes $\mathcal{X} = \{(u, t)\}$ we define $V(\mathcal{X})$ to be the set of nodes that appear in $\mathcal{X}$, that is, $V(\mathcal{X}) = \{u \in V \mid (u, t) \in \mathcal{X}\}$.

Next we consider a *dynamic propagation process* in the network, such as the spread of a virus in a physical social network or the adoption of an action/idea in an online social-media platform. We use the generic term *active* to refer to nodes that are activated (infected/adopted-a-product etc.) during the propagation process.

We assume that the propagation process is starting externally, from a small set of nodes (S), and all other nodes become active via interactions. In particular, if node u is active, and an interaction (u, v, t) occurs, then the node v *may* become active at time t, or it may *not* become active; it depends on the dynamics of the propagation process. Although our approach trivially generalises to multiple states, e.g., susceptible, infected, recovered, etc., for simplicity of exposition in the remainder we assume that once a node becomes active it remains active for the rest of its lifetime.

Apart from the previous very generic assumptions, we do not assume any particular model of the propagation process.

The activity-propagation flow in the network can be described as a set of time-stamped active nodes $\mathcal{A} = \{(u, t)\} \subseteq \mathcal{V}$, where the time-stamp t indicates the first time that node u becomes active. Then we call $\mathcal{V} \setminus \mathcal{A}$ as the set of all potentially *susceptible* time-stamped nodes.

We consider the set $\mathcal{A}$ as the *true* activity propagation that occurs in the network, even though it may be *non-observable*. Instead we consider that we observe a (relatively small) set of *activity reports* $\mathcal{R} = \{(u, t)\}$. A report $(u, t) \in \mathcal{R}$ indicates that node u was reported (or observed) to be active at time t. For a node to be reported it should have been activated earlier, thus, if there is a report $(u, t_1) \in \mathcal{R}$ then there should be an activation $(u, t_0) \in \mathcal{A}$ with $t_0 \leq t_1$. We write $t_{\mathcal{R}}(u)$ to refer to the time that a node u is reported as active in $\mathcal{R}$. If a node u is never reported as active, we define $t_{\mathcal{R}}(u) = T$, where T is the last time stamp in E.

Finally, we need to define *temporal connectivity*, as network activity propagates over temporal paths. A *temporal path* $\mathbf{p}$ between two time-stamped nodes (u, t_s) and (v, t_e) is a sequence of node-disjoint interactions (u, w_1, t_1), (w_1, w_2, t_2), $\ldots$, (w_{j-1}, v, t_j), such that $t_\mathrm{s} \leq t_1$, $t_j \leq t_\mathrm{e}$ and $t_i \leq t_{i+1}$ for all $1 \leq i \leq j - 1$. Given a time-stamped *root* node (r, t_0) and a set of *terminals* $\mathcal{T} = \{(u, t)\}$, a *rooted temporal Steiner tree* is a subgraph of the temporal network G, which contains r with no incoming interactions and unique temporal paths from (r, t_0) to every terminal node $(u, t) \in \mathcal{T}$.

Let $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_\ell\}$ be a set of temporal paths. Denote by $\mathcal{V}(P)$ the set of *time-stamped* nodes that appear in any path in P, and by $V(P) = V(\mathcal{V}(P))$ the set of *non time-stamped* nodes that appear in any path in P. We say that P *spans* a set of time-stamped nodes $\mathcal{X} \subseteq \mathcal{V}$ if every $(u, t) \in \mathcal{X}$ comes *after* (i.e. $t \geq t'$) corresponding $(u, t') \in \mathcal{V}(P)$.

Finally, given a set of temporal paths P, a node u is called a *seed* if $u \in V(P)$ and there is no interaction $(v, u, t) \in \mathbf{p}_i$ for any $\mathbf{p}_i \in P$. The set of seed nodes induced by a set of temporal paths P is denoted by $S(P)$.

3. PROBLEM FORMULATION

Given a temporal network and a (partial) report log of active nodes, our goal is to discover a sequence of time-stamped node activations (which nodes and when) that explains best the observed report log, given the set of interactions in the temporal network. In general, the problem is as follows.

PROBLEM 1. *We are given a temporal network $G = (V, E)$, where E is a set of interactions over a set of nodes V. We are also given a report log $\mathcal{R} = \{(u, t)\}$ of time-stamped activated nodes, a set of candidate seeds C, and an integer k. The goal is to find a set of temporal paths P in order to minimize a cost function* $\mathrm{cost}(P \mid \mathcal{R})$ *subject to:*

(i) P spans $\mathcal{R}$;

(ii) $S(P) \subseteq C$; and

(iii) $|S(P)| \leq k$.

To make Problem 1 concrete we need to define the cost function $\mathrm{cost}(P \mid \mathcal{R})$. Our approach is to introduce a weight function $w : E \to \mathbb{R}$ on the network interactions E, which depends on the temporal network G and the report log $\mathcal{R}$.

The weight function that is used in all the experiments of this paper is discussed in Section 5. We note, however, that there are many possible weighting schemes that can be explored and tested in practice. Our problem formulation is also meaningful for uniform weights, in which case we want to explain the activity-propagation history with as few interactions as possible. In any case, our theoretical results do not depend on the particular definition of w and are applicable for any weight function.

Given a weight function, we then assume that the cost of a path is the sum of the weights of the interactions on the path. Then it is natural to model the total cost of the set of paths P, as total cost of all interactions that are included in P, i.e.,

$$\mathrm{cost}(P \mid \mathcal{R}) = \sum_{e \in P} w(e). \tag{1}$$

PROBLEM 2. (TEMPSTEINERTREE) *Solve Problem 1 with cost function* $\mathrm{cost}(P \mid \mathcal{R})$ *given by Equation (1).*

Regarding the complexity of the TempSteinerTree problem, we can show the following.

Proposition 1. *Problem* TempSteinerTree *is* **NP***-hard.*

A proof of Proposition 1 can be obtained in a similar manner as in the work of Huang et al. [6].

4. FINDING TEMPORAL STEINER TREES

To solve the Steiner-tree problem on temporal networks, as defined in the previous section, we map it to the standard Steiner-tree problem on *static* (not temporal) *directed* graphs. The latter problem is defined as follows.

Problem 3. (MinDirSteinerTree) *We are given a directed static graph $H = (U, F)$ with edge weights $w : F \to \mathbb{R}$, a root node $r \in U$ and a set of terminal nodes $R \subseteq U$. We want to find a directed tree T, rooted in r and containing directed paths from r to every node in R such that total tree cost $\sum_{e \in F(T)} w(e)$ is minimized.*

We first show that for $k = 1$ the TempSteinerTree problem can be mapped to MinDirSteinerTree.

Proposition 2. *For $k = 1$ the* TempSteinerTree *problem can be solved by $|C|$ calls to* MinDirSteinerTree.

Proof. Let us consider an instance of the TempSteinerTree problem: a temporal network $G = (V, E)$, a report $\log \mathcal{R} = \{(u, t)\}$, and a set of candidate seeds C. Given this problem instance, we construct a directed static graph $H = (U, F)$ as follows. We first define U to be the set of time-stamped nodes $\mathcal{V}$. For each node $u \in V$ of the temporal network G we order its time-stamped copies $\{(u, t)\}$ in $\mathcal{V}$ in ascending time order, and we connect every two consequent copies (u, t_i) and (u, t_{i+1}) by a directed edge $e((u, t_i), (u, t_{i+1}))$, which is added in the set of static edges F with zero weight. Additionally, for each interaction $(u, v, t) \in E$ we create a directed edge $e((u, t), (v, t))$, which is added in F with weight $w((u, t), (v, t)) = w(u, v, t)$.

From the construction it follows that there is a bijection between the set of all temporal paths in G and the set of all directed paths in H.

Next, for every candidate seed node $u \in C$ we create a dummy node $z(u)$. The set of all dummy nodes is $z(C)$ and it is also added to the set of static nodes U. For each candidate seed node $u \in C$ we define $t_0(u) = \arg\min\{t \mid (u, t) \in \mathcal{V}\}$, that is, the earliest possible start of a temporal path for that seed node. An edge connecting $z(u)$ with $(u, t_0(u))$ is created and it is added to F with zero weight.

Finally, we need to specify the set of terminals R for MinDirSteinerTree. For each report $(u, t) \in \mathcal{R}$ we find the latest time $t^*(u, t)$ that node u participated in an interaction that took place before t, that is, $t^*(u, t) = \arg\max\{t' \mid (u, t') \in \mathcal{V} \text{ and } t' \leq t\}$, and we add $(u, t^*(u, t))$ in the set of terminals R.

It is easy to see that there is a bijection between the set of directed Steiner trees of H rooted in $z(C)$ and the set of temporal Steiner trees in G rooted in C. Furthermore, the corresponding trees have the same cost. Also if a terminal is connected in a directed Steiner tree then the corresponding report is covered in the corresponding temporal Steiner tree. It follows that the TempSteinerTree problem in G can be solved by $|C|$ calls to the MinDirSteinerTree problem in H, one for each root $z(u)$, for seed $u \in C$. □

The next step is to consider the TempSteinerTree problem for $k > 1$. The straightforward generalization of the construction presented in Proposition 2 is to create a dummy node for each k-size subset of C, and make a call to MinDirSteinerTree for each such dummy node. This approach, however, is not scalable as it requires $\binom{|C|}{k}$ calls to MinDirSteinerTree.

To avoid such a combinatorial explosion we follow a different approach. We introduce a Lagrange-type parameter α and we modify the objective by adding a term representing the number of temporal Steiner trees, weighted by α. A target value for k can be found by varying α. The α-version of TempSteinerTree is defined below.

Problem 4. (α-TempSteinerTree) *We are given a temporal network $G = (V, E)$, reports $\mathcal{R}$, a set of seed candidates C, and a number α. We want to find a set of temporal Steiner trees $\mathcal{T}$, which spans all nodes in $\mathcal{R}$ and the total cost $\text{cost}_\alpha(\mathcal{T}) = \alpha|\mathcal{T}| + \sum_{T \in \mathcal{T}} \sum_{e \in T} w(e)$ is minimized, where $|\mathcal{T}|$ denotes the number of temporal Steiner trees in $\mathcal{T}$.*

It is easy to see, by setting α appropriately large so that the optimal solution contains only one tree, that Problem 4 remains an **NP**-hard.

Proposition 3. *Problem α-*TempSteinerTree *is* **NP***-hard.*

We can now show that for a given value of α, α-TempSteinerTree can be reduced to an instance of the TempSteinerTree problem with $k = 1$. The mapping follows the proof of Proposition 2 and constructs a static graph H. In addition, a new dummy node z_0 is added, and it is connected via a directed edges of weight α to other dummy nodes $z(u)$ (as before there is one dummy node $z(u)$ for each candidate seed $u \in C$). Then, given a minimum Steiner tree T^* on H rooted at z_0, the optimal set of Steiner trees for the α-TempSteinerTree problem can be obtained as the level-1 branches of T^*. Thus, we obtain the following proposition.

Proposition 4. *For a given value of α, the problem α-*TempSteinerTree *can be solved with one call to the* MinDirSteinerTree *problem.*

For a z_0-rooted Steiner tree T^* returned as a solution to the MinDirSteinerTree problem, we denote the degree of z_0 in T^* by $d(z_0 \mid T^*)$. Note that for a large α the degree of z_0 will be small, while for $\alpha = 0$ the degree of z_0 can be as large as the number of candidate seeds in C.

As our goal is to find a target number k of temporal Steiner trees by varying α, and as the number of returned Steiner trees is given by $d(z_0 \mid T^*)$ in the solution of MinDirSteinerTree, we would like to show that $d(z_0 \mid T^*)$ is a monotone function of α.

Proposition 5. *Let $\alpha_1 < \alpha_2$ and consider the optimal Steiner trees $T^*_{\alpha_1}$ and $T^*_{\alpha_2}$ obtained by the mapping to* MinDirSteinerTree. *Then $d(z_0 \mid T^*_{\alpha_1}) \geq d(z_0 \mid T^*_{\alpha_2})$.*

Proof. Let c_1 and c_2 be total costs of the branches of z_0 in the optimal trees $T^*_{\alpha_1}$ and $T^*_{\alpha_2}$, respectively. By the optimality of the two solutions in their corresponding problem instances we have

$$\alpha_1 d(z_0 \mid T^*_{\alpha_1}) + c_1 \leq \alpha_1 d(z_0 \mid T^*_{\alpha_2}) + c_2$$

and

$$\alpha_2 d(z_0 \mid T^*_{\alpha_2}) + c_2 \leq \alpha_2 d(z_0 \mid T^*_{\alpha_1}) + c_1.$$

It follows that

$$\alpha_1 \left(d(z_0 \mid T^*_{\alpha_1}) - d(z_0 \mid T^*_{\alpha_2})\right) \leq \alpha_2 \left(d(z_0 \mid T^*_{\alpha_1}) - d(z_0 \mid T^*_{\alpha_2})\right)$$

and thus,

$$(\alpha_1 - \alpha_2)\left(d(z_0 \mid T^*_{\alpha_1}) - d(z_0 \mid T^*_{\alpha_2})\right) \leq 0.$$

Hence, $d(z_0 \mid T^*_{\alpha_1}) \geq d(z_0 \mid T^*_{\alpha_2})$. □

As the number of branches of z_0 in T^* is a non-increasing function of α we can use binary search on α to obtain a feasible solution for the TempSteinerTree problem, for a given k. The lower boundary for the binary search is set to $\alpha_\ell = 0$, while the upper boundary α_u must be selected so as to ensure that $d(z_0 \mid T^*_{\alpha_u}) \leq k$, or asserting that it is not possible to find a tree with less than k branches from z_0. The following proposition provides such a value for α.

Proposition 6. *Let L be the weighted length of the longest shortest path from a node in the candidate seed set C to a node in the reports set $\mathcal{R}$. Then the value $\alpha_u = L|\mathcal{R}|$ provides a Steiner tree $T^*_{\alpha_u}$ for which $d(z_0 \mid T^*_{\alpha_u})$ is the smallest possible degree for z_0 among all Steiner trees that can be found as a solution to the* MinDirSteinerTree *problem, for any α.*

Proof. Let c_1 be the minimum total cost along the branches of the Steiner tree that has the smallest possible number of branches $k_{\min}$ from z_0. Let c_2 be the minimum total cost along the branches of any feasible Steiner tree with more than $k_{\min}$ branches from z_0.

We want a value of α for which solution tree T^*_α has at most $k_{\min}$ branches from z_0. Thus, we require $\alpha k_{\min} + c_1 \leq \alpha(k_{\min} + 1) + c_2$, which gives $\alpha \geq c_1 - c_2$. Upper bounding c_1 by $L|\mathcal{R}|$ and lower bounding c_2 by 0 proves the claim. □

Note that the value of α given by Proposition 6 guarantees that $d(z_0 \mid T^*_\alpha) = 1$, if there exists a directed Steiner tree rooted in one of the nodes of the candidate seeds C and spanning all nodes in the reports set $\mathcal{R}$.

4.1 Solving the directed Steiner-tree problem

The best approximation algorithm for the directed Steiner-tree problem is given by Charikar et al. [1]. The algorithm is based on the recursive construction of ρ-level trees by concatenating subtrees with the lowest marginal *normalized length*. The approximation guarantee of this algorithm is $\rho(\rho-1)s^{1/\rho}$, where ρ is a depth of the recursion and s is the number of terminals. Note that ρ is a user-controled parameter that offers a quality–efficiency trade-off. Increasing ρ improves the approximation guarantee of the algorithm at the cost of running time. Huang et al. [6] proposed a time-efficient modification of this algorithm with running time $\mathcal{O}(n^\rho s^\rho)$, where n is the number of nodes in the graph. However, the algorithm is still not practical for large networks.

In this section we show how to further improve the running time of the Steiner-tree algorithm so that it becomes scalable to large networks. Our algorithm leverages the special structure of the problem, that is, the fact that the static directed graph is derived from a temporal network. We show that it is sufficient to keep track only on a small number of paths, which we call *global shortest temporal paths*.

Algorithm 1: $A_i(j, r, X)$: i-level directed Steiner tree (Charikar et al. [1])

Input: Directed graph $H = (U, F)$, set of uncovered terminals $X \subseteq U$, root $r \in U$, number of terminals to be covered j, level i.
Output: Steiner tree T rooted at r that covers at least j nodes of the set of terminals X.

```
if there are not j terminals in X reachable from r then
    return ∅
T = ∅;
while j > 0 do
    T* = ∅;
    foreach r' ∈ U and j' : 1 ≤ j' ≤ j do
        T' = A_{i-1}(j', r', X) ∪ {(r, r')};
        if len(T*) > len(T') then
            T* = T';
    T = T ∪ T*;
    j = j − |X ∩ U(T*)|;
    X = X − U(T*);
return T
```

Additionally, our algorithm can be executed in an *online* fashion, as new interactions and/or new reports arrive.

The basic approximation algorithm. We start by describing the basic algorithm of Charikar et al. [1], which is also presented as Algorithm 1. Let $H = (U, F)$ be the directed graph defined in the proof of Propositions 2 and 4. For any pair of nodes $(u, v) \in U \times U$ we define $\ell(u, v)$ to be the length of the shortest path from u to v in H. If no such path exists, then $\ell(u, v) = \infty$.

The algorithm uses the variable $A_i(j, r, X)$ to denote a Steiner tree rooted at node r and covering at least j nodes from a set of terminals $X \subseteq U$. The zero-level tree Steiner tree $A_0(j, r, X)$ consists of the root r itself. The Steiner tree $A_1(j, r, X)$ is constructed by connecting the root r with the j closest terminals in X. For $i > 1$ the algorithm *recursively* finds the best subroot r', and the best number of terminals j' so as to minimize the *normalized length* of the subtree $T' = A_{i-1}(j', r', X) \cup \{(r, r')\}$, defined as total edge length per terminal covered. The best choice for T' is used as a subtree in the current level of recursion, the terminals reachable by T' are declared covered, and the algorithm continues to cover the remaining $j - j'$ terminals required. As mentioned, the normalized length of a tree T is defined as its total length divided by the number of covered terminals, that is,

$$\text{len}(T) = \frac{\sum_{e \in F(T)} \ell(e)}{|X \cap U(T)|}.$$

Algorithm 1 provides an approximation guarantee for the MinDirSteinerTree but is quite inefficient. In order to avoid re-computing shortest paths multiple times, we can pre-compute all shortest-path distances $\ell(u, v)$ for all pairs of nodes $(u, v) \in U \times U$, but such a pre-computation requires $\mathcal{O}(|U|^2)$ space and $\mathcal{O}(|U|^\omega)$ time.[1] In terms of the temporal-network input size, it is $|U| = \mathcal{O}(|\mathcal{V}|) = \mathcal{O}(|E|)$, where E is the set of interactions in the network. When space requirements are limited we can execute the algorithm without pre-

[1] $\mathcal{O}(n^\omega)$ refers to the fastest algorithm for matrix multiplication. The current value of the exponent is $\omega \approx 2.373$.

computing all-pairs shortest paths, but then running-time cost is higher. Clearly such a space and running-time complexity is prohibitive for large networks.

Improving the running time. We can improve the running time of the algorithm significantly by exploiting the special structure of our problem, namely, the fact that the directed graph $H = (U, F)$ has been constructed in a specific manner from the input temporal network. Recall that the set of nodes of H consists of time-stamped copies of nodes $\{(u,t)\} = \bigcup_{u \in V} \mathcal{V}(u)$ and dummy nodes $\bigcup_{u \in C}\{z(u)\} \cup \{z_0\}$, where $z(u)$ is a dummy node for a candidate seed $u \in C$.

Our main observation is that it suffices to compute shortest paths only between dummy nodes $z(u)$ and a *single* time-stamped node (v,t), for each candidate seed $u \in C$ and each $v \in V$. The time-stamp of (v,t), for which the shortest path is kept, is the report time $t_{\mathcal{R}}(v)$ (recall that if a node v is not in the reports set, then $t_{\mathcal{R}}(v)$ is defined to be the last time stamp in E). We refer to these paths as *global shortest temporal paths*. More specifically, the global shortest temporal path between candidate seed $u \in C$ and node $v \in V$ is defined to be

$$\mathbf{q}(u,v) = \arg\min_{\mathbf{p}} \left\{ \min_{\substack{(v,t_j) \in \mathcal{V}(v) \\ t_j \leq t_{\mathcal{R}}(v)}} \ell(\mathbf{p}(z(u),(v,t_j))) \right\},$$

where $\mathbf{p}$ is a temporal path. We also write $L(u,v) = \ell(\mathbf{q}(u,v))$ to denote the length of a global shortest temporal path $\mathbf{q}(u,v)$. It is easy to see that we can compute and store global shortest temporal paths using $\mathcal{O}(|C||V|)$ space and $\mathcal{O}(|C||E|)$ time. Furthermore, global shortest temporal paths can be updated for all candidate seeds in $\mathcal{O}(|C|)$ time, when a new interaction arrives in the temporal network. Thus, global shortest temporal paths not only yield significant performance improvement but they also make the algorithm suitable to be executed in a streaming fashion.

If we consider a 2-level Steiner tree, we can notice that it essentially consists of a dummy root z_0 with branches, rooted in dummy candidate seeds $z(u)$. Each branch is a 1-level Steiner tree, thus it is simply a set of shortest paths from $z(u)$ to terminals. By definition, a global shortest temporal path is no longer than any other path to a terminal. Thus, each branch can be viewed as a set of global shortest temporal paths from $z(u)$ to terminals. In addition, for a 2-level Steiner tree we can avoid trying all possible subsets of terminals to be covered: we can simply consider terminals in order of increasing global shortest path length.

We can now show that while keeping only global shortest temporal paths we can still calculate the minimum-cost Steiner tree on H with the same guarantee as Algorithm 1.

PROPOSITION 7. *Consider the transformed directed graph $H = (U,F)$, and assume a set of candidate seeds $C = U$. Assume that we want to find a Steiner tree rooted at z_0 and spanning a set of terminals X, with $|X| = s$. Then, by executing Algorithm 1 for depth $\rho = 2$ and using only global shortest temporal paths we can calculate the minimum-cost Steiner tree with approximation guarantee $\rho(\rho-1)s^{1/\rho}$.*

PROOF. First note that the global shortest temporal paths prune away only information about non-optimal paths from a dummy node $z(u)$ to a node $(v,t) \in U$, while preserving the shortest one. To see this, consider a global shortest temporal path $\mathbf{q}(u,v)$ from $z(u)$ to (v,t_i). For all shortest paths $\mathbf{p}'$ in H between $z(u)$ and a time-stamped node (v,t_j) with $t_j \geq t_i$ we have $\ell(\mathbf{p}') = L(u,v)$.

Next, we show that for any best tree selected in the for-loop of Algorithm 1, there is always a tree, which has at most the same normalized length and it is rooted in a dummy node $z(u)$, for some $u \in C$.

Indeed, assume that Algorithm 1 picks a 1-level tree T', which is rooted at a node (u,t') and covers a subset of terminals $X' = \{(v_1,t_1),\ldots,(v_j,t_j)\} \subseteq \mathcal{V}$. Notice that since X' is reachable from (u,t'), it is also reachable from any (u,t), with $t \leq t'$, and also from $z(u)$. Let T be the tree rooted in $z(u)$ that covers the same set of terminals X'. We claim that T is less expensive than T'. To see this note that the cost of tree T' is $\sum_{x \in X'} \ell((u,t'),x)$ and the cost of tree T is $\sum_{x \in X'} \ell(z(u),x)$, while by the definition of the global shortest temporal path $\ell((u,t'),x) \geq \ell(z(u),x)$. □

Proposition 7 requires that the set of candidate seeds C is equal to U. Note that in many applications we do not have a reason to exclude any candidate seeds, so this is a perfectly reasonable assumption. In cases that the set of candidate seeds is a subset of the whole node set we can still use the algorithm and obtain improved running time, however, the approximation guarantee does not hold.

The resulting algorithm is presented in detail as Algorithm 2. The input is the temporal network $G = (V,E)$, a set of candidate seeds C, and a report log $\mathcal{R}$. For every candidate $v \in C$ (and corresponding dummy node $z(v)$) we compute the global shortest temporal path to every node in V and sort these paths in ascending length order. While not all terminals are covered, the algorithm computes the cost of all possible 1-level trees composed of global shortest temporal paths, finds the one with the smallest normalized density, and adds it to the solution. The ordering the global shortest temporal paths allows to test all possible trees in linear time. The algorithm returns a set of trees $\mathcal{T}$, which reconstructs the observed activity propagation.

Algorithm 2: Temporal Steiner forest construction

Input: Temporal network $G = (V,E)$, set of candidate seeds C, and reports $\mathcal{R}$.
Output: Temporal Steiner forest $\mathcal{T} \subseteq G$, rooted at seed candidates C and spanning reports $\mathcal{R}$.

```
foreach u ∈ C and v ∈ V do
    obtain global shortest temporal path L(z(u), v);
T = ∅;
X = V(R);
while X ≠ ∅ do
    T'_1 = ∅;
    foreach z(u) ∈ C do
        T_1 = ∅; cost = α; profit = 0;
        for v ∈ X ordered by increasing L(z(u), x) do
            T_1 = T_1 ∪ q(z(u), v);
            cost += L(z(u), v);
            profit += 1;
            if len(T'_1) > cost/profit then
                T'_1 = T_1; len(T'_1) = cost/profit;
    T = T ∪ T'_1;
    X = X \ V(T'_1);
return T;
```

Algorithm 3: Global shortest temporal paths

Input: Time series of interactions E, set of seed candidates C and reports $\mathcal{R}$.
Output: global shortest temporal path from each seed candidates to each node in $\mathcal{R}$.

```
L = ∅; L* = ∅;
foreach e = (u, v, t) ∈ E do
    foreach x ∈ C with t ≥ t_0(x) do
        if u = x then
            L(x, x) = 0;
            q(x, x) = (t, x, x);
            if u ∈ V(R) and t ≥ t_R(x) and
            L*(x, x) = NaN then
                L*(x, x) = L(x, x); q*(x, x) = q(x, x)
        if L(x, u) ≠ NaN then
            cost = L(x, u) + w(e);
            if L(x, v) > cost then
                L(x, v) = cost;
                q(x, v) = q(x, u) ∪ {(u, v, t)}
            if v ∈ V(R) and t ≥ t_R(v) and
            L*(x, v) = NaN then
                L*(x, v) = L(x, v); q*(x, v) = q(x, v)
foreach (x, x) ∈ L do
    if L*(x, x) = NaN then
        L*(x, x) = L(x, x)
return L* and q*;
```

Note that the performance of Algorithm 2 can be further improved by incorporating the pruning techniques described in the recent paper by Huang et al. [6]. Details on this improvement are omitted for lack of space.

4.2 Computing global shortest temporal paths

We now describe the last piece of our algorithm, how to compute global shortest temporal paths. The main observation for computing temporal shortest paths from a set of time-stamped nodes $\mathcal{V}(u)$, for a candidate seed $u \in C$, it sufficies to consider only the shortest paths from the corresponding dummy node $z(u)$. We can then compute global shortest temporal paths by processing the interactions in ascending time order and updating the current shortest paths from each dummy node $z(u)$ to all nodes in V.

The algorithm is shown in detail in Algorithm 3. For each candidate seed $u \in C$ and node $v \in V$ we keep the temporal shortest path $\mathbf{p}(z(u), v)$ and its length $\ell(z(u), v)$. For each new interaction (w_1, w_2, t) processed (in ascending time order) we check if the current shortest temporal path $\mathbf{p}(z(u), v)$ can be improved by using (w_1, w_2, t). Once we process an interaction for which $t \geq t_{\mathcal{R}}(v)$, we fix the current $\mathbf{p}(z(u), v)$ to be the global shortest temporal path from $z(u)$ to v. However, we keep updating $\mathbf{p}(z(u), v)$, as it may be used by other shortest paths.

4.3 Putting everything together

We summarize our main algorithm, as presented in the previous sections. Starting with the temporal network G a set of reports $\mathcal{R}$, and a target number of candidate seeds k, we solve the α-TempSteinerTree problem using the transformation presented in Proposition 4 and Algorithm 2. Using binary search, a value of α is found, for which the root (z_0) of the returned Steiner tree has degree equal to k.

5. EXPERIMENTS

In this section we empirically evaluate CulT. The implementation of all algorithms and scripts used for the experimental evaluation are publicly available.[2]

5.1 Setup

Datasets. We consider both synthetic and real temporal networks, and both synthetic and real activity propagations.

For *Synthetic* dataset, we start by generating a static background network of $n = 100$ nodes with a powerlaw degree distribution. Uniformly at random we choose k seed nodes, from which we start the activation propagation process using some model. Keeping track of the sequence of successful activations, we add $\delta = 100$ random interactions between each pair of consecutive activating interactions.

We also consider real networks with simulated activity propagations. We consider *Facebook*, *Tumblr*, *Students*, and *Enron*. We use subgraphs of these networks with $n = 100$ nodes, obtained by BFS starting from a random node.

Facebook is a 3-month subset of Facebook activity in a New Orleans community [16]. The dataset contains anonymized list of wall posts (interactions). *Tumblr* is subset of the Memetracker dataset,[3] which contains quoting between Tumblr users. *Students*[4] is an activity log of a student online community at the University of California, Irvine. Nodes represent students and edges represent messages, where the message direction is suppressed. Finally, *Enron*[5] is a well-known dataset of email communication. For experiments with real-world infection cascades we considered emails, containing word *California*, as infected.

To generate the infection paths, we consider four different propagation models. In particular, we experiment with susceptible-infected (*SI*), shortest path (*SP*), independent cascade (*IC*), and forest fire (*FF*). For each we simulate a propagation until at least half of the nodes are activated. For *SI* we use infection probability 0.1, threshold number of active neighbors for *FF* is 1, activation probability for *IC* is set to inverse of the largest eigenvalue of an adjacency matrix (related to the so-called 'epidemic threshold' [10]).

Finally, to create the reports $\mathcal{R}$, we use two different schemes. In the first (refer as *RS*), for each interaction every activated node has a probability β to be reported. In the second (*FR*), nodes at the frontier of the activation are reported after a θ interactions. By combining the two schemes we can evaluate how strongly the methods rely on frontier and on intermediate reports.

Our last dataset is *Flixster*[6]: A dataset from movie ratings social network. Given a friendship network and data on which user has rated which movie and when, we create an interaction (u, v, t_j) in E, if user v has rated movie f at t_j, his friend user u has rated f at t_i and $t_j - t_i \leq 7$ days.

[2] https://github.com/polinapolina/reconstructing-an-epidemic-over-time
[3] snap.stanford.edu/data/memetracker9.html
[4] toreopsahl.com/datasets/#online_social_network
[5] www.cs.cmu.edu/~./enron/
[6] www.cs.ubc.ca/~jamalim/datasets

For *Flixster* we took a movie f (ID=54053), which was rated by 10 K users, constructed a history of interactions E, and defined the first 10 users who rated f before their friends to be seeds. We consider all interactions, induced by movie f and propagated from these seeds. Then we sample reports among frontier nodes with probability 0.5 and delay $\theta = 1000$ interactions.

Baselines. As no other methods exist to reconstruct an epidemic in a temporal network, we compare CULT to two sensible baselines. The first is straightforward, we simply account for the given reports $\mathcal{R}$ (REPORTS). Somewhat more refined, the second baseline (BASELINE) returns the one-hop-cascade from the given reports. That is, given a reported activation (u, t) BASELINE assumes that every future interaction (u, v, t) leads to a successful activation. The activation is not propagated further than one-hop-neighbors of a reported node, however, as that unduly harms precision. Although none of the two baselines returns a collection of k temporal Steiner trees, we can still evaluate their accuracy against a ground-truth set of activated nodes. For instance, note that REPORTS has precision 1.0.

Weighing scheme. As pointed out in Section 3, our approach relies on a weighing scheme to identify which paths are to have participated in the activity-propagation process. In particular, for a node u that is reported to be active at time $t_{\mathcal{R}}(u)$ we assume that an interaction (u, v, t) is more likely to have contributed in the activation of u if the time of the interaction t is close to the report time $t_{\mathcal{R}}(u)$. Thus, we set the weight of an interaction to be the average time difference between the interaction time-stamp and the report times of the two interaction end-points. In other words, we set $w(u, v, t) = \frac{1}{2}(|t - t_{\mathcal{R}}(u)| + |t - t_{\mathcal{R}}(v)|)$.

Measures. To evaluate the quality of a set of temporal Steiner trees $\mathcal{T}$, we compare the set of nodes in the Steiner trees with a ground-truth set of activated nodes. To measure the quality we use Matthews correlation coefficient (MCC):

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}},$$

where $TP = |V(\mathcal{A}) \cap V(\mathcal{T})|$, $TN = |V \setminus (V(\mathcal{A}) \cup V(\mathcal{T}))|$, $FP = |V(\mathcal{T}) \setminus V(\mathcal{A})|$, and $FN = |V(\mathcal{A}) \setminus V(\mathcal{T})|$.

We also evaluate how well we recover the *temporal order of activations*, comparing the activation order in the discovered Steiner trees with the ground-truth order. As the activation order is captured by a *set* of interactions, we measure agreement with the ground truth in precision and recall.

We ignore the exact time of the activation and we consider the sets of pairs of nodes (u, v) so that u activated v. We again compare the set of pairs of nodes discovered by our method against the ground-truth set. We refer to these sets of node pairs as *static order of activation*, and we report precision and recall.

All reported values are averaged over 100 runs.

5.2 Effect of binary search

We start by evaluating the effect of binary search. The question we want to address is the following: can we find a value for α to obtain a target value of seeds k? The relation of α vs. k is shown in Figure 1. We use the *Facebook* dataset with parameters: *SI*, reporting scheme $FR(\theta = 100)$, and $\delta = 100$. The target number of seed is set to $k = 5$.

To obtain Figure 1 we solve α-TEMPSTEINERTREE for a range of values of α. All reported values are averaged over 100 of runs. The left-most plot shows how α influences the number of seeds found. This result is noteworthy, as in Proposition 5 we assume that we can solve α-TEMPSTEINERTREE optimally, while in practice we can only solve it approximately. Since the dependence of α vs. k is monotonic also in practice, we can conclude that the binary-search approach is effective. The right-most plot shows the solution cost as a function of k. Although in this paper we do not address the problem of discovering the optimal value of k, note the "elbow" near the ground-truth value of $k = 5$.

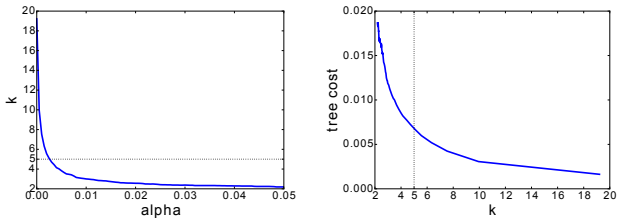

Figure 1: α is (easily) optimizable. True $K = 5$.

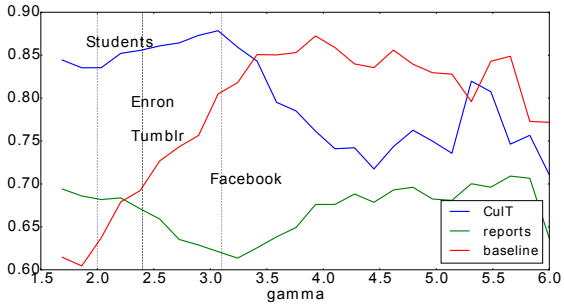

Figure 2: *MCC* for *Synthetic* powerlaw graphs of varying γ. We indicate the best-fitting values of γ for each of the real world datasets we consider.

5.3 Accuracy on powerlaw graphs

The degree distribution of many real-world graphs closely follows a power-law with γ between 2 and 3. Hence, it is interesting to evaluate how well CULT performs on such data. We run CULT on synthetic networks generated for different values of γ. *MCC* scores are shown in Figure 2. We observe that CULT is particularly accurate in the range of $\gamma = [1.5, 3.5]$, which corresponds to the range in which most real networks fall. To corroborate, we also indicate the values for γ that best fit the real networks used in this paper.

5.4 Noise level and different infection models

Next, we investigate how well CULT can reconstruct propagations for a range of different settings.

First, we simulate the *SI* model on our four real-world datasets, and inject different numbers of noise interactions. *MCC* scores are reported in Figure 3. We observe that the performance of CULT slightly declines as the fraction of relevant interactions decreases. That is, even if only very few interactions are related to the activity propagation, we are still able to recover the ground-truth propagation.

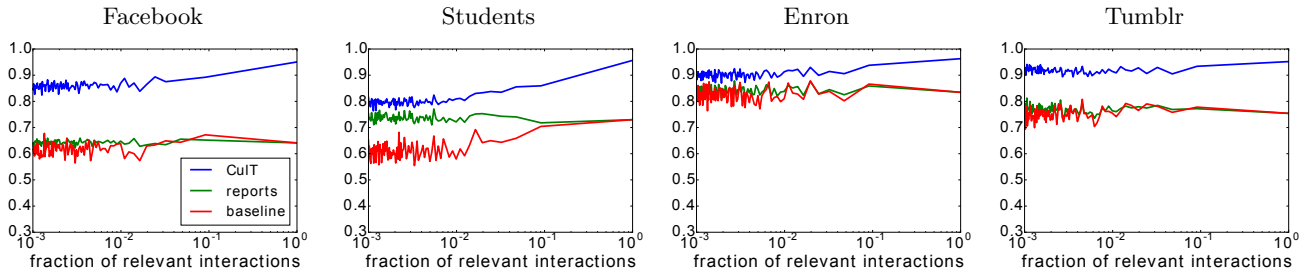

Figure 3: Effect of the fraction of interactions in interaction history E that are relevant to the propagation. Quality of reconstruction measured in MCC for four datasets, with the *SI* model used to simulate propagations.

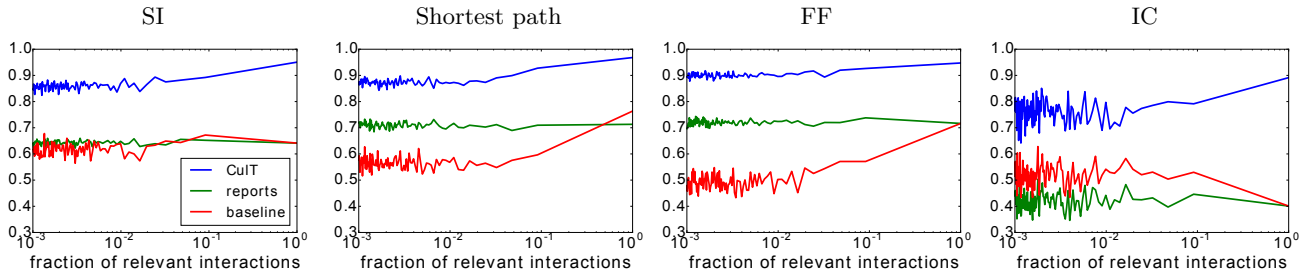

Figure 4: Effect of the fraction of interactions in the interaction history E that are relevant to the propagation. Reconstruction quality measured by MCC on the *Facebook* dataset, for different infection models.

Second, we evaluate how well we can reconstruct propagations generated by different models. We simulate cascades on the *Facebook* network using all four models: *SI*, *SP*, *IC*, and *FF*. We again vary the number of relevant interactions. The results are shown in Figure 4. CulT performs very consistently, regardless of the generating model.

Last, we check how the delay between a node activation and its reporting affects performance. We simulate the *SI* model on the four real datasets, and vary the delay between 0 and 5000 time steps. Results are provided in Figure 5. CulT is almost not affected at all by delays in reports, whereas the performance of Baseline deteriorates quickly.

5.5 Order of infection

Next, we evaluate how well we can reconstruct the *order* of infections in our temporal network. We use the frontier (*FR*) reporting scheme, while varying the fraction of reported frontier nodes. Figure 6 shows that the precision for both *temporal order* and *static order* is quite robust. In both cases the fraction of reported frontier nodes affects only the recall. When the number of the frontier nodes in the reports is low, CulT ignores many activated "leaves," which are neither in the reports, nor on the path to any other reported node. Thus, the number of false negative increases.

Naturally, the accuracy for static order is higher than for temporal order: an interaction between the same two nodes can occur at several time moments and it is difficult to select the correct one into a reconstructed propagation path.

5.6 Real cascades

Next we present our results on the *Flixster* dataset. The sequence of interactions E is divided into epochs $(t_0, t_1, \ldots, T)$. At the end of each epoch t_i we consider a snapshot of the network and compare the set of discovered activated nodes with the set of ground-truth activated nodes in $V(\mathcal{A}[t_0, t_i])$. Results shown in Figure 7 indicate that CulT gives high quality solution during the whole time interval, while the performance of Baseline degrades significantly with time.

5.7 Scalability

Last, but not least, we evaluate the scalability of CulT. We conducted these experiments on a 3.30GHz Intel Xeon machine with 16GB of memory.

First, we consider running time with respect to the number of interactions E in the temporal network. We construct a set E of a required length by increasing δ on the *Facebook* dataset. We show the results in the left-most plot of Figure 8. As the plot shows, CulT scales well with $|E|$.

In the center plot of Figure 8 we show the running time on the *Facebook* dataset, where we vary the number of activated nodes up till all nodes in the network are included. We see again a graceful increase in running time, almost linear.

Third, we investigate the applicability of CulT in a streaming scenario: we therefore report the running time *per update*, as the number of newly-arrived interactions increases. To this end we use the *Facebook* data, and vary the δ parameter to create a sequence of 3 000 interactions. We run CulT on the first 1000 interactions, and use the rest to test the update times. The total update time consists of two components, 1) the time needed to update the global shortest paths, and 2) the time needed for performing binary search. As can be seen in the right-most plot of Figure 8, the time for binary search remains constant regardless of the size of the batch of new interactions. On the other hand, as expected, the time needed for path updates increases with the batch size. Note, however, that even for larger batch sizes the total update time is less than a second.

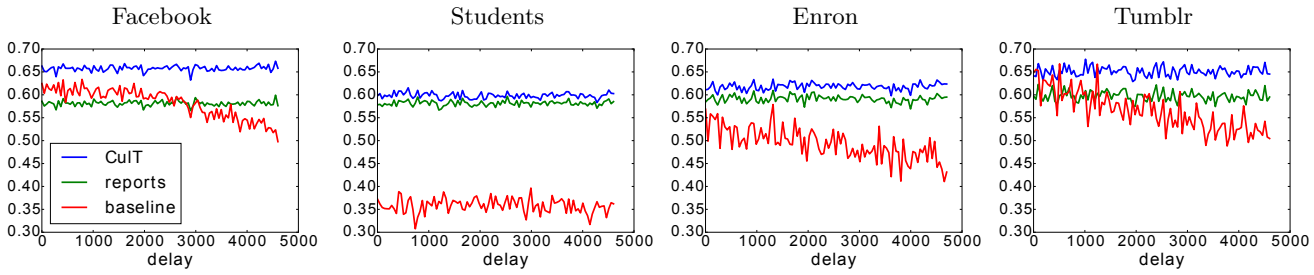

Figure 5: Effect of delay θ, between actual moment of infection and reporting, on the quality of reconstruction. Quality as measured by *MCC* for four datasets, with the *SI* model used to simulate the propagation.

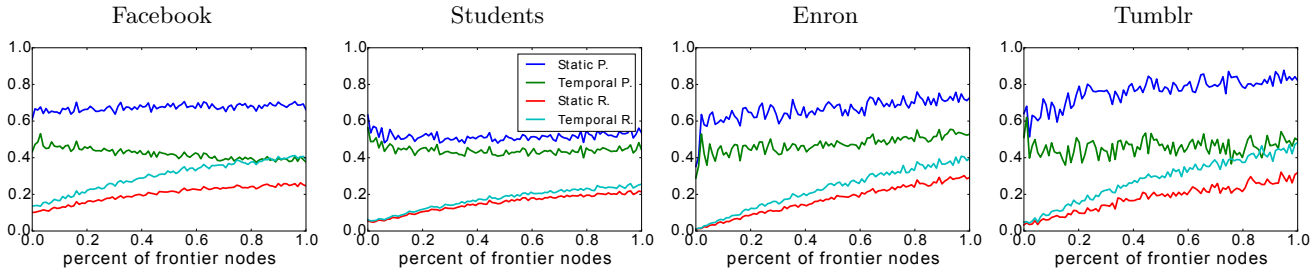

Figure 6: Precision/recall of reconstructed temporal and static infection orders per fraction of frontier nodes.

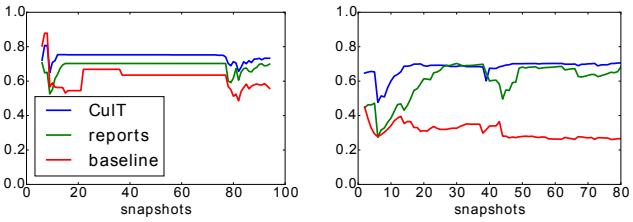

Figure 7: *MCC* on Flixster (left) and Enron (right).

6. RELATED WORK

Although diffusion processes have been widely studied in general, the problem of 'reverse engineering' an epidemic has received relatively little attention. Shah and Zaman [14] formalized the notion of *rumor-centrality* to identify the single source node of an epidemic under the SI model, and gave an optimal algorithm for *d-regular trees*, whereas Chen et al. [2] give a cubic-time approach. Prakash et al. [11] studied recovering multiple seed nodes under the SI model by MDL, while Lappas et al. [7] study the problem of identifying k seed nodes, or *effectors* of a partially activated network, which is assumed to be in steady-state under the IC (Independent-Cascade) model. Feizi et al. [4] and Sefer et al. [13] do the same for multiple snapshots, where the former consider SI, and the latter do so for the SEIRS model. All assume *complete* graphs and *noise-free* snapshots.

Correcting for the effects of missing data in cascades has not seen much attention. Sadikov et al. [12] aim to correct for sampling in broad statistical terms (like recovering the average size and depth of cascades) assuming a modified cascade model (k-trees). Farajtabar et al. [3] consider identifying a single seed given multiple partially observed cascades, assuming the SI model.

Closest to our work are Sundareisan et al. [15], who simultaneously find the starting points of the epidemic and missing infections given one sampled snapshot, assuming the SI model. In contrast, our paper addresses the general problem of finding missing nodes, given several noisy snapshots (possibly at different times), without assuming any model.

The term "dynamic graphs" is typically used to refer to the model where edges are added or deleted in a graph. In the dynamic-graph setting, once an edge is inserted in the graph it stays "alive" until the current time or until it is deleted. For example, this setting is used to model individuals establishing friendship connections in social networks.

Our model, on the contrary, intends to capture the continuous interaction between individuals. In this model, which we refer to as "temporal network," each edge has an associated time-stamp recording an interaction at that point. The temporal-network model is more recent than the dynamic-graph model. Two extensive surveys are provided by Holme and Saramäki [5] and Michail [9].

We refer the reader to the paper of Masuda and Holme [8] for a thorough survey on existing work related to epidemics in temporal networks. To the best of our knowledge the problem of reconstructing activity propagation and tracing back contagions has not been studied before in this context. From the methodology point-of-view, most related to our work is [6] where they use the directed Steiner-tree algorithm of Charikar et al. [1] in order to find the minimum spanning tree of a temporal network.

7. DISCUSSION

As should be clear by our discussion so far, CulT is designed to reconstruct an epidemic, or an activity propagation, as it is likely to have happened in the *past*. That is, it

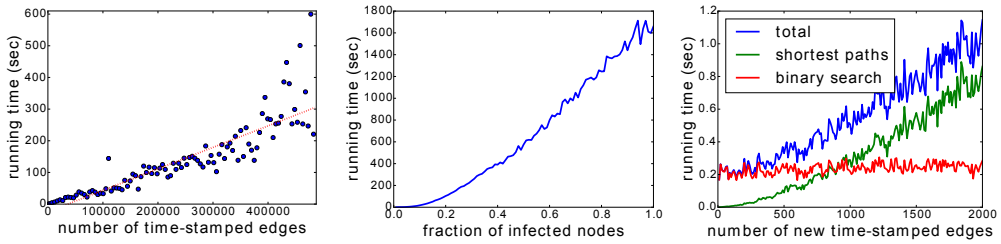

Figure 8: Running time

is not trying to make any predictions about the *future*. This is an important point that can help to put in better perspective the performance of CulT, as reported by MCC scores, in several of the experiments presented in the previous section. In particular, note that the ground-truth infected set may contain several nodes that are *downstream* from nodes in the reports set $\mathcal{R}$ (here we call a node downstream if there is no temporal path from it to any node in $\mathcal{R}$). Accordingly, CulT has no incentive to generate trees that will cover downstream nodes, as such trees will have additional cost and will not contribute in covering any node in $\mathcal{R}$.

For the sake of simplicity we have not removed downstream nodes from our ground-truth sets, and thus, the MCC performance of CulT is conservative. However, we would like to point out that detecting infected downstream nodes is a prediction task, not a reconstruction task. Moreover, it is a task that may require assuming a propagation model, and thus, a problem that we do not address here.

8. CONCLUSION

We consider the problem of reconstructing an epidemic over time. The novelty of our approach relies on the fact that we explicitly take into account the exact time that nodes interact, which leads to more accurate reconstructions. Additionally, our method requires only a small sample of nodes reported as infected, and it does not make any assumption regarding the underlying propagation model. We show how to map the reconstruction task into the classic directed Steiner-tree problem, and apply known approximation algorithms. We also present a new technique that significantly improves the running time of the approximation algorithm, and makes it applicable to online settings. We demonstrate the efficacy of CulT through multiple experiments on diverse datasets.

Acknowledgements

Jilles Vreeken is supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government. Aristides Gionis is supported by the Academy of Finland project "Nestor" (286211) and the EC H2020 RIA project "SoBigData" (654024). B. Aditya Prakash is supported by NSF Grant IIS-1353346, Maryland Procurement Office under contract H98230-14-C-0127, NEH Grant HG-229283-15 and a Facebook faculty gift.

9. REFERENCES

[1] M. Charikar, C. Chekuri, T.-y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[2] Z. Chen, H. Tong, and L. Ying. Full diffusion history reconstruction in networks. In *IEEE Big Data*, 2015.

[3] M. Farajtabar, M. Gomez-Rodriguez, M. Zamani, N. Du, H. Zha, and L. Song. Back to the past: Source identification in diffusion networks from partially observed cascades. In *AISTATS*, 2015.

[4] S. Feizi, K. Duffy, M. Kellis, and M. Medard. Network infusion to infer information sources in networks. In *RECOMB*, 2014.

[5] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[6] S. Huang, A. W.-C. Fu, and R. Liu. Minimum spanning trees in temporal graphs. In *SIGMOD*, 2015.

[7] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding effectors in social networks. In *KDD*, 2010.

[8] N. Masuda and P. Holme. Predicting and controlling infectious disease epidemics using temporal networks. *F1000 prime reports*, 5:6, 2013.

[9] O. Michail. An introduction to temporal graphs: An algorithmic perspective. In *Algorithms, Probability, Networks, and Games*, pages 308–343. Springer, 2015.

[10] B. A. Prakash, D. Chakrabarti, M. Faloutsos, N. Valler, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In *ICDM*, 2011.

[11] B. A. Prakash, J. Vreeken, and C. Faloutsos. Spotting culprits in epidemics: How many and which ones? In *ICDM*. IEEE, 2012.

[12] E. Sadikov, M. Medina, J. Leskovec, and H. Garcia-Molina. Correcting for missing data in information cascades. In *WSDM*. ACM, 2011.

[13] E. Sefer and C. Kingsford. Diffusion archaeology for diffusion progression history reconstruction. In *ICDM*, 2014.

[14] D. Shah and T. Zaman. Rumors in a network: Who's the culprit? *IEEE TIT*, 57(8):5163–5181, 2011.

[15] S. Sundareisan, J. Vreeken, and B. A. Prakash. Hidden hazards: Finding missing nodes in large graph epidemics. In *SDM*. SIAM, 2015.

[16] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, 2009.

[17] Y. Zhang and B. A. Prakash. Scalable vaccine distribution in large graphs given uncertain data. In *CIKM*, 2014.