Condensing Temporal Networks using Propagation

Bijaya Adhikari[*] Yao Zhang[*] Aditya Bharadwaj[*] B. Aditya Prakash[*]

Abstract

Modern networks are very large in size and also evolve with time. As their size grows, the complexity of performing network analysis grows as well. Getting a smaller representation of a temporal network with similar properties will help in various data mining tasks. In this paper, we study the novel problem of getting a smaller diffusion-equivalent representation of a set of time-evolving networks.

We first formulate a well-founded and general temporal-network condensation problem based on the so-called system-matrix of the network. We then propose NetCondense, a scalable and effective algorithm which solves this problem using careful transformations in sub-quadratic running time, and linear space complexities. Our extensive experiments show that we can reduce the size of large real temporal networks (from multiple domains such as social, co-authorship and email) significantly without much loss of information. We also show the wide-applicability of NetCondense by leveraging it for several tasks: for example, we use it to understand, explore and visualize the original datasets and to also speed-up algorithms for the influence-maximization problem on temporal networks.

1 Introduction

Given a large time-varying network, can we get a smaller, nearly "equivalent" one? Networks are a common abstraction for many different problems in various domains. Further, propagation-based processes are very useful in modeling multiple situations of interest in real-life such as word-of-mouth viral marketing, epidemics like flu, malware spreading, information diffusion and more. Understanding the propagation process can help in eventually managing and controlling it for our benefit, like designing effective immunization policies. However, the large size of today's networks, makes it very hard to analyze them. It is even more challenging considering that such networks evolve over time. Indeed, typical mining algorithms on dynamic networks are very slow.

One way to handle the scale is to get a summary: the idea is that the (smaller) summary can be analyzed instead of the original larger network. While summarization (and related problems) on static networks has

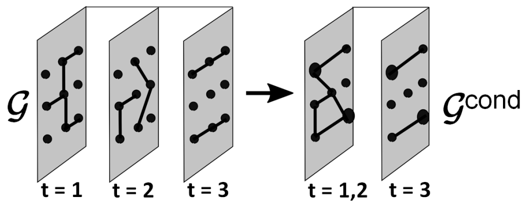


Figure 1: Condensing a Temporal Network

been recently studied, surprisingly getting a smaller representation of a temporal network has not received much attention (see related work). Since the size of temporal networks are order of magnitude higher than static networks, their succinct representation is important from a data compression viewpoint too. In this paper, we study the problem of 'condensing' a temporal network to get one *smaller in size* which is nearly 'equivalent' with regards to propagation. Such a condensed network can be very helpful in downstream data mining tasks, such as 'sense-making', influence maximization, immunization and so on. Our contributions are:

- *Problem formulation:* Using spectral characterization of propagation processes, we formulate a novel and general Temporal Network Condensation problem.
- *Efficient Algorithm:* We design careful transformations and reductions to develop an effective, near-linear time algorithm NetCondense which is also easily parallelizable. It merges unimportant node and time-pairs to quickly shrink the network without much loss of information.
- *Extensive Experiments:* Finally, we conduct multiple experiments over large diverse real datasets to show correctness, scalability, and utility of our algorithm and condensation in several tasks e.g. we show speed-ups of *48 x* in influence maximization over dynamic networks.

The rest of the paper is organized in the usual way. We omit proofs due to lack of space.

2 Preliminaries

We give some preliminaries next. Notations used and their descriptions are summarized in Table 1.

Temporal Networks: We focus on the analysis of dynamic graphs as a series of individual snapshots. In this paper, we consider directed, weighted graphs

[*]Department of Computer Science, Virginia Tech. Email: {bijaya, yaozhang, adb, badityap}@cs.vt.edu.

Table 1: Summary of symbols and descriptions

Symbol	Description
$\mathcal{G}$	Temporal Network
$\mathcal{G}^{\text{cond}}$	Condensed Temporal Network
G_i, $\mathbf{A}_i$	ith graph of $\mathcal{G}$ and adjacency matrix
$w_i(a, b)$	Edge-weight between nodes a and b in time-stamp i
V; E	Node-set; Edge-set
α_N	Target fraction for nodes
α_T	Target fraction for time-stamps
T	# of timestamps in Temporal Network
$F_{\mathcal{G}}$	Flattened Network of $\mathcal{G}$
$X_{\mathcal{G}}$	Average Flattened Network of $\mathcal{G}$
$\mathbf{S}_{\mathcal{G}}$	The system matrix of $\mathcal{G}$
$\mathbf{F}_{\mathcal{G}}$; $\mathbf{X}_{\mathcal{G}}$	The adjacency matrix of $F_{\mathcal{G}}$; $X_{\mathcal{G}}$
$\lambda_{\mathbf{S}}$	Largest eigenvalue of $\mathbf{S}_{\mathcal{G}}$
$\lambda_{\mathbf{F}}$; $\lambda_{\mathbf{X}}$	Largest eigenvalue of $\mathbf{F}_{\mathcal{G}}$; $\mathbf{X}_{\mathcal{G}}$
$\mathbf{A}$	Matrix (Bold capital letter)
$\mathbf{u}$, $\mathbf{v}$	Column Vectors (Bold small letter)

$G = (V, E, W)$ where V is the set of nodes, E is the set of edges and W is the set of associated edge-weights $w(a, b) \in [0, 1]$. A *temporal network* $\mathcal{G}$ is a sequence of T graphs, i.e., $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$, such that the graph at time-stamp i is $G_i = (V, E_i, W_i)$. WLOG, we assume every G_i in $\mathcal{G}$ has the same node-set V (as otherwise, if we have G_i with different V_i, just define $V = \cup_{i=1}^{T} V_i$). Our ideas can, however, be easily generalized to other types of dynamic graphs.

Propagation models: We primarily base our discussion on two fundamental discrete-time propagation/diffusion models: the SI [3] and IC models [10]. The SI model is a basic epidemiological model where each node can either be in 'Susceptible' or 'Infected' state. In a static graph, at each time-step, a node infected/active with the virus/contagion can infect each of its 'susceptible' (healthy) neighbors independently with probability $w(a, b)$. Once the node is infected, it stays infected. SI is a special case of the general 'flu-like' SIS model, as the 'curing rate' (of recovering from the infected state) δ in SI is 0 while in SIS $\delta \in [0, 1)$. In the popular IC (Independent Cascade) model nodes get exactly one chance to infect their healthy neighbors with probability $w(a, b)$; it is a special case of the general 'mumps-like' SIR model, where nodes in 'Removed' state do not get re-infected, with $\delta = 1$.

We consider generalizations of these models to temporal networks [17], where an infected node can only infect its susceptible 'current' neighbors (as given by $\mathcal{G}$). Note that models on static graphs are special cases of those on temporal networks (with all $G_i \in \mathcal{G}$ identical).

3 Our Problem Formulation

Real temporal networks are usually gigantic in size. However, their skewed nature (in terms of various distributions like degree, triangles etc.) implies the existence of many nodes/edges which are not important in propagation. Similarly, as changes are typically gradual, most of adjacent time-stamps are not drastically different. There may also be time-periods with sparse connectivities which will not contribute much to propagation. Overall, these observations intuitively imply that it should be possible to get a smaller 'condensed' representation of $\mathcal{G}$ while preserving its diffusive characteristics; which is our task.

It is natural to condense as a result of only local 'merge' operations on node-pairs and time-pairs of $\mathcal{G}$—such that each application of an operation maintains the propagation property and shrinks $\mathcal{G}$. This will also ensure that successive applications of these operations 'summarize' $\mathcal{G}$ in a multi-step hierarchical fashion.

More specifically, merging a node-pair $\{a, b\}$ will merge nodes a and b into a new *super-node* say c, *in all* G_i in $\mathcal{G}$. Merging a time-pair $\{i, j\}$ will merge graphs G_i and G_j to create a new *super-time*, k, and associated graph G_k. However, allowing merge operations on every possible node-pair and time-pair results in loss of interpretability of the result. For example, it is meaningless to merge two nodes who belong to completely different communities or merge times which are five time-stamps apart. Therefore, we have to limit the merge operations in a natural and well-defined way. This also ensures that the resulting summary is useful for downstream applications. We allow a single node-merge only on node pairs $\{a, b\}$ such that $\{a, b\} \in E_i$ for *at least* one G_i, i.e. $\{a, b\}$ is in the unweighted 'union graph' $U_{\mathcal{G}}(V, E_u = \cup_i E_i)$. Similarly, we restrict a single time-merge to only adjacent time-stamps. Note that we can still apply multiple successive merges to merge multiple node-pairs/time-pairs. Our general problem is:

Informal Problem 1. *Given a temporal network* $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ *with* $G_i = (V, E_i, W_i)$ *and target fractions* $\alpha_N \in (0, 1]$ *and* $\alpha_T \in (0, 1]$*, find a condensed temporal network* $\mathcal{G}^{\text{cond}} = \{G'_1, G'_2, \ldots, G'_{T'}\}$ *with* $G'_i = (V', E'_i, W'_i)$ *by repeatedly applying "local" merge operations on node-pairs and time-pairs such that (a)* $|V'| = (1 - \alpha_N)|V|$*; (b)* $T' = (1 - \alpha_T)T$*; and (c)* $\mathcal{G}^{\text{cond}}$ *approximates* $\mathcal{G}$ *w.r.t. propagation-based properties.*

3.1 Formulation framework

Formalizing Informal Problem 1 is challenging as we need to tackle following two research questions: (Q1) Characterize and quantify the propagation-based property of a temporal network $\mathcal{G}$; (Q2) Define "local" merge operations.

In general, Q1 is difficult as the characterization should be scalable and concise. For Q2, the merges are local operations, and so intuitively they should be defined so that any local diffusive changes caused by them is minimum. Using Q1 and Q2, we can formulate Informal Problem 1 as an optimization problem where the search space is all possible temporal networks with the desired size and which can be constructed via some sequence of repeated merges from $\mathcal{G}$.

3.2 Q1: Propagation-based property One possible naive answer is to run some diffusion model on $\mathcal{G}$ and $\mathcal{G}^{\text{cond}}$ and see if the propagation is similar; but this is too expensive. Therefore, we want to find a tractable concise metric that can characterize and quantify propagation on a temporal network.

A major metric of interest in propagation on networks is the epidemic threshold which indicates whether the virus/contagion will quickly spread throughout the network (and cause an 'epidemic') or not, regardless of the initial conditions. Past works [6, 16] have studied epidemic thresholds for various epidemic models on static graphs. Recently, [17] show that in context of temporal networks and the SIS model, the threshold depends on the largest eigenvalue λ of the so-called system matrix of $\mathcal{G}$: an epidemic will not happen in $\mathcal{G}$ if $\lambda < 1$. The result in [17] was only for undirected graphs; however it can be easily extended to weighted directed $\mathcal{G}$ with a strongly connected union graph $U_{\mathcal{G}}$ (which just implies that in principle any node can infect any other node via a path, ignoring time; as otherwise we can just examine each connected component separately).

Definition 1. ***System Matrix:*** *For the SI model, the system matrix $\mathbf{S}_{\mathcal{G}}$ of a temporal network $\mathcal{G} = \{G_1, G_2, ..., G_T\}$ is defined as $\mathbf{S}_{\mathcal{G}} = \prod_{i=1}^{T}(\mathbf{I} + \mathbf{A}_i)$.*

where $\mathbf{A}_t$ is the weighted adjacency matrix of G_t. For the SI model, the rate of infection is governed by $\lambda_{\mathbf{S}}$, the largest eigenvalue of $\mathbf{S}_{\mathcal{G}}$. Preserving $\lambda_{\mathbf{S}}$ while condensing $\mathcal{G}$ to $\mathcal{G}^{\text{cond}}$ will imply that the rate of virus spreading out in $\mathcal{G}$ and $\mathcal{G}^{\text{cond}}$ will be preserved too. Therefore $\lambda_{\mathbf{S}}$ is a well motivated and meaningful metric to preserve during condensation.

3.3 Q2: Merge Definitions We define two operators: $\mu(\mathcal{G}, i, j)$ merges a time-pair $\{i, j\}$ in $\mathcal{G}$ to a super-time k in $\mathcal{G}^{\text{cond}}$; while $\zeta(\mathcal{G}, a, b)$ merges node-pair $\{a, b\}$ in all $G_i \in \mathcal{G}$ and results in a super-node c in $\mathcal{G}^{\text{cond}}$.

As stated earlier, we want to condense $\mathcal{G}$ by successive applications of μ and ζ. We also want them to preserve local changes in diffusion in the locality of merge operands. At the node level, the level where local merge operations are performed, the diffusion process is best characterized by the probability of infection. Hence, working from first principles, we design these operations to maintain the probabilities of infection before and after the merges in the 'locality of change' without worrying about the system matrix. For $\mu(\mathcal{G}, i, j)$, the 'locality of change' is G_i, G_j and the new G_k. Whereas, for $\zeta(\mathcal{G}, a, b)$, the 'locality of change' is the neighborhood of $\{a, b\}$ in all $G_i \in \mathcal{G}$.

Time-pair Merge: Consider a merge $\mu(\mathcal{G}, i, j)$ between consecutive times i and j. Consider any edge (a, b) in G_i and G_j (note if $(a, b) \notin E_i$, then $w_i(a, b) = 0$) and assume that node a is infected and node b is susceptible in G_i (illustrated in Figure 2 (a)). Now, node a can infect node b in i via an edge in G_i, or in j via an edge in G_j. We want to maintain the local effects of propagation via the merged time-stamp G_k. Hence we need to readjust edge-weights in G_k such that it captures the probability a infects b in $\mathcal{G}$ (in i and j).

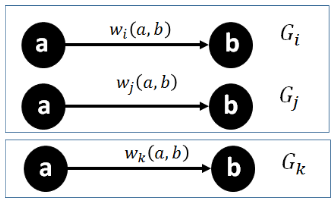
Lemma 3.1. *(Infection via i & j) Let $\Pr(a \to b | G_i, G_j)$ be the probability that a infects b in $\mathcal{G}$ in either time i or j, if it is infected in G_i. Then $\Pr(a \to b | G_i, G_j) = [w_i(a, b) + w_j(a, b)]$, upto a first order approximation.*

Lemma 3.1 suggests that the condensed time-stamp k, after merging a time-pair $\{i, j\}$ should be $\mathbf{A}_k = \mathbf{A}_i + \mathbf{A}_j$. However, consider a $\mathcal{G}$ such that all G_i in $\mathcal{G}$ are the same. This is effectively a static network: hence the time-merges should give the network G_i rather than TG_i. This discrepancy arises because for any single time-merge, as we reduce 'T' from 2 to 1, to maintain the final spread of the model, we have to increase the infectivity along each edge by a factor of 2 (intuitively speeding up the model [8]). Hence, the condensed network at time k should be $\mathbf{A}_k = \frac{\mathbf{A}_i + \mathbf{A}_j}{2}$ instead; while for the SI model, the rate of infection should be doubled for time k in the system matrix. Motivated by these considerations, we define a time-stamp merge as follows:

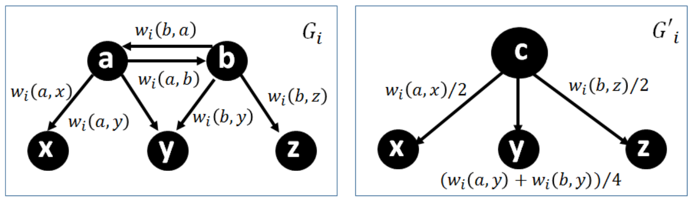
Definition 2. ***Time-Pair Merge*** *$\mu(\mathcal{G}, i, j)$. The merge operator $\mu(\mathcal{G}, i, j)$ returns a new time-stamp k with weighted adjacency matrix $\mathbf{A}_k = \frac{\mathbf{A}_i + \mathbf{A}_j}{2}$.*

Node-pair Merge: Similarly, in $\zeta(\mathcal{G}, a, b)$ we need to adjust the weights of the edges to maintain the local effects of diffusion between a and b and their neighbors. Note that when we merge two nodes, we need to merge them in all $G_i \in \mathcal{G}$.

Consider any time i. Suppose we merge $\{a, b\}$ in G_i to form *super-node* c in G_i' (note that $G_i' \in \mathcal{G}^{\text{cond}}$). Consider a node x such that $\{a, b\}$ and $\{a, x\}$ are neighbors in G_i (illustrated in Figure 2 (b)). When c is infected in G_i', it is intuitive to imply that *either* node a or b is infected in G_i uniformly at random. Hence we need to update the edge-weight from c to x in G_i', such that the new edge-weight is able to reflect the probability that either node a or b infects x in G_i.



(a) Time merge of a single edge



(b) Node Merge in a single time

Figure 2: (a) Example of merge operation on a single edge (a, b) when time-pair $\{i, j\}$ is merged to form super-time k. (b) Example of node-pair $\{a, b\}$ being merged in a single time i to form super-node c.

LEMMA 3.2. *(Probability of infecting out-neighbors) If either node a or node b is infected in G_i and they are merged to form a super-node c, then the first order approximation of probability of node c infecting its out-neighbors is given by:*

$$\Pr(c \to z|G_i) = \begin{cases} \dfrac{w_i(a,z)}{2} & \forall z \in Nb_i^o(a) \backslash Nb_i^o(b) \\ \dfrac{w_i(b,z)}{2} & \forall z \in Nb_i^o(b) \backslash Nb_i^o(a) \\ \dfrac{w_i(a,z)+w_i(b,z)}{4} & \forall z \in Nb_i^o(a) \cap Nb_i^o(b) \end{cases}$$

where, $Nb_i^o(v)$ is the set of out-neighbors of node v in time-stamp i. We can write down the corresponding probability $\Pr(z \to c|G_i)$ (for getting infected by in-neighbors) similarly. Motivated by Lemma 3.2, we define node-pair merge as:

DEFINITION 3. ***Node-Pair merge*** *$\zeta(\mathcal{G}, a, b)$. The merge operator $\zeta(\mathcal{G}, a, b)$ merges a and b to form a new super-node c in all $G_i \in \mathcal{G}$, s.t. $w_i(c, z) = \Pr(c \to z|G_i)$ and $w_i(z, c) = \Pr(z \to c|G_i)$.*

3.4 Problem Definition

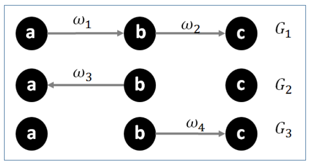
We can now formally define our problem.

PROBLEM 1. *(TEMPORAL NETWORK CONDENSATION Problem (`TNC`)) Given a temporal network $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ with strongly connected $U_\mathcal{G}$, $\alpha_N \in (0, 1]$ and $\alpha_T \in (0, 1]$, find a condensed temporal network $\mathcal{G}^{\text{cond}} = \{G'_1, G'_2, \ldots, G'_{T'}\}$ with $G'_i = (V', E'_i, W'_i)$ by repeated applications of $\mu(\mathcal{G}, \cdot, \cdot)$ and $\zeta(\mathcal{G}, \cdot, \cdot)$, such that $|V'| = (1 - \alpha_N)|V|$; $T' = (1 - \alpha_T)T$; and $\mathcal{G}^{\text{cond}}$ minimizes $|\lambda_S - \lambda_S^{\text{cond}}|$.*

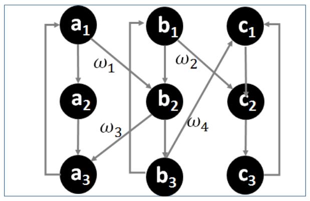
Problem 1 naturally contains the GCP coarsening problem for a static network [18] (which aims to preserve the largest eigenvalue of the adjacency matrix) as a special case: when $\mathcal{G} = \{G\}$. GCP itself is a challenging problem as it is related to immunization problems. Hence, Problem 1 is intuitively even more challenging.

4 Our Proposed Method

The naive algorithm is combinatorial. Even the greedy method which computes the next best merge operands will be $O(\alpha_N \cdot V^6)$, even without time-pair merges. In fact, even computing $\mathbf{S}_\mathcal{G}$ is inherently non-trivial due to matrix multiplications. It does not scale well for large temporal networks because $\mathbf{S}_\mathcal{G}$ gets denser as the number of time-stamps in $\mathcal{G}$ increases. Moreover, since $\mathbf{S}_\mathcal{G}$ is a dense matrix of size $|V|$ by $|V|$, it does not even fit in the main memory for large networks. Even if there was an algorithm for Problem 1 that could bypass computing $\mathbf{S}_\mathcal{G}$, $\lambda_\mathbf{S}$ still has to be computed to measure success. Therefore, even just measuring success for Problem 1, as is, seems hard.



(a) Temporal Network



(b) Flattened Network

Figure 3: (a) $\mathcal{G}$, and (b) corresponding $F_\mathcal{G}$.

4.1 Main idea

To solve the numerical and computational issues, our idea is to find an alternate representation of $\mathcal{G}$ such that the new representation has the same diffusive properties and avoids the issues of $\mathbf{S}_\mathcal{G}$. Then we develop an efficient sub-quadratic algorithm.

Our main idea is to look for a *static* network that is similar to $\mathcal{G}$ with respect to propagation. We do this in two steps. First we show how to construct a static flattened network $F_\mathcal{G}$, and show that it has similar diffusive properties as $\mathcal{G}$. We also show that eigenvalues of $\mathbf{S}_\mathcal{G}$ and the adjacency matrix $\mathbf{F}_\mathcal{G}$ of $F_\mathcal{G}$ are precisely related. Due to this, computing eigenvalues of $\mathbf{F}_\mathcal{G}$ too is difficult. Then in the second step, we derive a network from $F_\mathcal{G}$ whose largest eigenvalue is easier to compute and related to the largest eigenvalue of $\mathbf{F}_\mathcal{G}$. Using it we propose a new related problem, and solve it efficiently.

4.2 Step 1: An Alternate Static View

Our approach for getting a static version is to expand $\mathcal{G}$ and create *layers* of nodes, such that edges in $\mathcal{G}$ are captured by edges *between* the nodes in adjacent layers (see Figure 3). We call this the "flattened network" $F_\mathcal{G}$.

Definition 4. ***Flattened network.*** *$F_\mathcal{G}$ for $\mathcal{G}$ is defined as follows:*

- ***Layers:*** *$F_\mathcal{G}$ consists of $1,...,T$ layers corresponding to T time-stamps in $\mathcal{G}$.*
- ***Nodes:*** *Each layer i has $|V|$ nodes (so $F_\mathcal{G}$ has $T|V|$ nodes overall). Node a in the temporal network $\mathcal{G}$ at time i is represented as a_i in layer i of $F_\mathcal{G}$.*
- ***Edges:*** *At each layer i, each node a_i has a direct edge to $a_{(i+1) \bmod T}$ in layer $(i+1) \bmod T$ with edge-weight 1. And for each time-stamp G_i in the temporal network $\mathcal{G}$, if there is a directed edge (a,b), then in $F_\mathcal{G}$, we add a direct edge from node a_i to node $b_{(i+1) \bmod T}$ with weight $w_i(a,b)$.*

For the relationship between $\mathcal{G}$ and $F_\mathcal{G}$, consider the SI model running on $\mathcal{G}$ (Figure 3 (a)). Say node a is infected in G_1, which also means node a_1 is infected in $F_\mathcal{G}$ (Figure 3 (b)). Assume a infects b in G_1. So in the beginning of G_2, a and b are infected. Correspondingly in $F_\mathcal{G}$ node a_1 infects nodes a_2 and b_2. Now in G_2, no further infection occurs. So the same nodes a and b are infected in G_3. However, in $F_\mathcal{G}$ infection occurs between layers 2 and 3, which means a_2 infects a_3 and b_2 infects b_3. Propagation in $F_\mathcal{G}$ is different than in $\mathcal{G}$ as each 'time-stamped' node gets exactly one chance to infect others. Note that the propagation model on $F_\mathcal{G}$ we just described is the popular IC model. Hence, running the SI model in $\mathcal{G}$ should be "equivalent" to running the *IC model* in $F_\mathcal{G}$ in some sense.

We formalize this next. Assume we have the SI model on $\mathcal{G}$ and the IC model on $F_\mathcal{G}$ starting from the same node-set of size $I(0)$. Let $I_{SI}^{\mathcal{G}}(t)$ be the *expected* number of infected nodes at the end of time t. Similarly, let $I_{IC}^{F_\mathcal{G}}(T)$ be the expected number of infected nodes under the IC model till end of time T in $F_\mathcal{G}$. Note that $I_{IC}^{F_\mathcal{G}}(0) = I_{SI}^{F_\mathcal{G}}(0) = I(0)$. Then:

Lemma 4.1. *(Equivalence of propagation in $\mathcal{G}$ and $F_\mathcal{G}$) We have $\sum_{t=1}^{T} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F_\mathcal{G}}(T)$.*

That is, the cumulative expected infections for the SI model on $\mathcal{G}$ is the same as the infections after T for the IC model in $F_\mathcal{G}$. This suggests that the largest eigenvalues of $\mathbf{S}_\mathcal{G}$ and $\mathbf{F}_\mathcal{G}$ are closely related. Actually, we can prove a stronger statement that the spectra of $F_\mathcal{G}$ and $\mathcal{G}$ are closely related (Lemma 4.2).

Lemma 4.2. *(Eigen-equivalence of $\mathbf{S}_\mathcal{G}$ and $\mathbf{F}_\mathcal{G}$) We have $(\lambda_{\boldsymbol{F}})^T = \lambda_{\boldsymbol{S}}$. Furthermore, λ is an eigenvalue of $\mathbf{F}_\mathcal{G}$, iff λ^T is an eigenvalue of $\mathbf{S}_\mathcal{G}$.*

Lemma 4.2 implies that preserving $\lambda_\mathbf{S}$ in $\mathcal{G}$ is equivalent to preserving $\lambda_\mathbf{F}$ in $F_\mathcal{G}$. Therefore, Problem 1 can be re-written in terms of $\lambda_\mathbf{F}$ (of a static network) instead of $\lambda_\mathbf{S}$ (of a temporal one).

4.3 Step 2: A Well Conditioned Network

However $\lambda_\mathbf{F}$ is problematic too. The difficulty in computing $\lambda_\mathbf{F}$ arises because $\mathbf{F}_\mathcal{G}$ is ill-conditioned. So modern packages take many iterations and the result may be imprecise. Intuitively, it is easy to understand that computing $\lambda_\mathbf{F}$ is difficult: as if it were not, computing $\lambda_\mathbf{S}$ itself would have been easy (just compute $\lambda_\mathbf{F}$ and raise it to the T-th power).

So we create a new static network that has a close relation with $F_\mathcal{G}$ and whose adjacency matrix is well-conditioned. To this end, we look at the *average* flattened network, $X_\mathcal{G}$, whose adjacency matrix is defined as $\mathbf{X}_\mathcal{G} = \frac{\mathbf{F}_\mathcal{G} + \mathbf{F}_\mathcal{G}'}{2}$, where $\mathbf{F}_\mathcal{G}'$ is the transpose of $\mathbf{F}_\mathcal{G}$. It is easy to see that trace of $\mathbf{X}_\mathcal{G}$ and $\mathbf{F}_\mathcal{G}$ are equal, which means that the sum of eigenvalues of $\mathbf{X}_\mathcal{G}$ and $\mathbf{F}_\mathcal{G}$ are equal. Moreover, we have the following:

Lemma 4.3. *(Eigenvalue relationship of $\mathbf{F}_\mathcal{G}$ and $\mathbf{X}_\mathcal{G}$) The largest eigenvalue of $\mathbf{F}_\mathcal{G}$, $\lambda_{\boldsymbol{F}}$, and the largest eigenvalue of $\mathbf{X}_\mathcal{G}$, $\lambda_{\boldsymbol{X}}$, are related as $\lambda_{\boldsymbol{F}} \leq \lambda_{\boldsymbol{X}}$.*

Note that if $\lambda_\mathbf{X} < 1$, then $\lambda_\mathbf{F} < 1$. Moreover, if $\lambda_\mathbf{F} < 1$ then $\lambda_\mathbf{S} < 1$. Hence if there is no epidemic in $X_\mathcal{G}$, then there is no epidemic in $F_\mathcal{G}$ as well, which implies that the rate of spread in $\mathcal{G}$ is low. Hence, $X_\mathcal{G}$ is a good proxy static network for $F_\mathcal{G}$ and $\mathcal{G}$ and $\lambda_\mathbf{X}$ is a well-motivated quantity to preserve. Also we need only weak-connectedness of $U_\mathcal{G}$ for $\lambda_\mathbf{X}$ (and corresponding eigenvectors) to be real and positive (by the Perron-Frobenius theorem). Furthermore, $\mathbf{X}_\mathcal{G}$ is free of the problems faced by $\mathbf{F}_\mathcal{G}$ and $\mathbf{S}_\mathcal{G}$: it is well-conditioned and its eigenvalue can be efficiently computed.

New problem: Considering all of the above, we reformulate Problem 1 in terms of $\lambda_\mathbf{X}$. Since $\mathcal{G}$ and $X_\mathcal{G}$ are closely related networks, the merge definitions on $X_\mathcal{G}$ can be easily extended from those on $\mathcal{G}$.

Problem 2. *Given $\mathcal{G}$ with weakly connected $U_\mathcal{G}$ over V, α_N and α_T, find $\mathcal{G}^{\text{cond}}$ by repeated application of $\mu(X_\mathcal{G},.,.)$ and $\zeta(X_\mathcal{G},.,.)$ such that $|V'| = (1-\alpha_N)|V|$; $T' = (1-\alpha_T)T$; and $\mathcal{G}^{\text{cond}}$ minimizes $|\lambda_{\boldsymbol{X}} - \lambda_{\boldsymbol{X}}^{\text{cond}}|$.*

4.4 NetCondense

In this section, we propose a fast greedy algorithm for Problem 2 called NetCondense, which only takes sub-quadratic time in the size of the input. Again, the obvious approach is combinatorial. Consider a greedy approach using Δ-Score.

Definition 5. ***Δ-Score.*** $\Delta_{X_\mathcal{G}}(a,b) = |\lambda_{\boldsymbol{X}} - \lambda_{\boldsymbol{X}}^{\text{cond}}|$ *where $\lambda_{\boldsymbol{X}}^{\text{cond}}$ is the largest eigenvalue of the new $\mathbf{X}_\mathcal{G}$ after merging a and b (node or time-pair).*

The greedy approach will successively choose those merge operands at each step which have the *lowest*

Δ-Score. Doing this naively will lead to quartic time (due to repeated re-computations of $\lambda_{\mathbf{X}}$ for all possible time/node-pairs). Recall that we limit time-merges to adjacent time-pairs and node-merges to node-pairs with an edge in $U_{\mathcal{G}}$. Now, computing Δ-Score simply for all edges $(a, b) \in U_{\mathcal{G}}$ is still expensive. Hence we *estimate* Δ-Score for node/time pairs instead using Matrix Perturbation Theory [23]. Let $\mathbf{u}$ and $\mathbf{v}$ be right and left eigenvector of $\mathbf{X}_{\mathcal{G}}$, corresponding to $\lambda_{\mathbf{X}}$. Let $\mathbf{u}(a_i)$ and $\mathbf{v}(a_i)$ be the right and left 'eigenscore' of node a_i in $\mathbf{X}_{\mathcal{G}}$. Then we have the following lemmas.

LEMMA 4.4. *(Δ-Score for time-pair) Let V_i = nodes in Layer i of $X_{\mathcal{G}}$. Now, for merge $\mu(X_{\mathcal{G}}, i, j)$ to form k,*

$$\Delta_{X_{\mathcal{G}}}(i,j) = \frac{-\lambda_{\boldsymbol{X}}(\sum_{i \in V_i, V_j} \eta_{(i,i)}) + \sum_{k \in V_k} \mathbf{v}(i)\mathbf{k}^{oT}\mathbf{u} + Y}{\mathbf{v}^T\mathbf{u} - \sum_{i \in V_i, V_j} \eta_{(i,i)}}$$

upto a first-order approximation, where $\eta_{(i,j)} = \mathbf{u}(i)\mathbf{v}(j)$, $Y = \sum_{i \in V_i, j \in V_j}(\eta_{(i,j)} + \eta_{(j,i)})\mathbf{X}_{\mathcal{G}}(i,j)$, and $\mathbf{k}^{oT}\mathbf{u} = \frac{1}{2}(\lambda_{\boldsymbol{X}}\mathbf{u}(i) + \lambda_{\boldsymbol{X}}\mathbf{u}(j) + \mathbf{u}(i) + \mathbf{u}(j))$.

LEMMA 4.5. *(Δ-Score for node-pair) Let $V_a = \{a_1, a_2, \ldots, a_T\} \in X_{\mathcal{G}}$ corresponding to node a in $\mathcal{G}$. For merge $\zeta(X_{\mathcal{G}}, a, b)$ to form c,*

$$\Delta_{X_{\mathcal{G}}}(a,b) = \frac{-\lambda_{\boldsymbol{X}}(\sum_{a \in V_a, V_b} \eta_{(a,a)}) + \sum_{c \in V_c} \mathbf{v}(a)\mathbf{c}^{oT}\mathbf{u} + Y}{\mathbf{v}^T\mathbf{u} - \sum_{a \in V_a, V_b} \eta_{(a,a)}}$$

upto a first-order approximation, where $\eta_{(a,b)} = \mathbf{u}(a)\mathbf{v}(b)$, $Y = \sum_{a \in V_a, b \in V_b}(\eta_{(a,b)} + \eta_{(b,a)})\mathbf{X}_{\mathcal{G}}(a,b)$, and $\mathbf{c}^{oT}\mathbf{u} = \frac{1}{2}\lambda_{\boldsymbol{X}}(\mathbf{u}(a) + \mathbf{u}(b))$.

Our algorithm NETCONDENSE works as follows: we first calculate Δ-Score for time-pairs based on Lemma 4.4. Similarly, for all edges in $U_{\mathcal{G}}$ using Lemma 4.5. Then we choose the top number of node/time-pairs based on score, and we keep merging till $\mathcal{G}^{\text{cond}}$ is of the desired size. Complete pseudo-code is in Algorithm 1.

LEMMA 4.6. NETCONDENSE *has sub-quadratic time-complexity of $O(TE_u + E\log E + \alpha_N\theta TV + \alpha_T E)$, where θ is the maximum degree in any $G_i \in G$ and linear space-complexity of $O(E + TV)$.*

Parallelizability: We can easily parallelize NETCONDENSE: once the eigenvector of $\mathbf{X}_{\mathcal{G}}$ is computed, Δ-Score for node-pairs and time-pairs (loops in Lines 3 and 5 in Algorithm 1) can be computed independent of each other in parallel. Similarly, μ and ζ operators (in Line 11) are also parallelizable.

5 Experiments

5.1 Experimental Setup We briefly describe our set-up next. All experiments are conducted using a 4 Xeon E7-4850 CPU with 512GB 1066Mhz RAM. Our code is publicly available for academic purposes[1].

[1]`http://people.cs.vt.edu/~bijaya/codes/NetCondense.zip`

Algorithm 1 NETCONDENSE

```
Require: Temporal graph $\mathcal{G}$ , $0 < \alpha_N < 1$, $0 < \alpha_T < 1$
Ensure: Temporal graph $\mathcal{G}^{\text{cond}}(V', E', T')$
 1: obtain $\mathbf{X}_{\mathcal{G}}$ using Definition 4.
 2: for every adjacent time-pairs $\{i, j\}$ do
 3:     Calculate $\Delta_{X_{\mathcal{G}}}(i, j)$ using Lemma 4.4
 4: for every node-pair $\{a, b\}$ in $U_{\mathcal{G}}$ do
 5:     Claculate $\Delta_{X_{\mathcal{G}}}(a, b)$ using Lemma 4.5
 6: sort the lists of Δ-Score for time-pairs and node-pairs
 7: $\mathcal{G}^{\text{cond}} = \mathcal{G}$
 8: while $|V'| > \alpha_N \cdot |V|$ or $T' > \alpha_T \cdot T$ do
 9:     $(x, y) \leftarrow$ node-pair or time-pair with lowest Δ-Score
10:     $\mathcal{G}^{\text{cond}} \leftarrow \mu(\mathcal{G}^{\text{cond}}, x, y)$ or $\zeta(\mathcal{G}^{\text{cond}}, x, y)$
11: return $\mathcal{G}^{\text{cond}}$
```

Table 2: Datasets Information.

Dataset	Weight	$\lvert V\rvert$	$\lvert E\rvert$	T
`WorkPlace`	Contact Hrs	92	1.5K	12 Days
`School`	Contact Hrs	182	4.2K	9 Days
`Enron`	# Emails	184	8.4K	44 Months
`Chess`	# Games	7.3K	62.4K	9 Years
`Arxiv`	# Papers	28K	**3.8M**	9 Years
`Wikipedia`	# Pages	118K	**2.1M**	10 Years
`WikiTalk`	# Messages	497K	**2.7M**	12 Years
`DBLP`	# Papers	**1.3M**	**18M**	25 Years

Datasets. We run NETCONDENSE on a variety of real datasets (Table 2) of varying sizes from different domains such as social-interactions (`WorkPlace`, `School`, `Chess`), co-authorship (`Arxiv`, `DBLP`) and communication (`Enron`, `Wikipedia`, `WikiTalk`). They include weighted and both directed and undirected networks. Edge-weights are normalized to the range $[0, 1]$.

Baselines. Though there are no direct competitors, we adapt multiple methods to use as baselines.

RANDOM: Uniformly randomly choose node-pairs and time-stamps to merge.

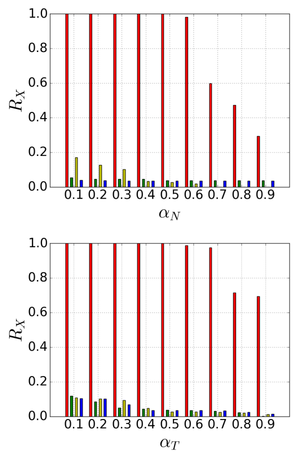
TENSOR: Here we pick merge operands based on the centrality given by tensor decomposition. $\mathcal{G}$ can be also seen as a tensor of size $|V| \times |V| \times T$. So we run PARAFAC decomposition [12] on $\mathcal{G}$ and choose the largest component to get three vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ of size $|V|$, $|V|$, and T respectively. We compute pairwise centrality measure for node-pair $\{a, b\}$ as $\mathbf{x}(a) \cdot \mathbf{y}(b)$ and for time-pair $\{i, j\}$ as $\mathbf{z}(i) \cdot \mathbf{z}(j)$ and choose the top-K least central ones.

CNTEMP: We run Coarsenet [18] (a summarization method which preserves the diffusive property of a static graph) on $U_{\mathcal{G}}$ and repeat the summary to create $\mathcal{G}^{\text{cond}}$.

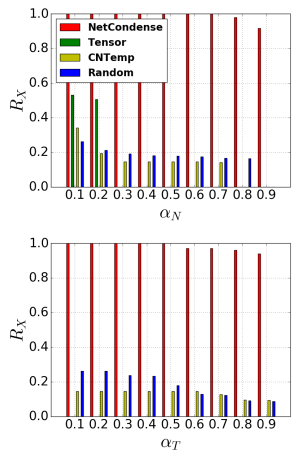
In RANDOM and TENSOR, we use our own merge definitions, hence the comparison is inherently unfair.

5.2 Perfomance of NetCondense: Effectiveness We ran all the algorithms to get $\mathcal{G}^{\text{cond}}$ for different values of α_N and α_T, and measure $R_{\mathbf{X}} = \lambda_{\mathbf{X}}^{\text{cond}}/\lambda_{\mathbf{X}}$ to judge performance for Problem 2. See Figure 4.

NetCondense is able to preserve $\lambda_{\mathbf{X}}$ excellently (upto 80% even when the the number of time-stamps and nodes are reduced by 50% and 70% respectively). On the other hand, the baselines perform much worse, and quickly degrade $\lambda_{\mathbf{X}}$. Note that Tensor does not even finish within *7 days* for DBLP for larger α_N. Random and Tensor perform poorly even though they use the same merge definitions, showcasing the importance of right merges. In case of Tensor, unexpectedly it tends to merge unimportant nodes with all nodes in their neighborhood even if they are "important"; so it is unable to preserve $\lambda_{\mathbf{X}}$. Finally CNTemp performs badly as it does not use the full temporal nature of $\mathcal{G}$.



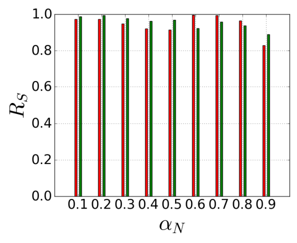
(a) WikiTalk



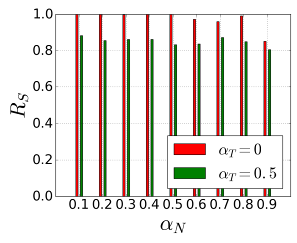
(b) DBLP

Figure 4: $R_{\mathbf{X}} = \lambda_{\mathbf{X}}^{\text{cond}}/\lambda_{\mathbf{X}}$ vs α_N (top row, $\alpha_T = 0.5$) and vs α_T (bottom row, $\alpha_N = 0.5$).

We also compare our performance for Problem 1, against an algorithm specifically designed for it. We use the simple greedy algorithm GreedySys for Problem 1 (as the brute-force is too expensive): it greedily picks top node/time merges by actually re-computing $\lambda_{\mathbf{S}}$. We can run GreedySys only for small networks due to the $\mathbf{S}_{\mathcal{G}}$ issues we mentioned before. See Figure 5 ($\lambda_{\mathbf{S}}^{\text{M}}$ is $\lambda_{\mathbf{S}}^{\text{cond}}$ obtained from method M). NetCondense does almost as well as GreedySys, due to our careful transformations and reductions.



(a) WorkPlace



(b) School

Figure 5: Plot of $R_{\mathbf{S}} = \lambda_{\mathbf{S}}^{\text{NetCondense}}/\lambda_{\mathbf{S}}^{\text{GreedySys}}$.

5.3 Application 1: Temporal Influence Maximization In this section, we show how to apply our method to the well-known Influence Maximization problem on a temporal network (TempInfMax) [2]. Given a propagation model, TempInfMax aims to find a seed-set $\mathcal{S} \subseteq V$ at time 0, which maximizes the 'footprint' (expected number of infected nodes) at time T. Solving it directly on large $\mathcal{G}$ can be very slow. Here we propose to use the much smaller $\mathcal{G}^{\text{cond}}$ as an approximation of $\mathcal{G}$, as it maintains the propagation-based properties well.

Specifically, we propose CondInf (Algorithm 2) to solve the TempInfMax problem on temporal networks. The idea is to get $\mathcal{G}^{\text{cond}}$ from NetCondense, solve TempInfMax problem on $\mathcal{G}^{\text{cond}}$, and map the results back to $\mathcal{G}$. Thanks to our well designed merging scheme that merges nodes with the similar diffusive property together, a simple random mapping is enough. To be specific, let the operator that maps node v from $\mathcal{G}^{\text{cond}}$ to $\mathcal{G}$ be $\zeta^{-1}(v)$. If v is a super-node then $\zeta^{-1}(v)$ returns a node sampled uniformly at random from v.

Algorithm 2 CondInf

```
Require: Temporal graph $\mathcal{G}$ , $0 < \alpha_N < 1$, $0 < \alpha_T < 1$
Ensure: seed set $\mathcal{S}$ of top $k$ seeds
1: $\mathcal{S} = \emptyset$
2: $\mathcal{G}^{\text{cond}} \leftarrow$ NetCondense ($\mathcal{G}$, $\alpha_N$, $\alpha_T$)
3: $k'_1, k'_2, ..., k'_S \leftarrow$ Run base TempInfMax on $\mathcal{G}^{\text{cond}}$
4: for every $k'_i$ do
5:     $k_i \leftarrow \zeta^{-1}(k'_i)$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{k_i\}$
6: return $\mathcal{S}$
```

We use two different base TempInfMax methods: ForwardInfluence [2] for the SI model and Greedy-OT [7] for the PersistentIC model. As our approach is general (our results can be easily extended to other models), and our actual algorithm/output is model-independent, we expect CondInf to perform well for both these methods. To calculate the footprint, we infect nodes in seed set $\mathcal{S}$ at time 0, and run the appropriate model till time T. We set $\alpha_T = 0.5$ and $\alpha_N = 0.5$ for all datasets. We show results only for ForwardInfluence in Table 3. The results for Greedy-OT were similar, however Greedy-OT did not even finish for datasets larger than Enron. As we can see, our method performs almost as good as the base method on $\mathcal{G}$, while being *significantly* faster (upto 48 times), showcasing its usefulness.

5.4 Application 2: Understanding/Exploring Networks We can also use NetCondense for 'sense-making' of temporal datasets: it ensures that important nodes and times remain unmerged while super-nodes and super-times form coherent interpretable groups of nodes and time-stamps. This is not the case for the baselines e.g. Tensor merges important nodes, giving

Table 3: Performance of CondInf (CI) with ForwardInfluence (FI) as base methods. σ_m and T_m are the footprint and running time for method m respectively. '-' means the method did not finish.

Dataset	σ_{FI}	σ_{CI}	T_{FI}	T_{CI}
School	130	121	14s	3s
Enron	110	107	18s	3s
Chess	1293	1257	36m	45s
Arxiv	23768	23572	3.7d	7.5h
Wikipedia	-	26335	-	7.1h

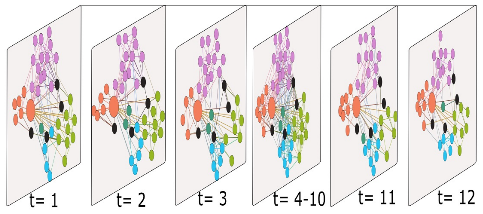


Figure 6: Condensed WorkPlace ($\alpha_N = 0.6$, $\alpha_T = 0.5$).

us heterogeneous super-nodes lacking interpretability.

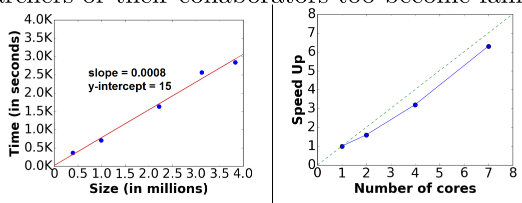
WorkPlace: It is a social-contact network between employees of a company with five departments, where weights are normalized contact time. In $\mathcal{G}^{\text{cond}}$ (see Figure 6), we find a super-node composed mostly of nodes from SRH (orange) and DSE (pink) departments, which were on floor 1 of the building while the rest were on floor 2. In the same super-node, surprisingly, we find a node from DMCT (green) department on floor 2 who has a high contact with DSE nodes. It turns out s/he was labeled as a "wanderer" in [4].

Unmerged nodes in the $\mathcal{G}^{\text{cond}}$ had high degree in all T. For example, we found nodes 80, 150, 751, and 255 (colored black) remained unmerged even for $\alpha_N = 0.9$. In fact, all these nodes were classified as "Linkers" whose temporal stability is crucial for epidemic spread [4]. The visualization of $\mathcal{G}^{\text{cond}}$ emphasizes that linkers connect consistently to nodes from multiple depts.; which is not obvious in the original networks. We also examined the super-times, and discovered that the days in and around the weekend (where there is little activity) were merged together.

School: It is socio-contact network between high school students from five different sections over several days [5]. In $\mathcal{G}^{\text{cond}}$, we find a super-node containing nodes from MP*1 and MP*2 sections and another super-node with nodes from remaining three sections PC, PC*, and PSI. The groupings in the super-nodes are intuitive as the dataset can broadly be divided into two components (MP*1 and MP*2) and (PC, PC*, and PSI) [5].

Enron: They are the email communication networks of employees of the Enron Corporation. In $\mathcal{G}^{\text{cond}}$ ($\alpha_N = 0.8, \alpha_T = 0.5$), we find that unmerged nodes are important nodes such as G. Whalley (President), K. Lay (CEO), and J. Skilling (CEO). We also found a star with Chief of Staff S. Kean in the center and important officials such as Whalley, Lay and J. Shankman (President) for six consecutive time-stamps. We also find a clique of high ranking officials in the same period. These structures appear only in consecutive time-stamps leading to when Enron declared bankruptcy. Sudden emergence, stability for over six/seven time-stamps, and sudden disappearance of these structures correctly suggests that a major event occurred during that time. We also note that time-stamps in 2001 were never merged, indicative of important and suspicious behavior.

DBLP: These are co-authorship networks from DBLP-CS bibiliography. This is an especially large dataset: hence exploration without any condensation is hard. In $\mathcal{G}^{\text{cond}}$ ($\alpha_N = 0.7, \alpha_T = 0.5$), we found that the unmerged nodes were very well-known researchers such as Philip S. Yu, Christos Faloutsos, Rakesh Aggarwal, and so on.We also find a giant super-node of size 395,000. An interesting observation is that famous researchers connect very weakly to the giant super-node. Whereas, less known researchers connect to the giant super-node with higher edge-weights. Another common pattern among famous researchers is that they connect to super-nodes only in the earlier time-stamps in the dataset. This observation suggests that as the authors become more famous, they collaborate less with non-famous researchers or their collaborators too become famous.



(a) Scalability (b) Parallelizability

Figure 7: Near-linear running time and speed-up.

5.5 Scalability and Parallelizability

Figure 7 (a) shows the runtime of NetCondense on the components of increasing size of Arxiv. NetCondense has subquadratic time complexity. In practice, it is *near-linear* w.r.t input size. Figure 7 (b) shows the *near-linear* run-time speed-up of parallel-NetCondense vs # cores on Wikipedia.

6 Related Work

Mining Dynamic Graphs. Dynamic graphs have gained a lot of interest recently (see [1] for a survey). Many graph mining tasks on static graphs have been introduced to dynamic graphs, including community detection [24] and link prediction [21]. Due to the increasing size, typically it is challenging to perform analysis on temporal networks.

Propagation. Cascade processes have been widely studied, including in epidemiology [3, 8], information diffusion [10], cyber-security [11] and product marketing [20]. A lot of work has been done on determining the epidemic threshold i.e. the conditions under which a virus causes an epidemic [6, 16, 17]. Examples of propagation-based optimization problems are influence maximization [10, 7, 2], and immunization [26]. Remotely related work deals with weak and strong ties over time for diffusion [9].

Graph Summarization. Here, we seek to find a compact representation of a large graph by leveraging global and local graph properties like local neighborhood structure [15], node/edge attributes [25], action logs [19], eigenvalue of the adjacency matrix [18], and key subgraphs. It is also related to graph sparsification algorithms [14]. The goal is to either reduce storage and manipulation costs, or simplify structure. Summarizing temporal networks has not seen much work, except recent papers based on bits-storage-compression [13], or extracting a list of recurrent sub-structures over time [22]. Unlike these, we are the first to focus on hierarchical condensation: using *structural* merges, giving a *smaller propagation-equivalent* temporal network.

7 Discussion and Conclusions

In this paper, we proposed a novel general Temporal Network Condensation Problem using the fundamental so-called 'system matrix' and present an effective, near-linear and parallelizable algorithm NetCondense. Using a variety of large datasets, we leverage it to dramatically speed-up influence maximization algorithms on temporal networks, and to explore and understand complex datasets. As also shown by our experiments, it is useful to note that our method itself is model-agnostic and has wide-applicability, thanks to our carefully chosen metrics which can be easily generalized to other propagation models. There are multiple ideas to explore: including leveraging NetCondense for other tasks such as role discovery and immunization.

Acknowledgements This paper is based on work partially supported by the NSF (IIS-1353346), the NEH (HG-229283-15), ORNL (Task Order 4000143330) and from the Maryland Procurement Office (H98230-14-C- 0127), and a Facebook faculty gift.

References

[1] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Survey*, 2014.

[2] C. C. Aggarwal, S. Lin, and S. Y. Philip. On influential node discovery in dynamic social networks. In *SDM*, 2012.

[3] R. M. Anderson and R. M. May. *Infectious Diseases of Humans.* Oxford University Press, 1991.

[4] M. G. et. al. Data on face-to-face contacts in an office building suggest a low-cost vaccination strategy based on community linkers. *Network Science*, 3(03), 2015.

[5] J. Fournet and A. Barrat. Contact patterns among high school students. *PloS one*, 9(9):e107878, 2014.

[6] A. Ganesh, L. Massoulie, and D. Towsley. The effect of network topology in spread of epidemics. *IEEE INFOCOM*, 2005.

[7] N. T. Gayraud, E. Pitoura, and P. Tsaparas. Diffusion maximization in evolving social networks. In *COSN*, 2015.

[8] H. W. Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[9] M. Karsai, N. Perra, and A. Vespignani. Time varying networks and the weakness of strong ties. *Scientific Reports*, 4:4001, 2014.

[10] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.

[11] J. O. Kephart and S. R. White. Measuring and modeling computer virus prevalence. In *Research in Security and Privacy*, pages 2–15. IEEE, 1993.

[12] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[13] W. Liu, A. Kan, J. Chan, J. Bailey, C. Leckie, J. Pei, and R. Kotagiri. On compressing weighted time-evolving graphs. In *CIKM12*. ACM, 2012.

[14] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537. ACM, 2011.

[15] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

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

[17] B. A. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos. Virus propagation on time-varying networks: Theory and immunization algorithms. In *ECML/PKDD10*, 2010.

[18] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. Subrahmanian. Fast influence-based coarsening for large networks. In *KDD14*.

[19] Q. Qu, S. Liu, C. S. Jensen, F. Zhu, and C. Faloutsos. Interestingness-driven diffusion process summarization in dynamic networks. In *ECML/PKDD*. 2014.

[20] E. M. Rogers. *Diffusion of innovations.* 2010.

[21] P. Sarkar, D. Chakrabarti, and M. Jordan. Nonparametric link prediction in dynamic networks. In *ICML*, 2012.

[22] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD15*.

[23] G. W. Stewart. *Matrix perturbation theory.* 1990.

[24] C. Tantipathananandh and T. Y. Berger-Wolf. Finding communities in dynamic social networks. In *ICDM11*.

[25] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *ICDE*, 2010.

[26] Y. Zhang and B. A. Prakash. Dava: Distributing vaccines over networks under prior information. In *SDM*, 2014.