Propagation based Temporal Network Summarization

Bijaya Adhikari, Yao Zhang, Sorour E. Amiri, Aditya Bharadwaj, B. Aditya Prakash
Department of Computer Science, Virginia Tech
Email: {bijaya, yaozhang, esorour, adb, badityap}@cs.vt.edu

Abstract—
Modern networks are very large in size and also evolve with time. As their sizes grow, 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 and event detection problems on temporal networks.

Index Terms—Graph Summarization, Temporal Networks, Propagation

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 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 orders 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, event detection, 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 *48x* in influence maximization and *3.8x* in event detection over dynamic networks.

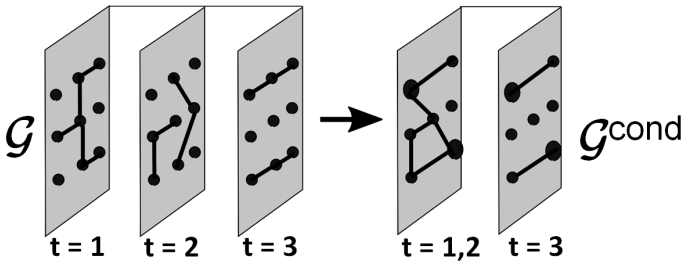


Fig. 1: Condensing a Temporal Network

The rest of the paper is organized in the following way. We present required preliminaries in Section 2, followed by problem formulation in Section 3. We present our approach and discuss empirical results in Sections 4 and 5 respectively. We discuss relevant prior works in Section 6 and we conclude in Section 7.

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 $G = (V, E, W)$ 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 $G_i = (V, E_i, W_i)$. Without loss of generality, 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 $V = \cup_{i=1}^{T} V_i$). We also assume, in principle there is a path for any node to send information to any other node in $\mathcal{G}$ (ignoring time), as otherwise we can simply decompose. 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 [5] and IC models [17]. 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 (Susceptible-Infected-Removed) model, where nodes in 'Removed' state do not get re-infected, with $\delta = 1$.

We consider generalizations of the SI model to temporal networks [27], where an infected node can only infect its susceptible 'current' neighbors (as given by $\mathcal{G}$). Specifically, any node a which is in the infected state at the beginning of time i, tries to infect any of its susceptible neighbor b in G_i with probability $w_i(a, b)$, where $w_i(a, b)$ is the edge-weight for edge (a, b) in G_i. Note that the SI model on static graphs are special cases of those on temporal networks (with all $G_i \in \mathcal{G}$ identical).

TABLE 1: Summary of symbols and descriptions

Symbol	Description
$\mathcal{G}$	Temporal Network
$\mathcal{G}^{\text{cond}}$	Condensed Temporal Network
G_i, $\mathbf{A}_i$	i$^{\text{th}}$ 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)

3 OUR PROBLEM FORMULATION

Real temporal networks are usually gigantic in size. However, their skewed nature [1] (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 [9]. 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 the 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 [11], [26] have studied epidemic thresholds for various epidemic models on static graphs. Recently, [27] 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 [27] 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 and $\mathbf{I}$ is the identity matrix. 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 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 $\Pr(a \to b|G_i, G_j) \approx [w_i(a, b) + w_j(a, b)]$, upto a first order approximation.

Proof: In the SI model, for node a to infect node b in time pair $\{i, j\}$, either the infection occurs in G_i or in G_j, therefore,

$$P(a \to b|G_k, G_j) = w_i(a, b) + (1 - w_i(a, b))w_j(a, b).$$

We have

$$P(a \to b|G_k, G_j) = w_i(a, b) + w_j(a, b) - w_i(a, b)w_j(a, b)$$

Now, ignoring the lower order terms, we have

$$P(a \to b|G_k, G_j) \approx w_i(a, b) + w_j(a, b).$$

□

Lemma 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 $T \times G_i$. 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 [14]). 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:

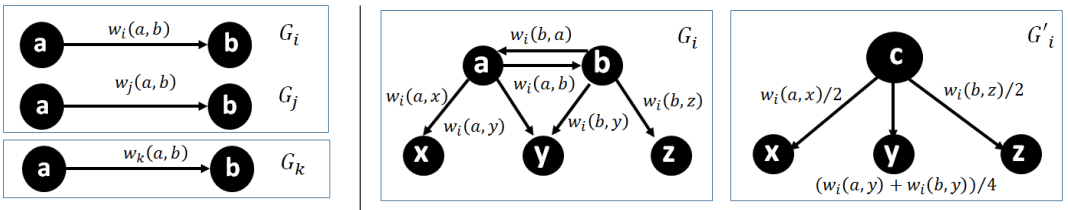
Definition 2. **Time-Pair Merge** $\mu(\mathcal{G}, i, j)$. The merge operator $\mu(\mathcal{G}, i, j)$ returns a new time-stamp k with weighted adjacency matrix $\mathbf{A}_k = \frac{\mathbf{A}_i + \mathbf{A}_j}{2}$.

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.

Lemma 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:

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



(a) Time merge of a single edge (b) Node Merge in a single time

Fig. 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.

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.

Proof: Note that $Nb_i^o(v)$ is the set of out-neighbors of node v at time-stamp i. When super-node c is infected in $G'_i \in \mathcal{G}^{\text{cond}}$ (the summary network), either node a or node b is infected in the underlying original network i.e, in $G_i \in \mathcal{G}$. Hence, for a node $z \in Nb_i^o(a) \backslash Nb_i^o(b)$, the probability of node c infecting z is,

$$P(c \to z|G_i) = \frac{P(a \to z|G_i) + P(b \to a|G_{i-1})P(a \to z|G_i)}{2}$$

Hence, if a is infected, it infects z at time i directly. But for b, to infect z at time i, b has to infect a at time $i-1$, and then a infects b at time i. We rewrite the probabilities as defined by the edge-weights,

$$P(c \to z|G_i) = \frac{w_i(a,z) + w_{i-1}(b,z)w_i(a,z)}{2}$$

Ignoring lowering order terms, we can get,

$$P(c \to z|G_i) \approx \frac{w_i(a,z)}{2}$$

Similarly, we can prove other cases. □

Motivated by Lemma 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 $w_i(z,c) = \Pr(z \to c|G_i)$.

Note: We use a first-order approximation of the infection probabilities in our merge definitions as the higher-order terms introduce non-linearity in the model. This in turn makes it more challenging to use matrix perturbation theory later [34]. As shown by our experiments, keepin just the first-order terms still leads to high quality summaries.

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 $|\lambda_{\mathbf{S}} - \lambda_{\mathbf{S}}^{\text{cond}}|$.

Problem 1 is likely to be challenging as it is related to immunization problems [40]. In fact, a slight variation of the problem can be easily shown to be NP-complete by reduction form the MAXIMUM CLIQUE problem [15] (see Appendix). Additionally, Problem 1 naturally contains the GCP coarsening problem for a static network [28] as a special case: when $\mathcal{G} = \{G\}$, which itself is 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.

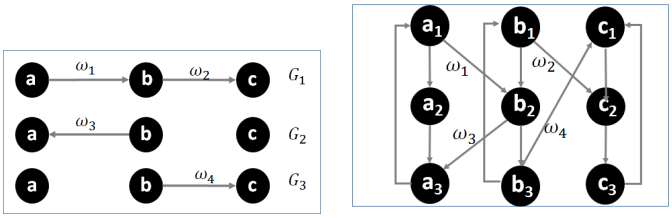
4.1 Main idea

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



(a) Temporal Network (b) Flattened Network

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

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 $a_{(i+1) \mod T}$ in layer $(i+1) \mod T$ 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 $b_{(i+1) \mod T}$ 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 3. (Equivalence of propagation in $\mathcal{G}$ and $F_{\mathcal{G}}$) We have $\sum_{t=1}^{T} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F_{\mathcal{G}}}(T)$.

Proof: First we will show the following:

$$\sum_{t=0}^{T-1} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F_{\mathcal{G}}}(T-1) \tag{1}$$

We prove this by induction over time-stamp $t = \{0, 1, ..., T-1\}$.

Base Case: At $t = 0$, since the seed set is the same, the infections in both the model are same. Hence, $I_{SI}^{\mathcal{G}}(0) = I_{IC}^{F_{\mathcal{G}}}(0)$

Inductive Step: For inductive step, let the inductive hypothesis be that for time-stamp $0 < k < T-1$, $\sum_{t=0}^{k} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F_{\mathcal{G}}}(k)$.

Let $\delta_{SI}^{\mathcal{G}}(k+1)$ be the number of new infection in the SI model in $\mathcal{G}$ at time $k+1$. The total number of infected nodes at time $k+1$ is $I_{SI}^{\mathcal{G}}(k) + \delta_{SI}^{\mathcal{G}}(k+1)$. Similarly, let $\delta_{IC}^{F_{\mathcal{G}}}(k+1)$ be the number of newly infected nodes at the time $k+1$. Since the number of $\delta_{SI}^{\mathcal{G}}(k+1)$ new nodes got infected in SI model in $\mathcal{G}$, the same number of nodes in the layer $k+2$ will get infected in $F_{\mathcal{G}}$. Moreover, all the nodes that are infected in layer $k+1$ at time k in $F_{\mathcal{G}}$ infect corresponding nodes in the next layer. Hence,

$$\delta_{IC}^{F_{\mathcal{G}}}(k+1) = \delta_{SI}^{\mathcal{G}}(k+1) + I_{SI}^{\mathcal{G}}(k)$$

Now, we have

$$\sum_{t=0}^{k+1} I_{SI}^{\mathcal{G}}(t) = \sum_{t=0}^{k} I_{SI}^{\mathcal{G}}(t) + I_{SI}^{\mathcal{G}}(k) + \delta_{SI}^{\mathcal{G}}(k+1)$$

By inductive hypothesis, we get,

$$\begin{aligned}\sum_{t=0}^{k+1} I_{SI}^{\mathcal{G}}(t) &= I_{IC}^{F}(k) + I_{SI}^{\mathcal{G}}(k) + \delta_{SI}^{\mathcal{G}}(k+1) \\ &= I_{IC}^{F}(k) + \delta_{IC}^{F_{\mathcal{G}}}(k+1) = I_{IC}^{F}(k+1)\end{aligned}$$

Now, at the time T, the infection in $\mathcal{G}$ occurs in time-stamp T, however, the infection in $F_{\mathcal{G}}$ occurs between layers T and 1. Recall that nodes are seeded in the layer 1 of $F_{\mathcal{G}}$ for IC model, hence they cannot get infected. Therefore, the difference in the cumulative sum of infection of SI and total infection in IC is $I_{IC}^{F_{\mathcal{G}}}(0)$. Therefore,

$$\sum_{t=0}^{T} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F}(T) + I_{IC}^{F_{\mathcal{G}}}(0)$$

Since $I_{IC}^{F}(0) = I_{SI}^{\mathcal{G}}(0)$, we have $\sum_{t=1}^{T} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F}(T)$. □

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).

Lemma 4. (Eigen-equivalence of $\mathbf{S}_{\mathcal{G}}$ and $\mathbf{F}_{\mathcal{G}}$) We have $(\lambda_{\mathbf{F}})^T = \lambda_{\mathbf{S}}$. Furthermore, λ is an eigenvalue of $\mathbf{F}_{\mathcal{G}}$, iff λ^T is an eigenvalue of $\mathbf{S}_{\mathcal{G}}$.

Proof: According to the definition of $\mathbf{F}_{\mathcal{G}}$, we have

$$\mathbf{F}_{\mathcal{G}} = \begin{pmatrix} 0 & \mathbf{I}+\mathbf{A}_1 & 0 & ... & 0 \\ 0 & 0 & \mathbf{I}+\mathbf{A}_2 & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & ... & \mathbf{I}+\mathbf{A}_{T-1} \\ \mathbf{I}+\mathbf{A}_T & 0 & 0 & ... & 0 \end{pmatrix}$$

Here $\mathbf{A}_i$ is the weighted adjacency matrix of $G_i \in \mathcal{G}$ and $\mathbf{I}$ is the identity matrix. Both have size $|V| \times |V|$. Now any eigenvalue λ and corresponding eigenvector $\mathbf{x}$ of $\mathbf{F}_{\mathcal{G}}$ satisfies the equation $\mathbf{F}_{\mathcal{G}}\mathbf{x} = \lambda\mathbf{x}$. We can actually decompose $\mathbf{x}$ as $\mathbf{x} = [\mathbf{x}_1^{'}, \mathbf{x}_2^{'}, \dots \mathbf{x}_T^{'}]^{'}$, where each $\mathbf{x}_i$ is a vector of size $|V| \times 1$. Hence, we get the following:

$$\mathbf{F}_{\mathcal{G}} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{T-1} \\ \mathbf{x}_T \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{T-1} \\ \mathbf{x}_T \end{pmatrix}$$

From the above equation, we can get

$$(\mathbf{I} + \mathbf{A}_1)\mathbf{x}_2 = \lambda \mathbf{x}_1,$$
$$(\mathbf{I} + \mathbf{A}_2)\mathbf{x}_3 = \lambda \mathbf{x}_2,$$
$$\vdots$$
$$(\mathbf{I} + \mathbf{A}_T\mathbf{x}_1) = \lambda \mathbf{x}_T$$

Multiplying the equations together,

$$\left[\prod_{i=1}^{T}(I + \mathbf{A}_i)\right] \mathbf{x}_1 = \lambda^T \cdot \mathbf{x}_1$$

Finally,

$$\mathbf{S}_{\mathcal{G}} \cdot \mathbf{x}_1 = \lambda^T \cdot \mathbf{x}_1$$

Hence λ^T is the eigenvalue of $\mathbf{S}_{\mathcal{G}}$ if λ is the eigenvalue of $\mathbf{F}_{\mathcal{G}}$. Same argument in reverse proves the converse.

Now, since $U_{\mathcal{G}}$ is strongly-connected, we have $|\lambda_{\mathbf{F}}| \geq |\lambda|$, for any λ that is eigenvalue of $\mathbf{F}_{\mathcal{G}}$. And we also have, if $|x| > |y|$ then $|x^k| > |y^k|$ for any $k > 1$. Therefore there are not any λ such that $|\lambda^T| > |\lambda_{\mathbf{F}}^T|$. So, λ_F^T has to be the principal eigenvalue of $S_{\mathcal{G}}$. □

Lemma 4 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. Note that $\mathbf{F}_{\mathcal{G}}$ is a very sparse directed network. The sparsity causes the smallest eigenvalue to be very small. Hence, the condition number [8], defined as the ratio of the largest eigenvalue to the smallest eigenvalue in absolute value is high, implying that the matrix is ill-conditioned. The ill-conditioned matrices are unstable for numerical operations. Therefore 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 5. (Eigenvalue relationship of $\mathbf{F}_{\mathcal{G}}$ and $\mathbf{X}_{\mathcal{G}}$) The largest eigenvalue of $\mathbf{F}_{\mathcal{G}}$, $\lambda_{\mathbf{F}}$, and the largest eigenvalue of $\mathbf{X}_{\mathcal{G}}$, $\lambda_{\mathbf{X}}$, are related as $\lambda_{\mathbf{F}} \leq \lambda_{\mathbf{X}}$.

Proof: First, according to the definition, $\mathbf{X}_{\mathcal{G}} = \frac{\mathbf{F}_{\mathcal{G}} + \mathbf{F}_{\mathcal{G}}'}{2}$. Let $\lambda(\mathbf{F}_{\mathcal{G}})$ be the spectrum of $\mathbf{F}_{\mathcal{G}}$ and $\lambda(\mathbf{X}_{\mathcal{G}})$ be spectrum of $\mathbf{X}_{\mathcal{G}}$. Let $\lambda_{\mathbf{X}}$ be the largest eigenvalue of $\mathbf{X}_{\mathcal{G}}$. Function $\mathrm{Re}(c)$ returns the real part of c.

Now, $\lambda(\mathbf{F}_{\mathcal{G}})$ and $\lambda(\mathbf{X}_{\mathcal{G}})$ are related by the majorization relation [38]. i.e., $\mathrm{Re}(\lambda(\mathbf{F}_{\mathcal{G}})) \prec \lambda(\mathbf{X}_{\mathcal{G}})$, which implies that any eigenvalue of $\mathbf{F}_{\mathcal{G}}$, $\lambda \in \lambda(\mathbf{F}_{\mathcal{G}})$, satisfies $\mathrm{Re}(\lambda) \leq \lambda_{\mathbf{X}}$.

Since the union graph $U_{\mathcal{G}}$ is strongly connected, $F_{\mathcal{G}}$ is strongly connected. Hence, by Perron Frobenius theorem [12], the largest eigenvalue of $\mathbf{F}_{\mathcal{G}}$, $\lambda_{\mathbf{F}}$, is real and positive. Therefore, $\lambda_{\mathbf{F}} \leq \lambda_{\mathbf{X}}$. □

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 $\mathbf{S}_{\mathcal{G}}$. Since $\mathbf{X}_{\mathcal{G}}$ is symmetric and denser than $\mathbf{F}_{\mathcal{G}}$, it is well-conditioned and more stable for numerical operations. Hence, its eigenvalue can be efficiently computed.

New problem: Considering all of the above, we re-formulate 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 $\mathcal{G}$.

Note that edges in one time-stamp of $\mathcal{G}$ are represented between *two* layers in $X_{\mathcal{G}}$ and edges in two consecutive time-stamps in $\mathcal{G}$ are represented in *three* consecutive layers in $X_{\mathcal{G}}$. Hence, merging a time-pair in $\mathcal{G}$ corresponds to merging three layers of $X_{\mathcal{G}}$.

A notable difference in $\mu(\mathcal{G}, ., .)$ and $\mu(X_{\mathcal{G}}, ., .)$ arises due to the difference in propagation models; we have the SI model in $\mathcal{G}$ whereas we the IC model in $X_{\mathcal{G}}$. Since a node gets only a single chance to infect its neighbors in the IC model, infectivity does not need re-scaling $X_{\mathcal{G}}$. Despite this difference, the merge definitions on $\mathcal{G}$ and $X_{\mathcal{G}}$ remain identical.

Let us assume we are merging time-stamps i and j in $\mathcal{G}$. For this, we need to look at the edges between layers i and j, and j and k, where k is layer following j. Now, merging time-stamps i and j in $\mathcal{G}$ corresponds to merging layers i and j in $X_{\mathcal{G}}$ and updating out-links and in-links in the new layers. Let $w_{i,j}(a, b)$ be the edge weight between any node a in layer i.

Definition 5. **Time-Pair Merge** $\mu(X_{\mathcal{G}}, i, j)$. The merge operator $\mu(X_{\mathcal{G}}, i, j)$ results in a new layer m such that edge weight between any nodes a in layer m and b in layer k, $w_{m,k}(a, b)$ is defined as

$$w_{m,k}(a, b) = \frac{w_{i,j}(a, b) + w_{j,k}(a, b)}{2}$$

Note for $h = i - 1 \mod T$, $w_{h,i}$ and $w_{h,m}$ are equal, since the time-stamp h in $\mathcal{G}$ does not change. And as $\mathbf{X}_{\mathcal{G}}$ is symmetric $w_{m,k}$ and $w_{k,m}$ are equal. Similarly, we extend node-pair merge definition in $\mathcal{G}$ as follows. As in $\mathcal{G}$, we merge node-pairs in all layers of $X_{\mathcal{G}}$.

Definition 6. **Node-Pair Merge** $\zeta(X_{\mathcal{G}}, a, b)$. Let $Nb^o(v)$ denote the set of out-neighbors of a node v. Let $w_{i,j}(a, b)$ be edge weight from any node a in layer i to any node

b at layer j. Then the merge operator $\zeta(X_{\mathcal{G}}, a, b)$ merges node pair a, b to form a super-node c, whose edges to out-neighbors are weighted as

$$w_{i,j}(c,z) = \begin{cases} \frac{w_{i,j}(a,z)}{2} & \forall z \in Nb^o(a) \backslash Nb^i(b) \\ \frac{w_{i,j}(b,z)}{2} & \forall z \in Nb^o(b) \backslash Nb^i(a) \\ \frac{w_{i,j}(a,z) + w_{i,j}(b,z)}{4} & \forall z \in Nb^o(a) \cap Nb^i(b) \end{cases}$$

Finally, our problem can be re-formulated as following.

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 $|\lambda_{\mathbf{X}} - \lambda_{\mathbf{X}}^{\text{cond}}|$.

4.4 NETCONDENSE

In this section, we propose a fast top-k selection 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 7. **Δ-Score**. $\Delta_{X_{\mathcal{G}}}(a,b) = |\lambda_{\mathbf{X}} - \lambda_{\mathbf{X}}^{\text{cond}}|$ where $\lambda_{\mathbf{X}}^{\text{cond}}$ 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 an expensive algorithm (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 at least one $G_i \in \mathcal{G}$, i.e. edge in the union of all $G_i \in \mathcal{G}$, called the Union Graph $U_{\mathcal{G}}$. Now, computing Δ-Score simply for all edges $(a,b) \in U_{\mathcal{G}}$ is still expensive, as it requires computing eigenvalue of $\mathbf{X}_{\mathcal{G}}$ for each node-pair. Hence we *estimate* Δ-Score for node/time pairs instead using Matrix Perturbation Theory [34]. Let $\mathbf{v}$ be the eigenvector of $\mathbf{X}_{\mathcal{G}}$, corresponding to $\lambda_{\mathbf{X}}$. Let $\mathbf{v}(a_i)$ be the 'eigenscore' of node a_i in $\mathbf{X}_{\mathcal{G}}$. $\mathbf{X}_{\mathcal{G}}(a_i, b_i)$ is the entry in $\mathbf{X}_{\mathcal{G}}$ in the row a_i and the column b_i. Now we have the following lemmas.

Lemma 6. (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,

$$\Delta_{X_{\mathcal{G}}}(i,j) = \frac{-\lambda_{\mathbf{X}}(\sum_{i \in V_i, V_j} \mathbf{v}(i)^2) + \sum_{k \in V_k} \mathbf{v}(i)\mathbf{k}^{oT}\mathbf{v} + Y}{\mathbf{v}^T\mathbf{v} - \sum_{i \in V_i, V_j} \mathbf{v}(i)^2}$$

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

Proof: For convenience, we write $\lambda_{\mathbf{X}}$ as λ and $\mathbf{X}_{\mathcal{G}}$ as $\mathbf{X}$. Similarly, we write $\mathbf{v}(x_i)$ as $\mathbf{v}_i$. Now, according to the matrix perturbation theory, we have

$$\Delta\lambda = \frac{\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} + \mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v}}{\mathbf{v}^T\mathbf{v} + \mathbf{v}^T\Delta\mathbf{v}} \tag{2}$$

When we merge a time-pair in $\mathbf{X}$, we essentially merge blocks corresponding to the time-stamps i and j in $\mathbf{X}$ to create new blocks corresponding to time-stamp k. Since we want to maintain the size of the matrix as the algorithm proceeds, we place layer k in layer i's place and set rows and columns of $\mathbf{X}_{\mathcal{G}}$ corresponding to layer j to be zero. Therefore, the change in $\mathbf{X}$ can be written as

$$\Delta\mathbf{X} = \sum_{i \in V_i, V_j} -(\mathbf{i}^i\mathbf{e}_i^T + \mathbf{e}_i\mathbf{i}^{oT}) + \sum_{k \in V_k} -(\mathbf{k}^i\mathbf{e}_k^T + \mathbf{e}_k\mathbf{k}^{oT}) \tag{3}$$

where $\mathbf{e}_a$ is a column vector with 1 at position a and 0 elsewhere, and $\mathbf{k}^i$ and $\mathbf{k}^{oT}$ are k-th column and row vectors of $\mathbf{X}$ respectively. Similarly, the change in the right eigenvector can be written as:

$$\Delta\mathbf{v} = \sum_{i \in V_i, V_j} -(\mathbf{v}_i\mathbf{e}_i) + \delta \tag{4}$$

As δ is very small, we can ignore it. Note that $\mathbf{v}^T\mathbf{e}_i = \mathbf{v}_i$, $\mathbf{v}^T\mathbf{i}^i = \lambda\mathbf{v}_i$ and $\mathbf{i}^{oT}\mathbf{v} = \lambda\mathbf{v}_i$. Now, we can compute Eqn. 2 as follows:

$$\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} = \sum_{i \in V_i, V_j} -(\mathbf{v}_i\mathbf{v}^T\mathbf{i}^i + \mathbf{v}_i\mathbf{i}^{oT}\mathbf{v}) + \sum_{k \in V_k} (\mathbf{v}_i\mathbf{v}^T\mathbf{k}^i + \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}) \tag{5}$$

Further simplifying,

$$\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} = \sum_{i \in V_i, V_j} -(2\lambda\mathbf{v}_i^2) + \sum_{k \in V_k} (\mathbf{v}_i\mathbf{v}^T\mathbf{k}^i + \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}) \tag{6}$$

And we have

$$\mathbf{v}^T\Delta\mathbf{v} = \mathbf{v} \sum_{i \in V_i, V_j} (-\mathbf{v}_i\mathbf{e}_i) = -\sum_{i \in V_i, V_j} \mathbf{v}_i^2 \tag{7}$$

Similarly,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \mathbf{v}^T[\sum_{i \in V_i, V_j} -(\mathbf{i}^i\mathbf{e}_i^T + \mathbf{e}_i\mathbf{i}^{oT}) + \sum_{k \in V_k} -(\mathbf{k}^i\mathbf{e}_k^T + \mathbf{e}_k\mathbf{k}^{oT})] [\sum_{i \in V_i, V_j} -(\mathbf{v}_i\mathbf{e}_i)] \tag{8}$$

Here we notice that there are edges between two layers in $X_{\mathcal{G}}$, only if they are adjacent. Moreover, the edges are in both directions between the layers. Hence,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \sum_{i \in V_i, V_j} \lambda\mathbf{v}_i\mathbf{v}_j + \sum_{i \in V_i, j \in V_j} (\mathbf{v}_i\mathbf{v}_j + \mathbf{v}_j\mathbf{v}_i)\mathbf{X}(i,j) - \sum_{k \in V_k} \mathbf{v}_i\mathbf{v}^T\mathbf{k}^i - \sum_{k \in V_k, j \in V_j} \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}_j\mathbf{e}_j. \tag{9}$$

Since self loops have no impact on diffusion, we can write $\sum_{k \in V_k, j \in V_j} \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}_j\mathbf{e}_j = 0$. Hence,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \sum_{i \in V_i, V_j} \lambda\mathbf{v}_i\mathbf{v}_j + \sum_{i \in V_i, j \in V_j} (2\mathbf{v}_i\mathbf{v}_j\mathbf{X}(i,j) - \sum_{k \in V_k} \mathbf{v}_i\mathbf{v}^T\mathbf{k}^i \tag{10}$$

Putting together, we have

$$\Delta\lambda = \frac{-\lambda\sum_{i \in V_i, V_j} \mathbf{v}_i^2 + \sum_{i \in V_i, j \in V_j} 2\mathbf{v}_i\mathbf{v}_j)\mathbf{X}(i,j) + \sum_{k \in V_k} \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}}{\mathbf{v}^T\mathbf{v} - \sum_{i \in V_i, V_j} \mathbf{v}_i^2} \tag{11}$$

Note that we merge the same node in different layers of $X_{\mathcal{G}}$ corresponding to different time-stamps in $\mathcal{G}$. Now, Let i be a node in t_i and j the same node in t_j, and we merge them to get new node k in t_k. Notice that i and j cannot have common neighbors. Let $Nb^o(v)$ be the set of out-neighbors of node v. For brevity, let $I = Nb^o(i)$ and $J = Nb^o(j)$. We have the following,

$$\begin{aligned}\mathbf{k}^{oT}\mathbf{v} &= \sum_{y\in I}\mathbf{v}_y\mathbf{k}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\mathbf{k}_z^{oT} \\ &= \sum_{y\in I}\mathbf{v}_y\frac{1}{2}\mathbf{i}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\frac{1}{2}\mathbf{j}_z^{oT}\end{aligned} \tag{12}$$

Now, let w be the edge-weight between i and j, $\lambda\mathbf{v}_i = \sum_{y\in I}\mathbf{v}_y\mathbf{i}_y^{oT} - \mathbf{v}_j w$ therefore, $\sum_{y\in I}\mathbf{v}_y\mathbf{i}_y^{oT} = \lambda\mathbf{v}_i + \mathbf{v}_j w$

Similarly, $\sum_{z\in B}\mathbf{v}_z\mathbf{j}_y^{oT} = \lambda\mathbf{v}_j + \mathbf{v}_i w$. By construction, we have $w = 1$ in $X_{\mathcal{G}}$. Hence,

$$\mathbf{k}^{oT}\mathbf{v} = \frac{1}{2}(\lambda_{\mathbf{X}}\mathbf{v}_i + \lambda_{\mathbf{X}}\mathbf{v}_j + \mathbf{v}_i + \mathbf{v}_j). \tag{13}$$

□

Lemma 7. (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,

$$\Delta_{X_{\mathcal{G}}}(a,b) = \frac{-\lambda_{\mathbf{X}}(\sum_{a\in V_a,V_b}\mathbf{v}(a)^2) + \sum_{c\in V_c}\mathbf{v}(a)\mathbf{c}^{oT}\mathbf{v} + Y}{\mathbf{v}^T\mathbf{v} - \sum_{a\in V_a,V_b}\mathbf{v}(a)^2}$$

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

Proof: Following the steps in Lemma 6, we have

$$\Delta\lambda = \frac{-\lambda\sum_{i\in V_i,V_j}\mathbf{v}_i^2 + \sum_{i\in V_i,j\in V_j}(2\mathbf{v}_i\mathbf{v}_j)\mathbf{X}(i,j) + \sum_{k\in V_k}\mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}}{\mathbf{v}^T\mathbf{v} - \sum_{i\in V_i,V_j}\mathbf{v}_i^2} \tag{14}$$

Note that we merge the same node in different layers of $X_{\mathcal{G}}$ corresponding to different time-stamps in $\mathcal{G}$. Now, Let i be a node in t_i and j the same node in t_j, and we merge them to get new node k in t_k. Notice that i and j cannot have common neighbors. Let $Nb^o(v)$ be the set of out-neighbors of node v. For brevity, let $I = Nb^o(i)$ and $J = Nb^o(j)$. We have the following,

$$\begin{aligned}\mathbf{k}^{oT}\mathbf{v} &= \sum_{y\in I}\mathbf{v}_y\mathbf{k}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\mathbf{k}_z^{oT} \\ &= \sum_{y\in I}\mathbf{v}_y\frac{1}{2}\mathbf{i}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\frac{1}{2}\mathbf{j}_z^{oT}\end{aligned} \tag{15}$$

Now, let w be the edge-weight between i and j, $\lambda\mathbf{v}_i = \sum_{y\in I}\mathbf{v}_y\mathbf{i}_y^{oT} - \mathbf{v}_j w$ therefore, $\sum_{y\in I}\mathbf{v}_y\mathbf{i}_y^{oT} = \lambda\mathbf{v}_i + \mathbf{v}_j w$.

Similarly, $\sum_{z\in B}\mathbf{v}_z\mathbf{j}_y^{oT} = \lambda\mathbf{v}_j + \mathbf{v}_i w$. By construction, we have $w = 1$ in $X_{\mathcal{G}}$. Hence,

$$\mathbf{k}^{oT}\mathbf{v} = \frac{1}{2}(\lambda_{\mathbf{X}}\mathbf{v}_i + \lambda_{\mathbf{X}}\mathbf{v}_j + \mathbf{v}_i + \mathbf{v}_j). \tag{16}$$

□

Lemma 8. 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 $O(E + TV)$.

Proof: Line 1 in NETCONDENSE takes $O(E)$ time. To calculate the largest eigenvalue and corresponding eigenvector of $\mathbf{X}_{\mathcal{G}}$ using the Lanczos algorithm takes $O(E)$ time [37]. It takes $O(TV + E)$ time for Lines 2 and 3. It takes $O(T)$ time to calculate score for each node pair. Therefore, Lines 4 and 5 take $O(TE_u)$. Line 6 takes $O(T\log T)$ to sort Δ-Score for time-pairs and $O(E\log E)$ to sort Δ-Score for node-pairs in the worst case. Now, for Lines 8 to 10, merging node-pairs require us to look at neighbors of nodes being merged at each time-stamp. Hence, it takes $O(T\theta)$ time and we require $\alpha_N V$ merges. Therefore, time-complexity for all node-merge is $O(\alpha_N\theta TV)$. Similarly, it takes $O(\alpha_T E)$ time for all time-merges.

Therefore, the time-complexity of NETCONDENSE is $O(TE_u + E\log E + \alpha_N\theta TV + \alpha_T E)$. Note that the complexity is *sub-quadratic* as $E_u \leq V^2$ (In our large real datasets, we found $E_u << V^2$).

For NETCONDENSE, we need $O(E)$ space to store $X_{\mathcal{G}}$ and $\mathcal{G}^{\text{cond}}$. We also need $O(E_u)$ and $O(T)$ space to store scores for node-pairs and time-pairs respectively. To store eigenvectors of $\mathbf{X}_{\mathcal{G}}$, we require $O(TV)$ space. Therefore, total space-complexity of NETCONDENSE is $O(E + TV)$. □

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 6
4: **for** every node-pair $\{a, b\}$ in $U_{\mathcal{G}}$ **do**
5: Calculate $\Delta_{X_{\mathcal{G}}}(a,b)$ using Lemma 7
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 $\mathcal{G}^{\text{cond}}$

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].

Datasets. We run NETCONDENSE on a variety of real datasets (Table 4) of varying sizes from different domains such as social-interactions (`WorkPlace`, `School`, `Chess`),

1. http://people.cs.vt.edu/~bijaya/code/NetCondense.zip

TABLE 2: Datasets Information.

Dataset	Weight	$\|V\|$	$\|E\|$	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
ProsperLoan	# Loans	89K	**3.3M**	7 Years
Wikipedia	# Pages	118K	**2.1M**	10 Years
WikiTalk	# Messages	497K	**2.7M**	12 Years
DBLP	# Papers	**1.3M**	**18M**	25 Years

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 $[0, 1]$.

WorkPlace, and School are contact networks publicly available from SocioPatterns[2]. In both datasets, edges indicate that two people were in proximity in the given time-stamp. Weights represent the total time of interaction in each day.

Enron is a publicly available dataset[3]. It contains edges between core employees of the corporation aggregated over 44 weeks. Weights in Enron represent the count of emails.

Chess is a network between chess players. Edge-weight represents number of games played in the time-stamp.

Arxiv is a co-authorship network in scientific papers present in arXiv's High Energy Physics – Phenomenology section. We aggregate this network yearly, where the weights are number of co-authored papers in the given year.

ProsperLoan is loan network among user of Prosper.com. We aggregate the loan interaction among users to define weights.

Wikipedia is an edit network among users of English Wikipedia. The edges represent that two users edited the same page and weights are the count of such events.

WikiTalk is a communication network among users of Spanish Wikipedia. Edge between nodes represent that users communicates with each other in the given time-stamp. Weight in this dataset is aggregated count of communication.

DBLP is coauthorship network from DBLP bibliography, where two authors have an edge between them if they have co-authored a paper in the given year. We define the weights for co-authorship network as the number of co-authored papers in the given year.

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 [19] 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.

2. http://www.sociopatterns.org/
3. https://www.cs.cmu.edu/~/enron/

CNTEMP: We run Coarsenet [28] (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 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 $\mathcal{G}$.

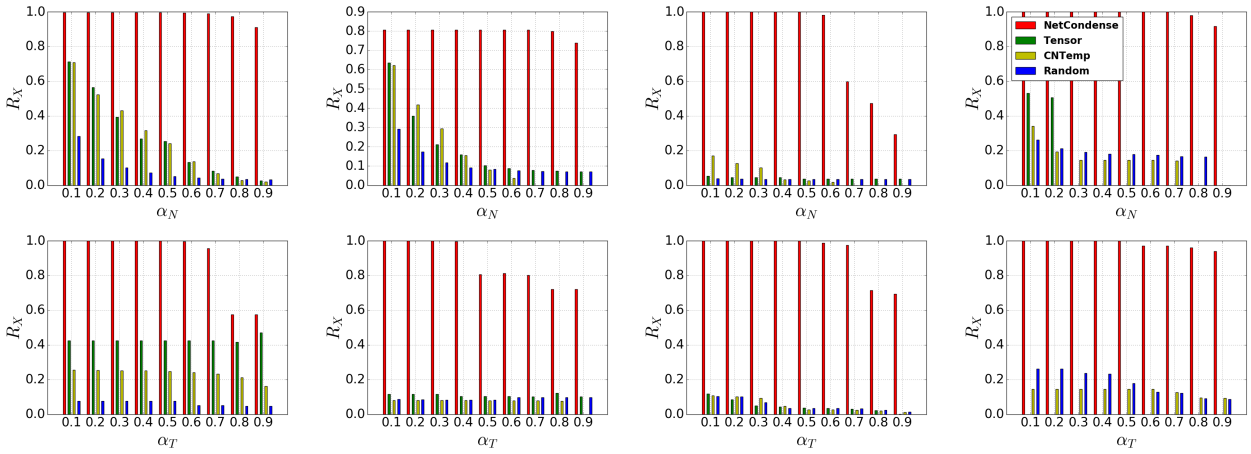
We also compare our performance for Problem 1, against an algorithm specifically designed for it. We use the simple greedy algorithm GREEDYSYS for Problem 1 (as the brute-force is too expensive): it greedily picks top node/time merges by actually re-computing $\lambda_{\mathbf{S}}$. We can run GREEDYSYS 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.

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) [3]. 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.

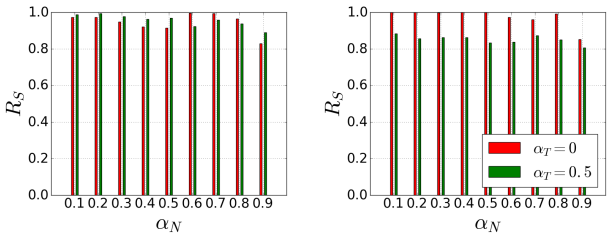
Specifically, we propose CONDINF (Algorithm 2) to solve the TempInfMax problem on temporal networks. The idea is to get $\mathcal{G}^{\text{cond}}$ from NETCONDENSE, solve TempInfMax problem on $\mathcal{G}^{\text{cond}}$, and map the results back to $\mathcal{G}$. Thanks to our well designed merging 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.

We use two different base TempInfMax methods: FORWARDINFLUENCE [3] for the SI model and GREEDY-OT [13] 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 use footprints and



(a) Arxiv (b) ProsperLoan (c)WikiTalk (d) DBLP

Fig. 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, $\alpha_N = 0.5$).



(a) WorkPlace (b) School

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

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 ($\mathcal{G}$, α_N, α_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 $\mathcal{S}$

running time as two measurements. We set $\alpha_T = 0.5$ and $\alpha_N = 0.5$ for all datasets for ForwardInfluence. Similarly, We set $\alpha_T = 0.5$ and $\alpha_N = 0.5$ for School, Enron, and Chess, $\alpha_N = 0.97$ for Arxiv, and $\alpha_N = 0.97$ for Wikipedia for Greedy-OT (as Greedy-OT is very slow). We show results for ForwardInfluence and Greedy-OT in Table 3. The results for Greedy-OT shows that it 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: Event Detection

Event detection [4], [30] is an important problem in temporal networks. The problem seeks to identify time points at which there is a significant change in a temporal network. As snapshots in a temporal network $\mathcal{G}$ evolve, with new nodes and edges appearing and existing ones disappearing, it is important to ask if a snapshot of $\mathcal{G}$ at a given time differs significantly from earlier snapshots. Such time points signify intrusion, anomaly, failure, e.t.c depending upon the domain of the network. Formally, the event detection problem is defined as follows:

Problem 3. (Event Detection Problem (EDP)) Given a temporal network $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$, find a list $\mathcal{R}$ of time-stamps t, such that $1 \leq t \leq T$ and G_{t-1} differs significantly from G_t.

As yet another application of NetCondense, in this section we show that summary $\mathcal{G}^{\text{cond}}$ of a temporal network $\mathcal{G}$ returned by NetCondense can be leveraged to speed up the event detection task. We show that one can actually solve the event detection task on $\mathcal{G}^{\text{cond}}$ instead of $\mathcal{G}$ and still obtain high quality results. Since, NetCondense groups only homogeneous nodes together and preserves important characteristics of the original network $\mathcal{G}$, we hypothesize that running SnapNETS [4] on $\mathcal{G}$ and $\mathcal{G}^{\text{cond}}$ should produce similar results. Moreover, due to the smaller size of $\mathcal{G}^{\text{cond}}$ running SnapNETS on $\mathcal{G}^{\text{cond}}$ is faster than running it on much larger $\mathcal{G}$. In our method, given a temporal network $\mathcal{G}$ and node reduction factor α_N (we set time reduction factor α_T to be 0), we obtain $\mathcal{G}^{\text{cond}}$ and solve the EDP problem on $\mathcal{G}^{\text{cond}}$. Specifically, we propose CondED (Algorithm 3) to solve the event detection problem.

Algorithm 3 CondED

Require: Temporal graph $\mathcal{G}$, $0 < \alpha_N < 1$
Ensure: List $\mathcal{R}$ of time-stamps
1: $\mathcal{G}^{\text{cond}} \leftarrow$ NetCondense ($\mathcal{G}$, α_N, 0)
2: $\mathcal{R} \leftarrow$ Run base Event Detection on $\mathcal{G}^{\text{cond}}$
3: return $\mathcal{R}$

For EDP we use SnapNETS as the base method. In addition to some of the datasets previously used, we run CondED on other datasets which are previously used for event detection [4]. These datasets are described below in detail and the summary is in Table 4.

AS Oregon-PA and AS Oregon-MIX are Autonomous Systems peering information network collected from the

TABLE 3: Performance of CONDINF (CI) with FORWARDINFLUENCE (FI) and GREEDY-OT (GO) 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

Dataset	σ_{GO}	σ_{CI}	T_{GO}	T_{CI}
School	135	128	15m	1.8m
Enron	119	114	9.8m	24s
Chess	-	2267	-	8.6m
Arxiv	-	357	-	2.2h
Wikipedia	-	4591	-	3.2h

TABLE 4: Additional Datasets for EDP.

Dataset	$\|V\|$	$\|E\|$	T
Co-Occurence	202	2.8K	31 Days
AS Oregon-PA	633	1.08K	60 Units
AS Oregon-MIX	1899	3261	70 Units
IranElection	126K	**5.5M**	30 Days
Higgs	456K	**14.8M**	7 Days

TABLE 5: Performance of CONDED. F1 stands for F1-Score. Speed-up is the ratio of time to run SNAPNETS on $\mathcal{G}$ to the time to run SNAPNETS on $\mathcal{G}^{\text{cond}}$.

	$\alpha_N = 0.3$		$\alpha_N = 0.7$	
Dataset	F1	Speed-Up	F1	Speed-Up
Co-Occurence	1	1.23	0.18	1.24
School	1	1.05	1	1.56
AS Oregon-PA	1	1.43	1	2.08
AS Oregon-MIX	1	1.22	1	2.83
IranElection	1	1.27	1	3.19
Higgs	0.66	1.17	1	3.79
Arxiv	1	1.09	1	2.27

Oregon router views[4]. IranElection and Higgs are twitter networks, where the nodes are twitter user and edges indicate follower-followee realtionship. More details on these datasets are given in [4].

Co-Occurence is a word co-occurrence network extracted from historical newspapers, published in January 1890, obtained from the library of congress[5]. Nodes in the network are the keywords and edges between two keywords indicate that they co-appear in a sentence in a newspaper published in a particular day. The edge-weights indicate the frequency with which two words co-appear.

To evaluate performance of CONDED, we compare the list of time-stamps $\mathcal{R}^{\text{cond}}$ obtained by CONDED with the list of time-stamps $\mathcal{R}$ obtained by SNAPNETS on the original network $\mathcal{G}$. We treat the time-stamps discovered by SNAPNETS as the ground truth and following the methodology in [4], we compute the F-1 score. We repeat the experiment with $\alpha_N = 0.3$ and $\alpha_N = 0.7$. The results are summarized in Table 5.

As shown in Table 5, CONDED has very high F1-score for most datasets even when $\alpha_N = 0.7$. This suggests that the list of time-stamps $\mathcal{R}^{\text{cond}}$ returned by CONDED matches the result from SNAPNETS very closely. Moreover, the time taken for the base method to run in $\mathcal{G}^{\text{cond}}$ is upto 3.5 times faster than the time it takes to run on $\mathcal{G}$.

However, for Co-Occurence dataset, the F1-score for $\alpha_N = 0.7$ is a mere 0.18, despite having F1-score of 1 for $\alpha_N = 0.3$. Note that Co-Occurence is one of the smallest dataset that we have, hence very high α_N seems to deteriorate the structure of the network, which suggests a different value of α_N is suitable for different networks in the EDP task.

4. http://www.topology.eecs.umich.edu/data.html
5. http://chroniclingamerica.loc.gov

5.5 Application 3: Understanding/Exploring Networks

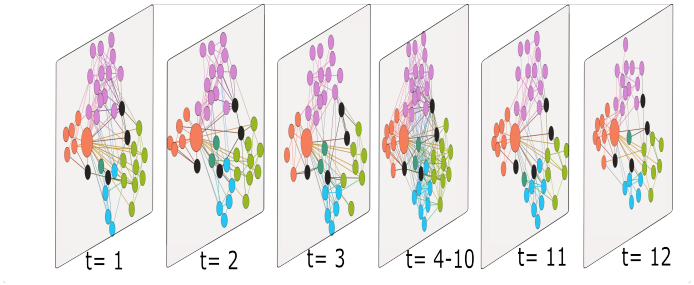


Fig. 6: Condensed WorkPlace ($\alpha_N = 0.6$, $\alpha_T = 0.5$).

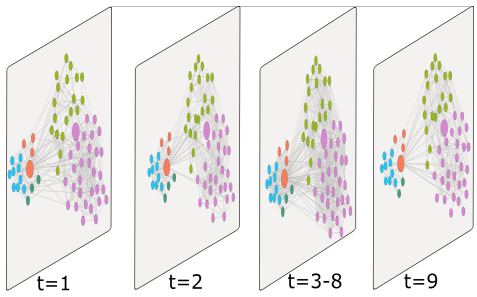


Fig. 7: Condensed School ($\alpha_N = 0.5$ and $\alpha_T = 0.5$).

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 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. It has been used for vaccination studies [9]. 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. We also noticed that the proportion of contact times between individuals of SRH and DSE were high in all the time-stamps. 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 [9].

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 $\alpha_N = 0.9$ suggesting their importance. In fact, all these nodes were classified as "Linkers" whose temporal stability is crucial for epidemic spread [9]. The stability of "Linkers" mentioned in [9] suggests that these are important nodes in every time-stamp. The visualization of $\mathcal{G}^{\text{cond}}$ emphasizes that linkers connect consistently to nodes from multiple departments; which is not obvious in the original networks. We also examined the super-times, and found that the days in and around the weekend (where there is little activity) were merged together.

`School`: It is a socio-contact network between high school students from five different sections over several days [10]. We condensed `School` dataset with $\alpha_N = 0.5$ and $\alpha_T = 0.5$. The students in the dataset belong to five sections, students from sections "MP*1" (green) and "MP*2" (pink) were maths majors, and those from "PC" (blue) and "PC*" (orange) were Physics majors, and lastly, students from "PSI" (dark green) were engineering majors [10]. In $\mathcal{G}^{\text{cond}}$ (see Figure 7), we find a super-node containing nodes from MP*1 (green) and MP*2 (pink) sections (enlarged pink node) and another super-node (enlarged orange node) with nodes from remaining three sections PC (blue), PC* (orange), and PSI (dark green). Our result is supported by [10], which mentions that the five classes in the dataset can brodaly be divided into two categories of (MP*1 and MP*2) and (PC, PC*, and PSI). The groupings in the super-nodes are intuitive as it is clear from the visualization itself that the dataset can broadly be divided into two components (MP*1 and MP*2) and (PC, PC*, and PSI). We also see that unmerged nodes in each major connect densely every day. These connections are obscured by unimportant nodes and edges in the original network. This dense inter-connection of "important" nodes is obscured by unimportant nodes and edges in the original network. However, visualization of the condensed network emphasizes on these important structures and how they remain stable over time. We also noted that small super-nodes of size 2 or 3, were composed solely of male students, which can be attributed to the gender homophily exhibited only by male students in the dataset [10]. We noticed that weekends were merged together with surrounding days in `School`, while other days remained intact.

`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). Other unmerged nodes included Vice-Presidents and Managing Directors. 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 various Vice-Presidents, L. Blair (Director), and S. Horton (President) for seven time-stamps 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.

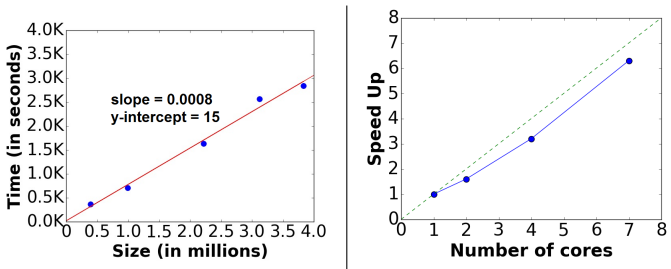
To investigate the nature of nodes which get merged early, we look at the super-nodes at the early stage of NETCONDENSE, we find two super-nodes with one Vice-President in each (Note that most other nodes are still unmerged). Both super-nodes were composed mostly of Managers, Traders, and Employees who reported directly or indirectly to the mentioned Vice-Presidents. We also look at unmerged time-stamps and notice that time-stamps in 2001 were never merged. Even though news broke out on October 2001, analysts were suspicious of practices in Enron Corporation from early 2001[6]. Therefore, our time-merges show that the events in early 2001 were important indicative of suspicious activities in Enron Corporation.

`DBLP`: These are co-authorship networks from DBLP-CS bibliography. 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, whereas super-nodes grouped researchers who had only few publications. In super-nodes, we found researchers from the same institutions or countries (like Japanese Universities) or same or closely-related research fields (like Data Mining/Visualization). In larger super-nodes, we found that researchers were grouped together based on countries and broader fields. For example, we found separate super-nodes composed of scientists from Swedish, and Japanese Universities only. We also found super-nodes composed of researchers from closely related fields such as Data Mining and Information visualization. Small super nodes typically group researchers who have very few collaborations in the dataset, collaborated very little with the rest of the network and have edges among themselves in few time-stamps. Basically, these are researchers with very few publications. We also found a giant super-node of size $395,000$. An interesting observation is that famous researchers connect very weakly to the giant super-node. For example, Rakesh Aggarwal connects to the giant super-node in only two time-stamps with almost zero edge-weight. Whereas, less known researchers connect to the giant super-node with higher edge-weights. This suggests researchers exhibit homophily in co-authorship patterns, i.e. famous researchers collaborate with other famous researchers and non-famous researchers collaborate with other non-famous researchers. Few super-nodes that famous researchers connect to at different time-stamps, also shows how their research interest has evolved with time. For example, we find that Philip S. Yu connected to a super-node of IBM researchers who primarily worked in Database in early 1994 and to a super-node of data mining researchers in 2000.

5.6 Scalability and Parallelizability

Figure 8 (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 8 (b) shows the *near-linear* run-time speed-up of parallel-NETCONDENSE vs # cores on `Wikipedia`.

6. http://www.webcitation.org/5tZ26rnac



(a) Scalability (b) Parallelizability

Fig. 8: (a) Near-linear Scalability w.r.t. size; (b) Near-linear speed up w.r.t number of cores for parallelized implementation.

6 Related Work

Mining Dynamic Graphs. Dynamic graphs have gained a lot of interest recently (see [2] for a survey). Many graph mining tasks on static graphs have been introduced to dynamic graphs. For example, Tantipathananandh et.al studied community detection in dynamic graphs [35]. Similarly, link prediction [32], and event detection [30] have also been studied in dynamic graphs. Due to the increasing size, typically it is challenging to perform analysis on temporal networks and summarizing large temporal networks can be very useful.

Propagation. Cascade processes have been widely studied, including in epidemiology [5], [14], information diffusion [17], cyber-security [18] and product marketing [31]. Based on the models, several papers studied propagation related applications, such as propagation of memes [20] and so on. In addition, there are research works that focus on the epidemic threshold of static networks, which determines the conditions under which virus spreads throughout the network [6], [7], [11], [18], [23], [25], [26]. Recently [27] studied the threshold for temporal networks based on the system matrix. Examples of propagation-based optimization problems are influence maximization [3], [13], [17], and immunization [40]. Remotely related work deals with weak and strong ties over time for diffusion [16].

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 [24], [36], node/edge attributes [39], action logs [29], eigenvalue of the adjacency matrix [28], and key subgraphs. It is also related to graph sparsification algorithms [22]. 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 [21], or extracting a list of recurrent sub-structures over time [33]. Unlike these, we are the first to focus on hierarchical condensation: using *structural* merges, giving a *smaller propagation-equivalent* temporal network. There has been some work on summarizing temporal graphs, including compression-based Liu et. al. [21] tackled the problem by compressing edge weights at each timestamps. However, their method does not merge either nodes or timestamps. Hence, it cannot get a summary with much less nodes and timestamps. Shah et. al. [33] proposed a temporal graph summarization method (Time Crunch) [33], by extracting representative local structures across timestamp. However, Time Crunch do not work for our problem, as our problem requires the summary to be a compact temporal graph that maintains the diffusion property, while Time Crunch just outputs pieces of subgraphs. To summarize, none of the above papers focused on the temporal graph summarization problem w.r.t. diffusion.

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. We leverage it to dramatically speed-up influence maximization and event detection algorithms on a variety of large temporal networks. We also leverage the summary network given by NetCondense to visualize, explore, and understand multiple networks. 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 such as SIS, SIR, and so on.

There are multiple ideas to explore further. Condensing attributed temporal networks, and leveraging NetCondense for other graph tasks such as role discovery, immunization, and link prediction are some examples. We leave these tasks for future works.

Acknowledgements We would like to thank anonymous reviewers for their suggestions. This paper is based on work partially supported by the National Science Foundation (IIS-1353346), the National Endowment for the Humanities (HG-229283-15), ORNL (Task Order 4000143330) and from the Maryland Procurement Office (H98230-14-C-0127), and a Facebook faculty gift. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the respective funding agencies.

References

[1] L. A. Adamic and B. A. Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000.
[2] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Survey*, 2014.
[3] C. C. Aggarwal, S. Lin, and S. Y. Philip. On influential node discovery in dynamic social networks. In *SDM*, 2012.
[4] S. E. Amiri, L. Chen, and B. A. Prakash. Snapnets: Automatic segmentation of network sequences with node labels. 2017.
[5] R. M. Anderson and R. M. May. *Infectious Diseases of Humans*. Oxford University Press, 1991.
[6] N. T. Bailey et al. The mathematical theory of epidemics. 1957.
[7] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *TISSEC*, 10(4):1, 2008.
[8] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson. An estimate for the condition number of a matrix. *SIAM Journal on Numerical Analysis*, 16(2):368–375, 1979.
[9] 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.
[10] J. Fournet and A. Barrat. Contact patterns among high school students. *PloS one*, 9(9):e107878, 2014.
[11] A. Ganesh, L. Massoulie, and D. Towsley. The effect of network topology in spread of epidemics. *IEEE INFOCOM*, 2005.
[12] F. R. Gantmacher and J. L. Brenner. *Applications of the Theory of Matrices*. Courier Corporation, 2005.

[13] N. T. Gayraud, E. Pitoura, and P. Tsaparas. Diffusion maximization in evolving social networks. In *COSN*, 2015.
[14] H. W. Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.
[15] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
[16] M. Karsai, N. Perra, and A. Vespignani. Time varying networks and the weakness of strong ties. *Scientific Reports*, 4:4001, 2014.
[17] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.
[18] J. O. Kephart and S. R. White. Measuring and modeling computer virus prevalence. In *IEEE Research in Security and Privacy*, 1993.
[19] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.
[20] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD09*, 2009.
[21] 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.
[22] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537. ACM, 2011.
[23] A. M'Kendrick. Applications of mathematics to medical problems. *Proceedings of the Edinburgh Mathematical Society*, 44:98–130, 1925.
[24] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.
[25] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.
[26] B. A. Prakash, D. Chakrabarti, M. Faloutsos, N. Valler, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In *ICDM*, 2011.
[27] 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.
[28] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. Subrahmanian. Fast influence-based coarsening for large networks. In *KDD14*.
[29] Q. Qu, S. Liu, C. S. Jensen, F. Zhu, and C. Faloutsos. Interestingness-driven diffusion process summarization in dynamic networks. In *ECML/PKDD*. 2014.
[30] S. Rayana and L. Akoglu. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *SDM*, 2015.
[31] E. M. Rogers. *Diffusion of innovations*. 2010.
[32] P. Sarkar, D. Chakrabarti, and M. Jordan. Nonparametric link prediction in dynamic networks. In *ICML*, 2012.
[33] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD15*.
[34] G. W. Stewart. *Matrix perturbation theory*. 1990.
[35] C. Tantipathananandh and T. Y. Berger-Wolf. Finding communities in dynamic social networks. In *ICDM11*.
[36] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *KDD*, 2011.
[37] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012.
[38] F. Zhang. *Matrix theory: basic results and techniques*. Springer Science and Business Media, 2011.
[39] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *ICDE*, 2010.
[40] Y. Zhang and B. A. Prakash. Dava: Distributing vaccines over networks under prior information. In *SDM*, 2014.

Bijaya Adhikari received the bachelor's degree in computer engineering from Vistula University, Warsaw, Poland. He is working towards the Ph.D. degree in the Department of Computer Science, Virginia Tech. His current research includes data mining large networks, temporal networks analysis, and network embedding with focus on propagation in networks. He has published papers in the SDM conference.

Yao Zhang is a PhD candidate in the Department of Computer Science at Virginia Tech. He got his Bachelor and Master degrees in computer science from Nanjing University, China.His current research interests are graph mining and social network analysis with focus on understanding and managing information diffusion in networks. He has published several papers in top conferences and journals such as KDD, ICDM, SDM and TKDD.

Sorour E. Amiri received the bachelor's degree in computer engineering from Shahid Beheshti University and master's degrees in Algorithms and Computation from the University of Tehran, Iran. She is working toward the Ph.D. degree in the Department of Computer Science, Virginia Tech. Her current research interests include graph mining and social network analysis with a focus on segmentation of graph sequences and summarizing graphs. She has published papers in AAAI conference and ICDM workshops.

Aditya Bharadwaj received the bachelor's degree in computer science from Birla Institute of Technology and Science, Pilani, India. He is currently working towards the Ph.D. degree in the Department of Computer Science, Virginia Tech. His current research includes data mining large networks, analyzing network layouts and computational system biology.

B. Aditya Prakash is an Assistant Professor in the Computer Science Department at Virginia Tech. He graduated with a Ph.D. from the Computer Science Department at Carnegie Mellon University in 2012, and got his B.Tech (in CS) from the Indian Institute of Technology (IIT) – Bombay in 2007. He has published more than 50 refereed papers in major venues, holds two U.S. patents and has given three tutorials (SIGKDD 2016, VLDB 2012 and ECML/PKDD 2012) at leading conferences. His work has also received a best paper award and two best-of-conference selections (CIKM 2012, ICDM 2012, ICDM 2011) and multiple travel awards. His research interests include Data Mining, Applied Machine Learning and Databases, with emphasis on big-data problems in large real-world networks and time-series. His work has been funded through grants/gifts from the National Science Foundation (NSF), the Department of Energy (DoE), the National Security Agency (NSA), the National Endowment for Humanities (NEH) and from companies like Symantec. He received a Facebook Faculty Gift Award in 2015. He is also an affiliated faculty member at the Discovery Analytics Center at Virginia Tech. Aditya's homepage is at: http://www.cs.vt.edu/ badityap.