Task-specific Summarization of Networks: Optimization and Learning

Sorour E. Amiri

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

B. Aditya Prakash, Chair
T M Murali
Christopher L. North
Chandan K. Reddy
Hanghang Tong

April 26, 2019
Blacksburg, Virginia

Keywords: Graph mining, Sensemaking, Task-specific summarization, Learning to summarize, Interactive visualization, Reinforcement learning, Spectral representation

Task-specific Summarization of Networks: Optimization and Learning

Sorour E. Amiri

ABSTRACT

Networks (also known as graphs) are everywhere. People-contact networks, social networks, email communication networks, internet networks (among others) are examples of graphs in our daily life. The increasing size of these networks makes it harder to understand them. Instead, summarizing these graphs can reveal key patterns and also help in sensemaking as well as accelerating existing graph algorithms. Intuitively, different summarizes are desired for different purposes. For example, to stop viral infections, one may want to find an effective policy to immunize people in a people-contact network. In this case, a high-quality network summary should highlight roughly structurally important nodes. Others may want to detect communities in the same people-contact network, and hence, the summary should show cohesive groups of nodes. This implies that for each task, we should design a specific method to reveal related patterns. Despite the importance of task-specific summarization, there has not been much work in this area.

Hence, in this thesis, we design task-specific summarization frameworks for univariate and multivariate networks. We start with optimization-based approaches to summarize graphs for a particular task and finally propose general frameworks which automatically learn how to summarize for a given task and generalize it to similar networks.

1. **Optimization-based approaches:** Given a large network and a task, we propose summarization algorithms to highlight specific characteristics of the graph (i.e., structure, attributes, labels, dynamics) with respect to the task. We develop effective and efficient algorithms for various tasks such as content-aware influence maximization and time segmentation. In addition, we study many real-world networks and their summary graphs such as people-contact, news-blogs, etc. and visualize them to make sense of their characteristics given the input task.

2. **Learning-based approaches:** As our next step, we propose a unified framework which learns the process of summarization itself for a given task. First, we design a generalizable algorithm to learn to summarize graphs for a set of graph optimization problems. Next, we go further and add sparse human feedback to the learning process for the given optimization task.

To the best of our knowledge, we are the first to systematically bring the necessity of considering the given task to the forefront and emphasize the importance of learning-based approaches in network summarization. Our models and frameworks lead to meaningful discoveries. We also solve problems from various domains such as epidemiology, marketing, social media, cybersecurity, and interactive visualization.

Task-specific Summarization of Networks: Optimization and Learning

Sorour E. Amiri

GENERAL AUDIENCE ABSTRACT

Networks (also known as graphs) are everywhere. People-contact networks, social networks, email communication networks, internet networks (among others) are examples of graphs in our daily life. The increasing size of these networks makes it harder to understand them. Instead, summarizing these graphs can reveal key information and also help in sensemaking as well as accelerating existing graph analysis methods. Intuitively, different summarizes are desired for different purposes. For example, to stop viral infections, one may want to find an effective policy to immunize people in a people-contact network. In this case, a high-quality network summary should highlight roughly important nodes. Others may want to detect friendship communities in the same people-contact network, and hence, the summary should show cohesive groups of nodes. This implies that for each task, we should design a specific method to reveal related patterns. Despite the importance of task-specific summarization, there has not been much work in this area.

Hence, in this thesis, we design task-specific summarization frameworks for various type of networks with different approaches. To the best of our knowledge, we are the first to systematically bring the necessity of considering the given task to the forefront and emphasize the importance of learning-based approaches in network summarization. Our models and frameworks lead to meaningful discoveries. We also solve problems from various domains such as epidemiology, marketing, social media, cybersecurity, and interactive visualization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivation. Large graphs appear everywhere, as they capture relations between objects very well [50]. For example, social networks [88], co-purchased product networks [143], people-contact networks [27], and protein interaction graphs [49] are all instances of large graphs in the real-world. Analyzing these graphs is common among many applications in different fields such as cybersecurity [177], recommendation systems [94], sociology [89], biology [160], and so on. For example, analyzing network traffic data for making sense of it and detecting malware is critical in cybersecurity. One common way to perform such sensemaking tasks is by visualizing networks effectively. However, the increasing size of such networks makes performing fast computations on them challenging [109, 130]. The large size also affects the quality of visualizations and our ability to highlight important characteristics [30]. How can we alleviate these problems? One way is to design faster and more efficient algorithms to scale up existing graph algorithms. Another approach is to summarize graphs in order to solve given problems on smaller ones and then map the results back [130, 87]. A summary graph G_s is a smaller meaningful representation of the original graph G [109, 7]. Summarization has several advantages in addition to speeding up the existing algorithms; it compresses large data and reduces the storage size and also helps in sensemaking and visualization of graphs. Here, we investigate such approaches and formulate the novel problem of task-specific graph summarization.

Summary graph. As mentioned above having a summary graph can intuitively help in both scaling up existing algorithms and making sense of the original network [109, 95]. For example, consider a task of finding friendship communities in a social network, a very common task in the graph mining area [77]. Figure 1.1 (a) shows the well-known social network of a university karate club [66]. Nodes are karate club members and edges show their interaction outside of the club. There are two clear ground-truth communities in this network which show the members clustering around two main instructors in the karate club. Members of the same communities frequently interact with each other while they rarely interact with members of another community. A summary graph that groups nodes

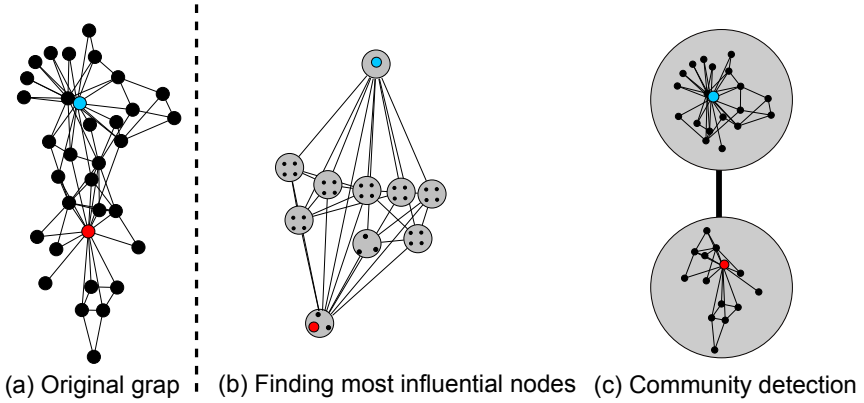


Figure 1.1: Summarizing KARATE CLUB network [66] for (a) finding the most influential nodes and (b) community detection tasks. Note that each summary highlights a particular characteristic of the network that is related to the given task.

into "supernode" and identifies the relations between them by "superedges" can be useful to highlight such ground-truth communities in this network. As we can see in Figure 1.1 (c), such summary graph groups nodes in the same community matching the ground-truth explained above and therefore helps in highlighting them quickly. Furthermore, the structure of the summary graph will also give a high-level understanding of the relations between different communities.

Effect of the given task. Intuitively, different tasks produce different network summaries. As mentioned above, when the goal is to find the friendship communities in a social network (Figure 1.1 (c)) the summary should highlight different communities in the graph [66] and the relation between them. On the other hand, if we want to find most influential people in the same network (Say for viral marketing [100]), the summary graph must highlight the central nodes instead. As shown in Figure 1.1 (a), the blue and the red nodes are the two influential instructors in the karate club network. In the summary graph in Figure 1.1 (b) these nodes are in two different supernodes with the highest degree; which makes them easy to detect and highlights their role in the network. Looking at these two examples, we contend that generating different summaries for different given tasks is a meaningful research question. We will provide more context about this line of research later on. We call such summaries "task-based network summarizations" and show that they have many real-world applications in cybersecurity, sensemaking, biology, and so on.

Previous work. Despite the usefulness and importance of task-based network summarization, there is surprisingly not much work in this area. Most prior work focuses on task independent summarization techniques and consider the summary graph G_s as a function of G only (i.e., $G_s = f(G)$) [32, 178] and as a result, for several popular tasks such as visualization and combinatorial problems, ready-made summarization methods do not exist. Some try to develop specific algorithms for specific tasks [130]. However, we explore promising alternative approaches to generate summaries as a function of both original networks and

the given task (i.e., $G_s = f(G, T)$) instead and propose algorithms for a more extensive set of problems and networks.

Our focus. In this thesis, we focus on summarizing large graphs and graph sequences based on a given task. We look at both univariate and multivariate networks. *Univariate* networks are networks in which nodes are unlabeled or binary labeled. For example, we can consider a social network of students in a university setting as a univariate network with unlabeled nodes. Also, by including the information is a student is a male/female make it a univariate network with binary labeled nodes. On the other hand, *multivariate* networks are networks which have attributed nodes/edges. For example, in the above student-network, if we take the student's gender, major, age, etc. into account, the network becomes a multivariate one.

Questions we try to answer include: Given a large temporal network, can we summarize it and find time segments when the pattern of node properties or the structure of the network changes dramatically? Given a large graph can we quickly detect anomalous/ influential nodes in it? Can we generate high-quality visualization by summarizing large networks? As mentioned before, in contrast to the existing literature, our summarization algorithms are tailored to the given tasks and vary based on the task goals.

For many tasks such as detecting influential nodes, community detection, and anomaly detection [132], a natural way to summarize networks is to optimize a particular objective to maximize the quality of the solution of the given task. Hence, we first explore the above questions using optimization-based approaches. In these approaches, we summarize the network by optimizing an explicit objective function to maintain a particular characteristic of the network related to the given task. For example, if we are given a community detection task on a social network, our objective is to minimize the normalized cut or maximize the connection of grouped nodes in each supernode of the summary graph [77]. Our key novelty in optimization approaches is to carefully formulate the objective for each given task and design effective and scalable sub-quadratic algorithms to generate summary networks that optimize those objectives leveraging techniques form approximation combined with linear algebra.

In many cases such as in visualization and sensemaking [125, 69], the given task is very complicated. For example, consider visualizing a network of documents where nodes are documents, and edges show their similarity. Such a task requires drawing each document in a 2-dimensional space. In such tasks, we may consider many criteria such as putting similar documents closer to each other, reflecting users topical interests, highlighting the underlying story in documents, and many more [31]. Many of these criteria are not well defined and depend on the data, task, and user itself. Also, these criteria may sometimes have conflicts, and it is incredibly complicated to write a unified objective for summarization. So we may not be able to explicitly formulate the objective function or pick the characteristic we want to maintain. Furthermore, there are cases where we want to solve the same problem on similar graphs many times. Hence for such cases, we introduce a more flexible approach with the goal of *learning* to summarize graphs. We propose, and design novel learning

approaches to summarize networks to answer the above questions which we call "learning-based network summarization". Our key novelty in such approaches is in generalizing the formulation of different summarization tasks to a uniform learning problem and in the way we incorporate human feedback into the learning process with the overall goal of generating high-quality summaries. In the following chapters, we study these task-specific summarization problems, using optimization-based and learning-based approaches, and with different types of networks.

In summary, we highlight the importance of considering the given task and learning approaches in network summarization. We study different task-based network summarization approaches for various types of networks with different methods. This chapter consists of the thesis overview, a summary of the completed work, and possible future directions.

1.1 Thesis Overview

In this section, we present our thesis statement and briefly explain our main work.

1.1.1 Thesis Statement

Task-specific network summaries can be generated in an efficient and generalizable way which can effectively support diverse applications from a variety of domains.

In the above statement, we focus on tasks motivated by cybersecurity, epidemiology, human-computer interaction, and sociology domains. More specifically we focus on applications such as sensemaking, identifying malware and anomalies, event detection, solving combinatorial problems, and interactive visualization. For example, we show that summarizing networks for the anomaly detection tasks can reveal the malicious requests in computer networks which is crucial for malware detection in the cybersecurity domain. Also, we show that summarization for interactive visualization task in security domains helps in sensemaking and revealing suspicion activities in intelligence reports. Our designed algorithms are sub-quadratic and parallelizable. We leverage deep reinforcement learning [150, 115] in our learning-based approaches which makes it possible to re-apply the learned models to unseen networks. Our methods provide high-quality solutions for the aforementioned tasks and outperform state-of-the-art competitors.

1.1.2 Thesis Structure

Following the thesis statement, we organize the thesis into two parts: optimization-based and learning-based approaches. The outline is shown in Table 1.1.

Table 1.1: Structure of the thesis with references to the chapters.

	Part I: Optimization-based approaches	**Part II:** Learning-based approaches
Univariate networks	• Unlabeled network sequences (Chapter 3) • Binary-labeled network sequences (Chapter 4)	• Learning to generate network summaries (Chapter 6)
Multivariate networks	• Attributed networks for influence based tasks (Chapter 5)	• Feedback-based summarization (Chapter 7)

As a first step, we tackle the task-based summarization problem using intuitive optimization-based approaches. We begin with univariate networks where nodes are unlabeled or binary labeled as they appear in many real-world scenarios such as in people-contact networks and epidemiology (Chapters 3 and 4). Next, to encode more information in more complicated real-world scenarios, we investigate multivariate networks (Chapter 5). In these chapters, we design effective sub-quadratic and parallelizable algorithms to summarize networks for the given tasks mentioned earlier in this section. In the next part of the thesis, we aim to summarize networks for more complicated tasks in a reusable fashion. We start with univariate graphs and develop a unified and reusable learning-based framework for any graph optimization task (Chapter 6). As mentioned before, graph optimization tasks include a large set of important graph mining problems such as influence maximization, community detection, anomaly detection, and so on. However, for more complicated tasks such as interactive visualizations and sensemaking [133, 152] we need more flexible methods that can incorporate human interests into the learning process. Hence, toward our goal to summarize multivariate networks, we design feedback based summarization methods which use human in the loop and integrate the goals of the given task with human feedback to generate meaningful summaries (Chapter 7).

1.2 Summary of Work

1.2.1 Part I: Optimization-based approaches

For each class of network, we propose optimization-based approaches to summarize it for a given task. We start with univariate graphs which include graphs with unlabeled nodes and graphs with binary labeled nodes. Next, we study multivariate graphs where nodes can have several real-valued attributes. We study how to obtain a smaller representation of a large network while maintaining the structural and attribute based properties of it.

Univariate networks

Chapter 3 (Unlabeled network sequences) [2]. In this chapter, we study the novel problem of generating a smaller diffusion-equivalent representation of a set of time-evolving networks with unlabeled nodes. 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 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. Figure 1.2 shows an example of a summary network that NetCondense generates on a temporal social-contact network between employees of a company. In the summarization, the supernodes with the same colors highlight different departments in the company. Also, the singleton nodes in the summarization reveal "Linker" nodes which are crucial for the epidemic spread in the network. The visualization of the summary network emphasizes that linkers connect consistently to nodes from multiple departments; which is not obvious in the original networks.

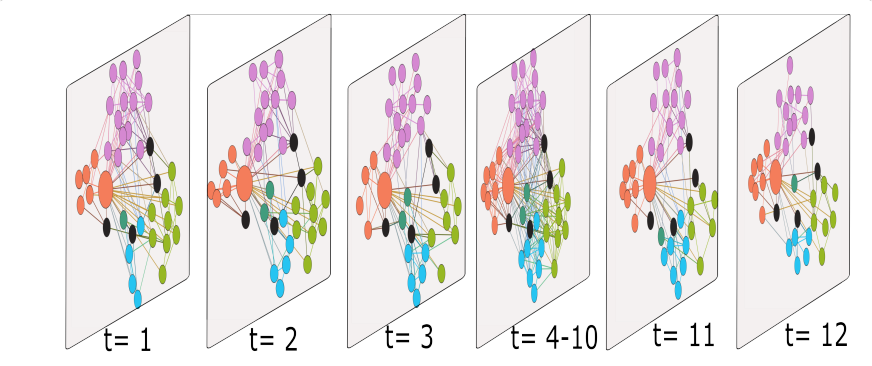


Figure 1.2: Summarization of an unlabeled temporal social-contact network between employees of a company [76] for an anomaly detection task. Note, the summarization highlights different departments in the company (supernodes with the same color) and reveals "Linkers" which are crucial for epidemic spread in the network (singleton nodes).

Chapter 4 (Binary-labeled network sequences) [14, 15, 13]. In the previous chapter, we focused on summarizing graph sequences with unlabeled nodes. In this chapter, we extend our work to networks with binary labeled nodes. The main question we answer in this chapter is: Given a sequence of snapshots of a network with evolving binary node labels, can we find a segmentation when the patterns of the labels change, possibly due to interventions? We

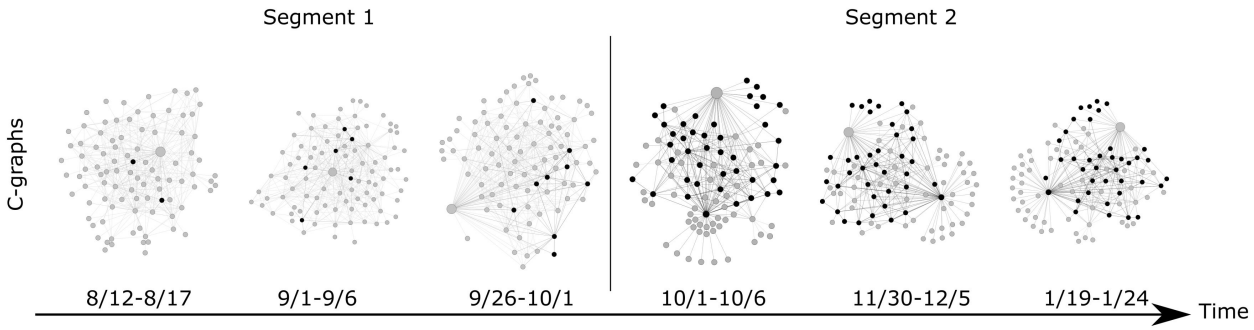


Figure 1.3: Summarization of a binary-labeled blog and website network sequence [101] for the event detection task. Note, the summarization helped in detecting a cut point in the network sequence on Oct 01, 2008 which matches the date of the televised debate between Joe Biden and Sarah Palin (the vertical line is the detected cut point).

study the problem of segmenting graph sequences with labeled nodes. Memes on the Twitter network, diseases over a contact network, movie-cascades over a social network, etc. are all graph sequences with labeled nodes.

Most related work on this subject is on plain graphs and hence ignores the label dynamics. Others require to fix parameters or feature engineering. We propose SnapNETS, to automatically find segmentations of such graph sequences, with different characteristics of nodes of each label in adjacent segments. It satisfies all the desired properties (being parameter free, comprehensive and scalable) by leveraging a principled, multilevel, flexible framework which maps the problem to a path optimization problem over a weighted DAG. Also, we develop the parallel framework of SnapNETS which speeds up its running time. Finally, we propose an extension of SnapNETS to handle the dynamic graph structures and use it to detect anomalies (and events) in network sequences.

Extensive experiments on several diverse real datasets show that it finds cut points matching ground-truth or meaningful external signals and detects anomalies outperforming non-trivial baselines. We also show that the segmentations are easily interpretable and that SnapNETS scales near-linearly with the size of the input. Finally, we show how to use SnapNETS to detect anomalies in a sequence of dynamic networks. Figure 1.3 illustrates an example of our results. It shows the summarization of the network of who-copies-from-whom in blogs and websites [101]. We consider a blog/website active if it adopts the phrase 'hey can I call you Joe'. We want to find how the adoption pattern changes. The summarization helped in detecting a cut point in the network sequence on Oct 01, 2008 which matches the date of the televised debate between Joe Biden and Sarah Palin.

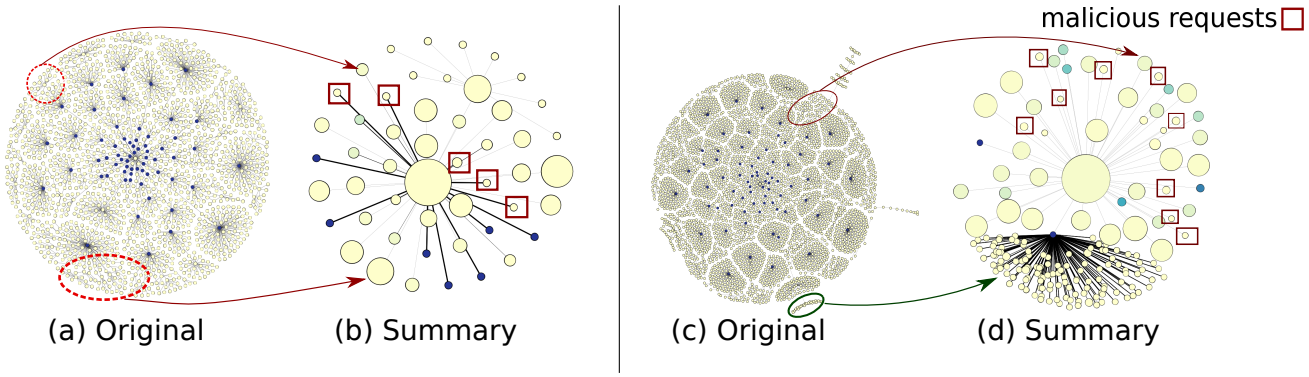


Figure 1.4: Summarization of an attributed computer network-traffic data [177] for the task of finding malicious requests. (a) is the original network traffic data of a user. Dark blue nodes are type 'user-input' and yellow ones are type 'network-request'. (b) shows the summary graph generated by ANeTS. The summarization highlights the malicious requests in the network. They are the singleton nodes with red squares.

Multivariate networks

Chapter 5 (Summarization of attributed networks for influence based tasks) [16] Chapter 4 and 3 study the summarization of univariate networks. However, in most of the real networks, nodes have attribute and content. For example, in a social network, each user has particular characteristics and interests. In this chapter, the main question we answer is: Given a large attributed network, can we find a compact, diffusion equivalent representation while keeping the attribute properties? Diffusion networks with user attributes such as friendship, email communication, and people contact networks are increasingly commonplace in the real-world. However, analyzing them is challenging due to their large size. We first formally formulate a novel problem of summarizing an attributed diffusion graph to preserve its attributes and influence-based properties. Next, we propose ANeTS, an effective sub-quadratic parallelizable algorithm to solve this problem: it finds the best set of candidate nodes and merges them to construct a smaller network of 'supernodes' preserving the desired properties. Extensive experiments on diverse real-world datasets show that ANeTS outperforms all state-of-the-art baselines (some of which do not even finish in 14 days). Finally, we show how ANeTS helps in multiple applications such as Topic-Aware viral marketing and sensemaking of diverse graphs from different domains. Figure 1.4 shows an example of our summarization on a computer network-traffic data for finding malicious requests. It is hard to recognize malicious requests by simply inspecting the original graphs. However, our summaries highlight these anomalous nodes, matching the ground-truth [177] and also showing that they are structurally similar to user inputs–Figure 5.1(b) (which makes them hard to find in the first place). Hence, our summary helps provide structural evidence for system and network assurance and is useful for human experts cognition, and decision making in cybersecurity.

1.2.2 Part II: Learning-based approaches

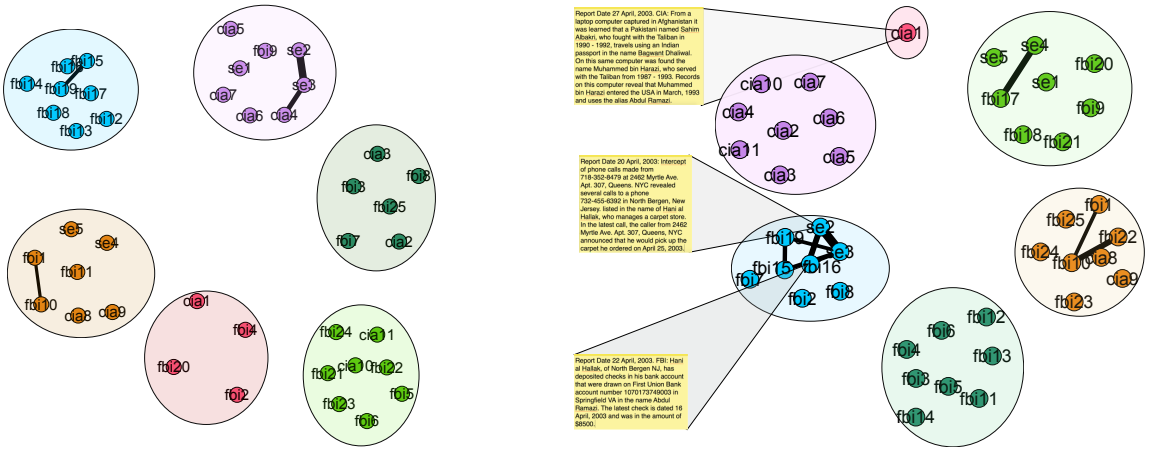
This part of the thesis is devoted to get a deeper understanding of the summarization task and propose more advanced methods to learn the summarization approach. A better approach to obtain summary for multiple tasks is to define a unified framework that can automatically 'learn' to generate task-specific summaries. This way, we can generate summaries for any given task, using our learned approach. Another advantage of such a 'meta-algorithm' is that once the model learns how to summarize a network, the same model can be applied to summarize graphs with similar characteristics.

Univariate networks

Chapter 6 (Learning to generate network summaries for graph optimization problems) [10, 11] Given a network, can we visualize it for any given task, highlighting the important characteristics? Networks are widespread, and hence summarizing and visualizing them is of primary interest for many applications such as viral marketing, extracting communities and immunization. Summaries can help in solving new problems, in visualization, sensemaking, and many other goals. However, most prior work focuses on general structural summarization techniques or on developing specific algorithms for specific tasks. This is both tedious and challenging. As a result, for several popular tasks, there do not exist readymade summarization methods. In this work, we explore a promising alternative approach instead. We propose NetGist, a framework which automatically learns how to generate a summary for a given task on a given network. In addition to generating the required summary, this also allows us to reuse the learned process on other similar networks. We formulate a novel task-based graph summarization problem and leverage reinforcement learning to design a flexible framework for our solution. Via extensive experiments, we show that NetGist robustly and effectively learns meaningful summaries, and helps solve challenging problems (such as document network visualization), and aids in complex task-based sensemaking of networks. Figure 1.5 shows a layout given by our method using a simple post-processing approach; indeed, as will be discussed in Chapter 6, our method consistently generates better layouts than competitors.

Multivariate networks

Chapter 7 (Feedback-based summarization for text analysis problems) [12] Using human-in-the-loop techniques to generate high-quality visualizations is a challenging and important area which has seen growing research work in recent years. For example, for when analyzing large text datasets, a user can express her agreement/disagreement with the visualization via iterative feedback to improve its quality. However, to incorporate such human feedback in generating a high-quality visualization, we should answer several challenging questions: How should we integrate human input with the original objective of

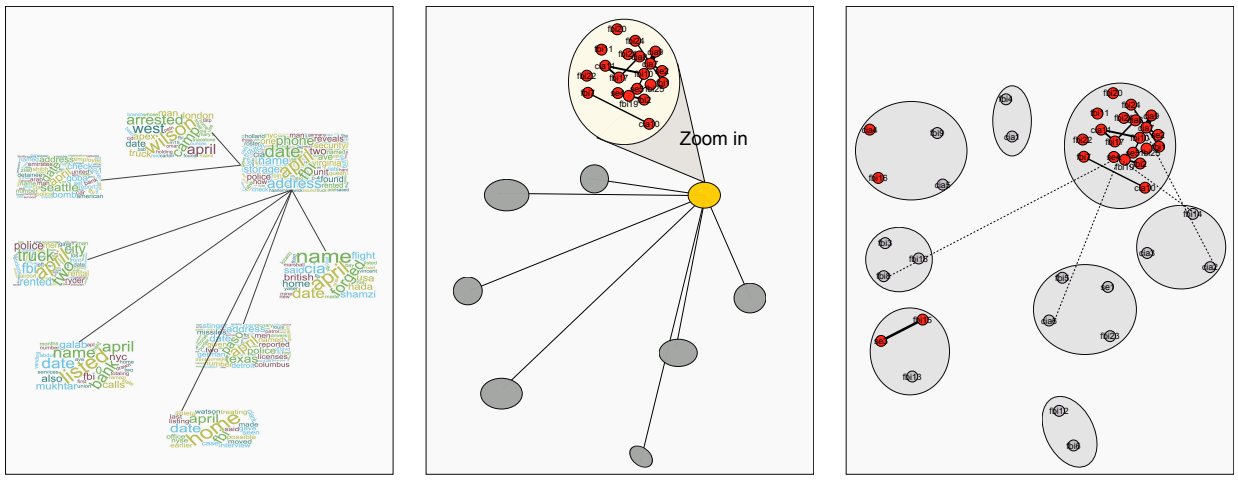


(a) Before considering human suggestions (b) Our framework

Figure 1.5: Visualization of CRESCENT, an intelligence-related document data [31] for a sensemaking task (a) before considering human suggestions and (b) after consideration by our summarization framework. Each node represents a document in the dataset. Edges show the satisfied user suggestions. Our framework satisfies user suggestions and gives a summary that reflects user interests.

the visualization? Sometimes human opinions can be inconsistent with the original objective of the visualization. How should we deal with such situations? Also, human feedback is typically sparse in the real-world, especially as the analysis task is usually exploratory, and the user only has a partial understanding of a final visualization. How should we generalize sparse user feedback?

We design NetReAct (Inter**act**ive **Net**work Summarization with **Re**inforcement learning), a novel interactive network algorithm for text corpora that enables a more comprehensible visualization by a multilevel approach. NetReAct incorporates human feedback with reinforcement learning to visualize document corpora. It converts such datasets to a network structure and summarizes it by grouping relevant documents and laying them out in groups close to each other. We show how NetReAct can help in generating high-quality visualization and revealing hidden patterns and giving an in-depth understanding of documents. We also show it can detect related documents and group them better than the competitors' algorithms. Figure 1.6 shows the final visualization of the CRESCENT dataset. NetReAct generates multilevel visualization and guides the user to quickly find documents relevant to the hidden story and avoid reading unnecessary ones. Note, the red nodes show documents relevant to the hidden story in the document corpus.



(a) Word-cloud visualization (b) Summary network visualization (c) Expanded visualization

Figure 1.6: Interactive visualization of the CRESCENT dataset using NetReAct for the task of revealing hidden scenarios behind intelligence reports. (a) Our method visualizes the summary document network and displays the word-cloud of the documents for each supernode as its representation. (b) It expands the supernode that the user selects and shows its documents for further investigation. (c) NetReAct can display all the documents and their hierarchical structure.

1.3 Contributions

To the best of our knowledge, we are the first to systematically considered tasks as a crucial component when it comes to network summarization and introduced learning-based approaches as an important and novel track to achieve such task-specific methods. We study different task-based network summarization approaches for various types of networks with different methods. Through extensive experiments, we show that our designed algorithms give meaningful summaries that help in solving real-world tasks and outperform competitors. We divide our contributions into three main aspects of theoretical, algorithmic, and application-centric as follows:

- **Theoretical.** We are the first to formally define several novel problems on task-specific summarization for different types of networks and tasks.
  - *Chapter 3 and 4*: We define the problem of summarizing temporal networks using the fundamental so-called 'system matrix'. Also, we formulate the problem of finding the best segmentation in a sequence of networks in an automatic and parameter-free way.
  - *Chapter 5*: We define the problem of summarizing attributed networks by carefully integrating the structural and attribute properties of the networks using spectral

characteristics.

- *Chapter 6*: We propose the novel general problem of learning task-specific summarization to automatically learn how to generate task-specific summaries for a set of similar networks.
- *Chapter 7*: We extend the problem definition in Chapter 6 for using human-in-the-loop and interactively learning to summarize multivariate networks (Chapter 7).

- **Algorithmic.** To solve the proposed novel problems we designed effective, scalable and parallelizable algorithms that outperform competitors.
  - *Chapter 3 and 4*: We present effective, model-agnostic, near-linear and parallelizable algorithms NetCondense and SnapNETS to summarize and segment unlabeled and binary labeled temporal networks.
  - *Chapter 5*: We proposed ANeTS, a new unsupervised, sub-quadratic and parallelizable algorithm which gets high-quality summaries of attributed influence graphs, in *near-linear* time in practice (in contrast to non-trivial competitors).
  - *Chapter 6*: We proposed an effective and re-usable method NetGist by leveraging the deep Q-learning framework. As shown by our experiments on both real and synthetic data, NetGist is able to learn meaningful summaries for various tasks like *Influence maximization* and *Community detection* even when we reduce the network size by 90%.
  - *Chapter 7*: We proposed a novel and effective algorithm NetReAct which leverages a feedback-based reinforcement learning approach to incorporate human input as well as the visualization task to produce high-quality network summaries.

- **Application-centric.** We show the usefulness of our proposed algorithms in various real-world applications such as anomaly, malware and event detection, sensemaking, and visualization. We also show that our summarization algorithms help in speeding up many current graph mining algorithms.
  - *Chapter 3 and 4*: Our proposed approaches dramatically speed-up influence maximization, event detection and anomaly detection algorithms (i.e., upto $48X$) on people-contact, social-contact, blog/website and social networks. We also leverage the summary network given by NetCondense and SnapNETS to visualize, explore, and understand these networks.
  - *Chapter 5*: Our proposed algorithm ANeTS speeds-up the TIM task ($\sim 20X$) and detects malware in network-traffic data; and it intuitively highlights structurally and attributed homogeneous regions in the network—helping in sensemaking of complex network datasets.

- *Chapter 6*: Our proposed frameworks NetGist and Guided-NetGist develop practical new algorithms for challenging open problems such as GUIDED-TBNS and sense-making via visualization. Also, since they are entirely automatic and reusable, they have a significant impact in speeding up creating high-quality visualization for multiple document datasets.
- *Chapter 7*: Our extensive experiments show that NetReAct is able to summarize and visualize document networks meaningfully to reveal hidden stories in document corpora and connect the dots between different documents. We are also planning on aggregating NetReAct framework with StarSPIRE [31] to develop a novel text analysis tool.

1.4 Publications

The following are my publications related to the thesis.

1. **Sorour. E. Amiri**, Bijaya Adhikari, John Wenskovitch, Michelle Dowling, Chris North, B. Aditya Prakash. NetReAct: Learning network visualizations for text analytics using interactions (Under review).
2. **Sorour. E. Amiri**, Bijaya Adhikari, Michelle Dowling, John Wenskovitch, Chris North, B. Aditya Prakash. Learning to Generate Effective Network Summaries (Under review).
3. **Sorour. E. Amiri**, Anika Tabassum, E. Thomas Ewing, B. Aditya Prakash. Tracking and analyzing dynamics of newscycles during global pandemics: a historical perspective (Under review).
4. **Sorour. E. Amiri**, B. Adhikari, A. Bharadwaj, B. A. Prakash. NetGist: Learning to generate task-based network summaries, *In IEEE 18th International Conference on Data Mining (ICDM)* 2018. [URL]
5. **Sorour E. Amiri**, Liangzhe Chen and B. Aditya Prakash. Efficiently summarizing attributed diffusion, *in Data Mining and Knowledge Discovery, pages 1-24,* 2018. [URL]
6. B. Adhikari, Y. Zhang, **Sorour. E. Amiri**, A. Bharadwaj, and B. A. Prakash. Propagation-based temporal network summarization, *in IEEE Transactions on Knowledge and Data Engineering, 30(4):729-742,* 2018. [URL]
7. **Sorour E. Amiri**, Liangzhe Chen and B. Aditya Prakash. Automatic segmentation of dynamic network sequences with node labels, *in IEEE Transactions on Knowledge and Data Engineering, 30(3):407-420,* 2018. [URL]

8. Liangzhe Chen, **Sorour E. Amiri**, and B. Aditya Prakash. Automatic Segmentation of Data Sequences, *in the 32st AAAI Conference on Artificial Intelligence (AAAI),* 2018, Orleans, Louisiana, USA. [URL]

9. **Sorour E. Amiri**, Liangzhe Chen and B. Aditya Prakash. Automatic Segmentation of Network Sequences with Node Labels, *in the 31st AAAI Conference on Artificial Intelligence (AAAI),* 2017, San Francisco, USA. [URL]

10. **Sorour E. Amiri**, Liangzhe Chen and B. Aditya Prakash. Segmenting sequences of node-labeled graphs, *in IEEE 16th International Conference on Data Mining Workshops (ICDMW),* 2016, Barcelona, Spain. [URL]

1.5 Outline of the Thesis

The rest of the thesis is organized as follows. We first give a survey of the related work in Chapter 2, next present our work in Chapters 4, 3, 5 (Optimization-based approaches), and Chapters 6 and 7 (Learning-based approaches), and finally we conclude in Chapter 8 and introduce the research questions, contributions, and publications of our work.

Chapter 2

Survey

In this section, we survey the related work in multiple areas, including, static and dynamic graph summarization, graph visualization, and deep-learning approaches. More specifically, We divide the related work into several buckets. First, we study the general graph mining. It helps us to understand the relevant research problems in this area. Then, we explore the graph summarization methods for different types of graphs to get a thorough understanding of the state-of-the-art approaches in graph summarization. Also, we study the graph drawing and visualization area as it is one of our primary applications. Finally, we review the deep learning approaches on graphs as it is a recently gaining much attention in the graph mining and graph summarization area.

2.1 General Graph mining

Graph mining is a novel approach to analyze structured datasets which gained much attention in the last decades [9, 54, 41]. Graph mining tackles problems such as frequent pattern mining [170, 157], classification [142], compression [32] and many more. Also, it has many applications in various fields such as cybersecurity [177], recommendation systems [94], sociology, biology [160], etc.

2.1.1 Community detection and node clustering

There is no clear general agreement on the definition of the communities on graphs. Usually, communities are viewed as a set of subgraphs with dense internal connections and sparse external connections. There has been much work on this area [46, 58, 119, 120]. One of the most important techniques to identify communities is graph partitioning algorithms which include hierarchical clustering [118, 120, 139] and spectral partitioning [57, 77]. The

next popular approach to detect communities is measuring modularity metric of communities [28, 119]. Also, there has been some work on overlapping communities. [6, 102, 114, 123, 173].

There is an increasing interest in finding communities for graphs with content/attributes [169, 124, 7]. Many different approaches have been taken for detecting communities for graphs (see [29, 166] for a review). Yang et al. [174] model interactions between network structure and node attributes to find overlapping communities. Some methods combine structural and attribute similarity through a unified distance, measure to find communities [134, 181]. These algorithms only give clusters and not diffusion/attribute equivalent summary graphs as we do.

2.2 Graph summarization

The graph summarization problem aims to find a compact representation of a large graph considering its global and local properties. Plain graph, attributed graph, and dynamic graph summarization are the three main categories of this area.

2.2.1 Plain graph summarization

The methods in this category aim to find compact representations of graphs which maintain desired properties of a plain graph. The properties can be defined based on specific user queries [55], action logs [110], or, more generally, the encoding cost [108, 117], weights of nodes and edges [131, 154], influence-properties [130] and connectivity of nodes [154].

Summarizing networks is a popular problem in many domains (see survey in [109]). These algorithms help to reduce the processing cost of large graphs and maintain (sometimes amplify) the patterns in the graphs. METIS [87] and GRACLUS [46] summarize networks for better community detection. Navlakha et. al [117] propose a summarization technique for better compression. Similarly, Purohit et. al [130] propose a summarization technique for diffusion-related problems over networks. Toivonen et al. [154] find a summary of the graph by minimizing the error of the edge/path weights. Fan et al. [55] compress the graph such that the answers to a set of queries would remain the same. Other works include subgraphs-based methods [95] and node and edge attributes-based methods [131].

Our work is also related to network sparsification [110, 60]. Mathioudakis et al. [110] sparsify influence networks by removing edges and maximizing the likelihood of the action log. Unlike network sparsification where nodes and edges are removed from the network, we aim to summarize the network by merging nodes. Also, we add a new direction to this line of work, by aiming to *learn task-based* summaries automatically instead of *designing* a different algorithm for each one separately.

2.2.2 Attributed graph summarization

As pointed out in surveys [91, 109], only a few recent approaches are summarizing attributed graphs. [167] formulate the summarization as an information theory problem—however, their method is not diffusion consistent and scalable & parallel. Tian et al. [153] summarize attributed graph by grouping nodes based on user-selected attributed and relationships, and they allow users to navigate through summaries with different resolutions. Perozzi et al. [124] find both focused clusters and outliers from an input exemplar node set. Gunnemann et al. [74] introduce subspace clustering on graphs with feature values. Zhou et al. [180, 181] define a unified distance measure to find node clusters with dense connection and homogeneous attributes. Yang et al. [172] distinguish functional and structural community. These algorithms help reducing the processing cost of large graphs, and maintain (sometimes amplify) the patterns in the graphs. [153] and [140] restrict summary graphs to 'grouped' nodes with the *exact same* subset of attributes. In contrast, we tackle a more general problem by allowing grouping of nodes with *similar* attributes in principle. The most closely related work is the VEGAS algorithm [145] which summarizes influence flows from a source node. However, their pipeline is specifically designed for citation networks and is not scalable in practice.

2.2.3 Dynamic graph summarization

Dynamic graph summarization is gaining much interest because of the evolutionary nature of many networks we see nowadays (see [5] for an overview). Many traditional machine learning tasks on static graphs have been extended to dynamic ones, such as the clustering problem [92, 75], classification problem [3, 73], link prediction [138], anomaly detection [78], and trend mining [45]. Qu et al. [131] summarize dynamic networks by capturing only the most interesting nodes and edges over time. Liu et al. [108] compress weighted time-evolving graphs by minimizing the encoding cost.

There is also work finding time cut-points according to the change of patterns in the dynamic graph. Ferlez et al. [56] use the MDL principle to detect the cut points when communities in the evolving network change abruptly. Sun et al. [148] use MDL principles to find partition and segmentation of graph streams. Araujo et al. [21] use tensor decomposition with MDL to discover temporal communities in dynamic graphs. Their work is community-based while in our problem, we study the patterns in a more general way which is not restricted to communities or clusters.

Time-series segmentation. We can consider time-series segmentation as a sub-problem of the dynamic graph summarization. The research community has made significant efforts in developing algorithms for different problems on time series data. These include algorithms for mining multivariate time series [24], summarizing time series with missing values [106], a generative analysis of time series [161], and many others. Among them, there is also much

work about time series segmentation [111, 137, 156, 38, 37, 112, 65].

2.3 Graph drawing and visualization

Researchers have extensively studied the problem of drawing and visualizing graphs [79]. Most of the approaches heavily rely on user-input through direct manipulation of a graph. For example, ForceSpire [52] and StarSPIRE [30, 19] leverage semantic interaction of user to visualize document networks. Self et. al [141] propose an interactive metric learning method which enables parametric and observation-level interaction to explore data. Work by Ruotsalo et al. uses these direct manipulation interactions to influence information retrieval [135], while similar work from Teevan et al. translates user feedback within a topic spatialization to tune search results incrementally [151]. VIGOR [126] is an interactive visual analytics system for sense-making of graph query results in a cybersecurity setting. FACETS [127] helps non-expert users adaptively explore graphs and focus on neighborhoods that are most subjectively interesting. g-Miner [33] is an interactive system which enables visual mining of groups on multivariate graphs.

There has not been much work on graph drawing from a data mining perspective [109]. Typograph takes a multilevel abstraction approach, using extracted topics, keywords, and document snippets to assist in visualizing large text corpora [51]. Some of the popular works include constraint programming based force-directed layout for directed networks [48], simulated annealing based technique [43], embedding on a Riemann Sphere for path tracing, interactive clustering and data representation [164, 165] and so on. In another approach, Kang et al. propose Net-Ray [83] to plot various metrics on graphs to detect and interpret outliers. Kwon et al. [97] explore an immersive approach for graph visualization, designed specifically for virtual reality environments. Most work in this line of research is task specific: hence these frameworks are not generalizable and cannot be used for other tasks.

2.4 Deep Learning for Graphs

There has been some work in leveraging deep learning for various graph mining tasks, especially in feature learning for graphs. For example, Dai et al. [42] combine deep learning and latent variable models to embed structural data such as sequences and graphs to feature vectors. Other works such as [71, 159] use various deep neural networks to extract feature representations in an unsupervised manner. Researchers have also exploited deep learning for various graph mining tasks like cascade prediction [104], graph classification [121], and graph kernels [171]. As far as we know, we are the first to leverage deep learning for network summarization.

Meta Algorithms on Graphs. Although there has not been much work in this area, it

is seeing increasing recent interest. Dai et. al [90] combine reinforcement learning, latent variable models, and deep learning to propose a meta-algorithm to solve combinatorial problems in graphs. Unlike their method, we propose leveraging deep reinforcement learning to summarize the network in a task-dependent manner such that the original problem can be approximately solved in the summary network. Bay and Sengupta [25] proposed homotopic recurrent neural networks to approximate a metaheuristic for computing shortest path in a graph. Similarly, Nowak et. al [122] propose to train neural networks to solve a quadratic optimization problem over graphs. Bello et. al [26] combine reinforcement and deep learning to solve the Traveling Salesman Problem. They train a recurrent neural network to predict the correct permutation of the nodes. In contrast to all these previous works, we present deep reinforcement learning for network summarization in a task-dependent manner.

Part I

Optimization-based approaches

2.5 Overview

As a first step, we tackle our simplest problem with our more intuitive approach. We start with deterministic approaches. We begin with univariate networks where nodes are unlabeled or binary labeled (Chapters 3 and 4). We propose, a parameter-free, multilevel and flexible framework to summarize sequences of such networks. These summaries have similar influential characteristics to the original graphs. Next we go further and expand our work to multivariate networks (Chapter 5).

In Chapter 3 we proposed a novel general problem of summarizing unlabeled temporal networks 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. Next in Chapter 4 we present SnapNETS and Anomaly-SnapNets, intuitive and effective methods to summarize and segment dynamic binary-labeled network sequences. They efficiently find high-quality segmentations, detects anomalies and important events and gives useful insights into diverse, complex datasets. Finally, we parallelize SnapNETS and Anomaly-SnapNets to accelerate the computations.

In Chapter 5 we proposed ANeTS, a new unsupervised, scalable and parallelizable algorithm which gets high-quality summaries of attributed influence graphs, in *near-linear* time in practice (in contrast to non-trivial competitors). The summary can be used for many applications: For example, it speeds-up the TIM task and detects malicious nodes in a network; and it intuitively highlights structurally and attributed homogeneous regions in the network—helping in sensemaking of complex network datasets.

Chapter 3

Unlabeled network sequences

As explained in Chapter 1, we first tackle the problem of summarizing unlabeled networks. Many of the real-world networks evolve with time. For example, connections in a people-contact network changes during weekdays and weekends, social-network users un/follow new users and so on. As mentioned, getting a smaller representation of a temporal network with similar properties will help in various data mining tasks. In this Chapter, 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 unlabeled 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.

3.1 Introduction

As mentioned, many of the modern networks are time evolving, and they are large. We can consider many of such networks unlabeled, which means that the nodes do not have a label or attribute and edges are unweighted. In this chapter we try to answer; Given a large time-varying unlabeled network, can we get a smaller, nearly "equivalent" one? 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

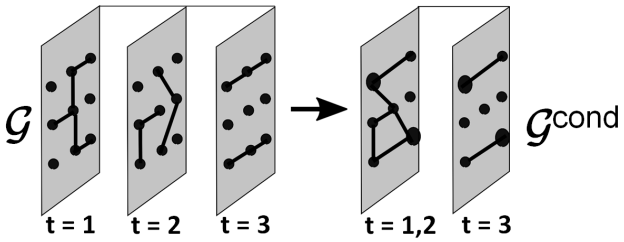


Figure 3.1: Condensing a Temporal Network

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. We study the problem of 'condensing' an unlabeled 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.

3.2 Preliminaries

We give some preliminaries next. Notations used and their descriptions are summarized in Table 3.1.

Temporal Networks: We focus on the analysis of dynamic graphs as a series of individual snapshots. 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 [18] and IC models [88]. 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 [129], 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).

3.3 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 [64]. 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 prop-

Table 3.1: 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)

erty 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.*

3.3.1 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 $\mathcal{G}$.

3.3.2 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 [61, 128] have studied epidemic thresholds for various epidemic models on static graphs. Recently, [129] 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 [129] 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).

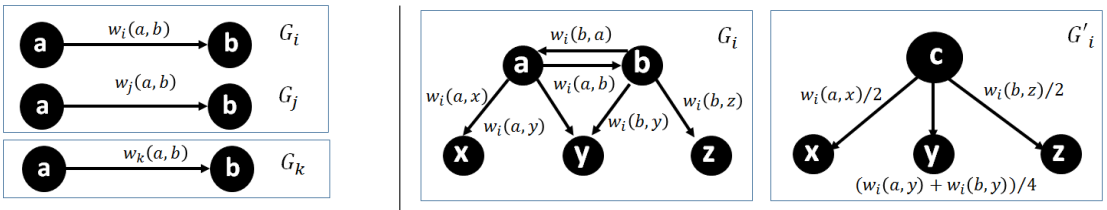
Definition 1 ***System Matrix:*** *For the SI model, the system matrix* $\mathbf{S}_{\mathcal{G}}$ *of a temporal network* $\mathcal{G} = \{G_1, G_2, ..., G_T\}$ *is defined as* $\mathbf{S}_{\mathcal{G}} = \prod_{i=1}^{T}(\mathbf{I} + \mathbf{A}_i)$.

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 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.

3.3.3 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 $G_i \in \mathcal{G}$.



(a) Time merge of a single edge

(b) Node Merge in a single time

Figure 3.2: (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.

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 3.2 (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 1 *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 [80]). 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 3.2 (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:*

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

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.

Proof 2 *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 [147]. As shown by our experiments, keepin just the first-order terms still leads to high quality summaries.

3.3.4 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* $|\lambda_{\boldsymbol{S}} - \lambda_{\boldsymbol{S}}^{\text{cond}}|$*.*

Problem 1 is likely to be challenging as it is related to immunization problems [179]. In fact, a slight variation of the problem can be easily shown to be NP-complete by reduction form the Maximum Clique problem [85] (see Appendix). Additionally, Problem 1 naturally contains the GCP coarsening problem for a static network [130] as a special case: when $\mathcal{G} = \{G\}$, which itself is challenging.

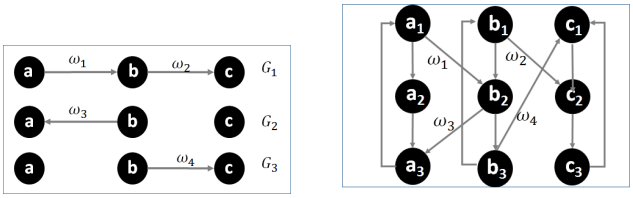
3.4 Our Proposed Method

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.

3.4.1 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.



(a) Temporal Network (b) Flattened Network

Figure 3.3: (a) $\mathcal{G}$, and (b) corresponding $F_{\mathcal{G}}$.

3.4.2 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 3.3). 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 3.3 (a)). Say node a is infected in G_1, which also means node a_1 is infected in $F_{\mathcal{G}}$ (Figure 3.3 (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 3 *First we will show the following:*

$$\sum_{t=0}^{T-1} I_{SI}^{\mathcal{G}}(t) = I_{IC}^{F_{\mathcal{G}}}(T-1) \tag{3.1}$$

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 $(\lambda_{\boldsymbol{F}})^T = \lambda_{\boldsymbol{S}}$. Furthermore, λ is an eigenvalue of $\mathbf{F}_{\mathcal{G}}$, iff λ^T is an eigenvalue of $\mathbf{S}_{\mathcal{G}}$.*

Proof 4 *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

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

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 $|\lambda_{\boldsymbol{F}}| \geq |\lambda|$, 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 $|\lambda^T| > |\lambda_{\boldsymbol{F}}^T|$. 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).

3.4.3 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[39], 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 $\mathbf{F}_\mathcal{G}$, $\lambda_{\boldsymbol{F}}$, and the largest eigenvalue of $\mathbf{X}_\mathcal{G}$, $\lambda_{\boldsymbol{X}}$, are related as $\lambda_{\boldsymbol{F}} \leq \lambda_{\boldsymbol{X}}$.*

Proof 5 *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 $\lambda_{\boldsymbol{X}}$ 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 [175]. 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* $\mathrm{Re}(\lambda) \leq \lambda_{\boldsymbol{X}}$.

Since the union graph $U_\mathcal{G}$ is strongly connected, $F_\mathcal{G}$ is strongly connected. Hence, by Perron Frobenius theorem [62], the largest eigenvalue of $\mathbf{F}_\mathcal{G}$, $\lambda_{\boldsymbol{F}}$, is real and positive. Therefore, $\lambda_{\boldsymbol{F}} \leq \lambda_{\boldsymbol{X}}$.

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 $\mathbf{S}_{\mathcal{G}}$. 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*

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

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

3.4.4 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.*** *$\Delta_{X_{\mathcal{G}}}(a,b) = |\lambda_{\boldsymbol{X}} - \lambda_{\boldsymbol{X}}^{\text{cond}}|$ 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 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 [147]. 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,*

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

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* $\mathbf{k}^{oT}\mathbf{v} = \frac{1}{2}(\lambda_{\boldsymbol{X}}\mathbf{v}(i) + \lambda_{\boldsymbol{X}}\mathbf{v}(j) + \mathbf{v}(i) + \mathbf{v}(j))$.

Proof 6 *For convenience, we write* $\lambda_{\boldsymbol{X}}$ *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*

$$\Delta\lambda = \frac{\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} + \mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v}}{\mathbf{v}^T\mathbf{v} + \mathbf{v}^T\Delta\mathbf{v}} \tag{3.2}$$

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*

$$\Delta\mathbf{X} = \sum_{i\in V_i, V_j} -(\mathbf{i}^i\mathbf{e}_i^T + \mathbf{e}_i\mathbf{i}^{oT}) + \sum_{k\in V_k} -(\mathbf{k}^i\mathbf{e}_k^T + \mathbf{e}_k\mathbf{k}^{oT}) \tag{3.3}$$

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:*

$$\Delta\mathbf{v} = \sum_{i\in V_i, V_j} -(\mathbf{v}_i\mathbf{e}_i) + \delta \tag{3.4}$$

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. 3.2 as follows:*

$$\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} = \sum_{i\in V_i, V_j} -(\mathbf{v}_i\mathbf{v}^T\mathbf{i}^i + \mathbf{v}_i\mathbf{i}^{oT}\mathbf{v}) + \sum_{k\in V_k}(\mathbf{v}_i\mathbf{v}^T\mathbf{k}^i + \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}) \tag{3.5}$$

Further simplifying,

$$\mathbf{v}^T\Delta\mathbf{X}\mathbf{v} = \sum_{i\in V_i, V_j} -(2\lambda\mathbf{v}_i^2) + \sum_{k\in V_k}(\mathbf{v}_i\mathbf{v}^T\mathbf{k}^i + \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}) \tag{3.6}$$

And we have

$$\mathbf{v}^T\Delta\mathbf{v} = \mathbf{v}\sum_{i\in V_i, V_j}(-\mathbf{v}_i\mathbf{e}_i) = -\sum_{i\in V_i, V_j}\mathbf{v}_i^2 \tag{3.7}$$

Similarly,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \mathbf{v}^T[\sum_{i\in V_i,V_j} -(\mathbf{i}^i\mathbf{e}_i^T + \mathbf{e}_i\mathbf{i}^{oT}) + \sum_{k\in V_k} -(\mathbf{k}^i\mathbf{e}_k^T + \mathbf{e}_k\mathbf{k}^{oT})] \\ [\sum_{i\in V_i,V_j} -(\mathbf{v}_i\mathbf{e}_i)] \tag{3.8}$$

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,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \sum_{i\in V_i,V_j} \lambda\mathbf{v}_i\mathbf{v}_j + \sum_{i\in V_i,j\in V_j} (\mathbf{v}_i\mathbf{v}_j + \mathbf{v}_j\mathbf{v}_i)\mathbf{X}(i,j) - \sum_{k\in V_k} \mathbf{v}_i\mathbf{v}^T\mathbf{k}^i - \sum_{k\in V_k,j\in V_j} \mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}_j\mathbf{e}_j. \tag{3.9}$$

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,

$$\mathbf{v}^T\Delta\mathbf{X}\Delta\mathbf{v} = \sum_{i\in V_i,V_j} \lambda\mathbf{v}_i\mathbf{v}_j + \sum_{i\in V_i,j\in V_j} (2\mathbf{v}_i\mathbf{v}_j\mathbf{X}(i,j) - \sum_{k\in V_k} \mathbf{v}_i\mathbf{v}^T\mathbf{k}^i \tag{3.10}$$

Putting together, we have

$$\Delta\lambda = \frac{-\lambda\sum_{i\in V_i,V_j}\mathbf{v}_i^2 + \sum_{i\in V_i,j\in V_j} 2\mathbf{v}_i\mathbf{v}_j)\mathbf{X}(i,j) + \sum_{k\in V_k}\mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}}{\mathbf{v}^T\mathbf{v} - \sum_{i\in V_i,V_j}\mathbf{v}_i^2} \tag{3.11}$$

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,

$$\mathbf{k}^{oT}\mathbf{v} = \sum_{y\in I}\mathbf{v}_y\mathbf{k}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\mathbf{k}_z^{oT} \qquad = \sum_{y\in I}\mathbf{v}_y\frac{1}{2}\mathbf{i}_y^{oT} + \sum_{z\in J}\mathbf{v}_z\frac{1}{2}\mathbf{j}_z^{oT}$$

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,

$$\mathbf{k}^{oT}\mathbf{v} = \frac{1}{2}(\lambda_{\boldsymbol{X}}\mathbf{v}_i + \lambda_{\boldsymbol{X}}\mathbf{v}_j + \mathbf{v}_i + \mathbf{v}_j). \tag{3.12}$$

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,*

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

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 $\mathbf{c}^{oT}\mathbf{v} = \frac{1}{2}\lambda_{\boldsymbol{X}}(\mathbf{v}(a) + \mathbf{v}(b))$.

Proof 7 *Following the steps in Lemma 6, we have*

$$\Delta\lambda = \frac{-\lambda\sum_{i\in V_i,V_j}\mathbf{v}_i^2 + \sum_{i\in V_i,j\in V_j}(2\mathbf{v}_i\mathbf{v}_j)\mathbf{X}(i,j) + \sum_{k\in V_k}\mathbf{v}_i\mathbf{k}^{oT}\mathbf{v}}{\mathbf{v}^T\mathbf{v} - \sum_{i\in V_i,V_j}\mathbf{v}_i^2} \tag{3.13}$$

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,

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

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,

$$\mathbf{k}^{oT}\mathbf{v} = \frac{1}{2}(\lambda_{\boldsymbol{X}}\mathbf{v}_i + \lambda_{\boldsymbol{X}}\mathbf{v}_j + \mathbf{v}_i + \mathbf{v}_j). \tag{3.14}$$

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 8 *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 [155]. 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 $O(\alpha_N \theta T V)$. Similarly, it takes $O(\alpha_T E)$ time for all time-merges.

Therefore, the time-complexity of NetCondense *is $O(TE_u + E \log E + \alpha_N \theta T V + \alpha_T E)$. 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.

3.5 Experiments

3.5.1 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/code/NetCondense.zip`

Datasets. We run NetCondense on a variety of real datasets (Table 3.4) 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 $[0, 1]$.

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 [93] 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 [130] (a summarization method which preserves the diffusive

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

Table 3.2: 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

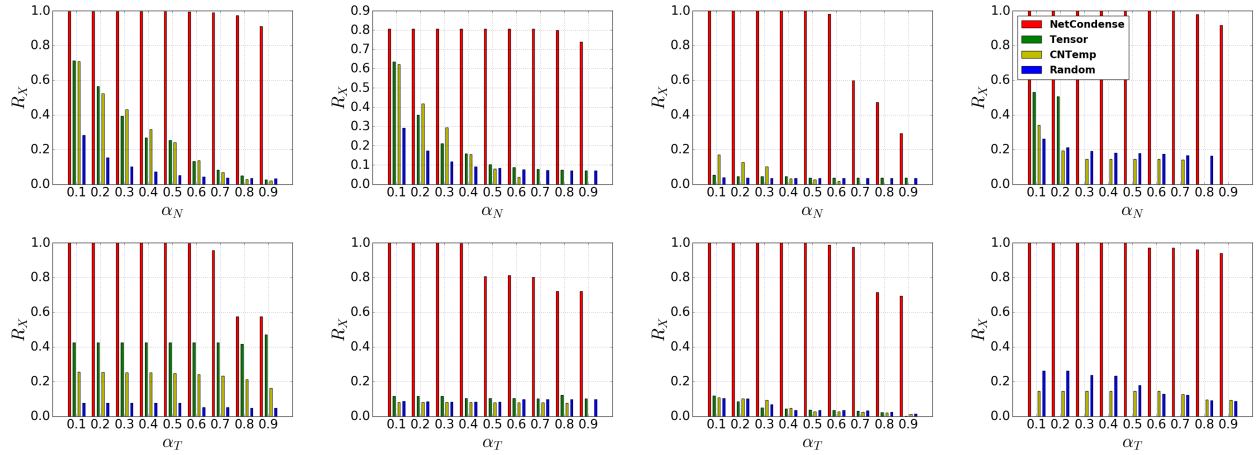
property of a static graph) on $U_{\mathcal{G}}$ and repeat the summary to create $\mathcal{G}^{\text{cond}}$.

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

3.5.2 Perfomance of NetCondense: Effectiveness

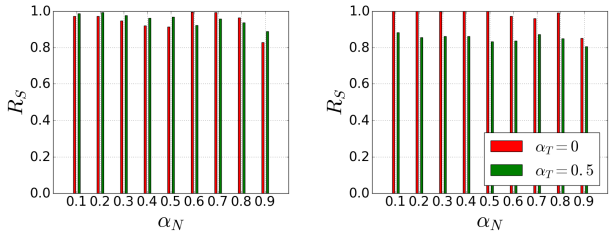
We ran all the algorithms to get $\mathcal{G}^{\text{cond}}$ for different values of α_N and α_T, and measure $R_{\mathbf{X}} = \lambda_{\mathbf{X}}^{\text{cond}}/\lambda_{\mathbf{X}}$ to judge performance for Problem 2. See Figure 3.4. NetCondense is able to preserve $\lambda_{\mathbf{X}}$ excellently (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 3.5 ($\lambda_{\mathbf{S}}^{\text{M}}$ is $\lambda_{\mathbf{S}}^{\text{cond}}$ obtained from method M). NetCondense does almost as well as GreedySys, due to our careful transformations.



(a) ARXIV (b) PROSPERLOAN (c)WIKITALK (d) DBLP

Figure 3.4: $R_{\mathbf{X}} = \lambda_{\mathbf{X}}^{\text{cond}}/\lambda_{\mathbf{X}}$ vs α_N (top row, $\alpha_T = 0.5$) and vs α_T (bottom row, $\alpha_N = 0.5$).



(a) WORKPLACE (b) SCHOOL

Figure 3.5: Plot of $R_{\mathbf{S}} = \lambda_{\mathbf{S}}^{\text{NetCondense}}/\lambda_{\mathbf{S}}^{\text{GREEDYSYS}}$.

3.5.3 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) [4]. 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 ($\mathcal{G}$, 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: $\quad k_i \leftarrow \zeta^{-1}(k'_i)$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{k_i\}$
6: return $\mathcal{S}$

We use two different base TempInfMax methods: ForwardInfluence [4] for the SI model and Greedy-OT [63] 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 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 3.3. 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.

Table 3.3: Performance of CondInf (CI) with ForwardInfluence (FI) and Greedy-OT (GO) as base methods. σ_m and $\mathrm{T_m}$ are the footprint and running time for method m respectively. '-' means the method did not finish.

Dataset	σ_{FI}	σ_{CI}	$\mathrm{T_{FI}}$	$\mathrm{T_{CI}}$
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}	$\mathrm{T_{GO}}$	$\mathrm{T_{CI}}$
SCHOOL	135	128	15m	1.8m
ENRON	119	114	9.8m	24s
CHESS	-	2267	-	8.6m
ARXIV	-	357	-	2.2h
WIKIPEDIA	-	4591	-	3.2h

3.5.4 Application 2: Event Detection

Event detection [132, 14] 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 [14] 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 [14]. These datasets are described below in detail and the summary is in Table 3.4.

AS Oregon-PA and AS Oregon-MIX are Autonomous Systems peering information network collected from the 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 [14].

[4]http://www.topology.eecs.umich.edu/data.html

Table 3.4: 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 3.5: 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

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 [14], 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 3.5.

As shown in Table 3.5, 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

[5]http://chroniclingamerica.loc.gov

suggests a different value of α_N is suitable for different networks in the EDP task.

3.5.5 Application 3: Understanding/Exploring Networks

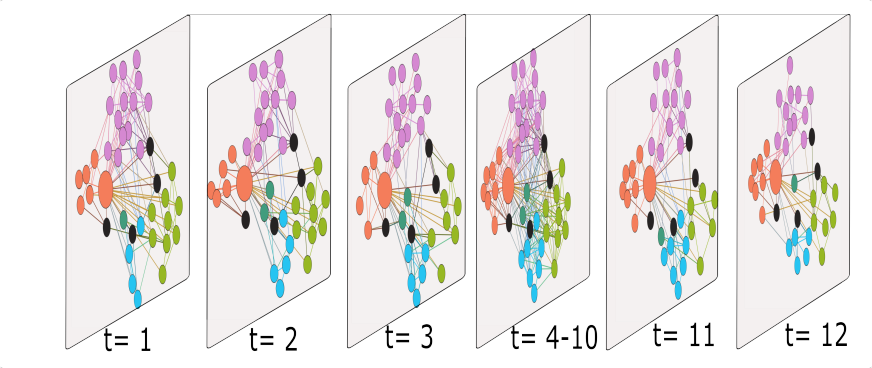


Figure 3.6: Condensed WorkPlace ($\alpha_N = 0.6$, $\alpha_T = 0.5$).

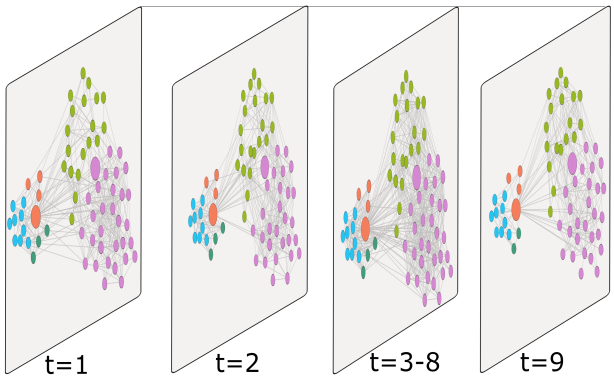


Figure 3.7: Condensed School ($\alpha_N = 0.5$ and $\alpha_T = 0.5$).

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 [64]. In $\mathcal{G}^{\text{cond}}$ (see Figure 3.6), 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 [64].

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 [64]. The stability of "Linkers" mentioned in [64] 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 [59]. 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 [59]. In $\mathcal{G}^{\text{cond}}$ (see Figure 3.7), 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 [59], 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 [59]. 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.

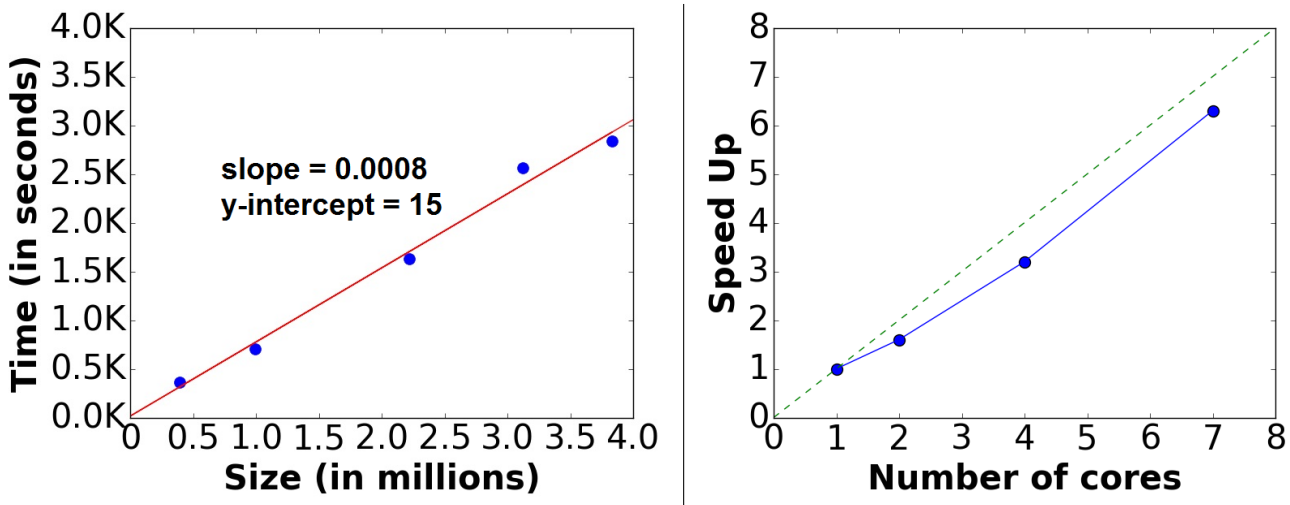
To investigate the nature of nodes which get merged early, we look at the super-nodes at the early stage of NetCondense, we find two super-nodes with one Vice-President in each (Note that most other nodes are still unmerged). Both super-nodes were composed mostly of Managers, Traders, and Employees who 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.

3.5.6 Scalability and Parallelizability

Figure 3.8 (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

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



(a) Scalability (b) Parallelizability

Figure 3.8: (a) Near-linear Scalability w.r.t. size; (b) Near-linear speed up w.r.t number of cores for parallelized implementation.

input size. Figure 3.8 (b) shows the *near-linear* run-time speed-up of parallel-NetCondense vs # cores on WIKIPEDIA.

3.6 Discussion and Conclusions

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.

Chapter 4

Binary-labeled network sequences

In this chapter, we continue to study the problem of summarizing univariate networks by investigating the binary-labeled network sequences. Memes on the Twitter network, diseases over a contact network, movie-cascades over a social network, etc., are all graph sequences with binary labeled nodes. For example, flu can propagate in a people-contact network, and each person can be infected or uninfected in the network. In another example, a meme can spread over the Twitter network and users tweet about it or ignore it.

Given a sequence of snapshots of flu propagating over a population network, can we find a segmentation when the patterns of the disease spread change, possibly due to interventions? In this chapter, we study the problem of segmenting graph sequences with labeled nodes with the help of network summarization. Most related work on this subject is on plain graphs and hence ignores the label dynamics. Others require to fix parameters or feature engineering. We propose SnapNETS, to *automatically* summarize and find segmentations of such binary labeled graph sequences, with different characteristics of nodes of each label in adjacent segments. It satisfies all the desired properties (being parameter free, comprehensive and scalable) by leveraging a principled, multilevel, flexible framework which maps the problem to a path optimization problem over a weighted DAG. Also, we develop the parallel framework of SnapNETS which speeds up its running time. Finally, we propose an extension of SnapNETS to handle the dynamic graph structures and use it to detect anomalies (and events) in network sequences.

Extensive experiments on several diverse real datasets show that it finds cut points matching ground-truth or meaningful external signals and detects anomalies outperforming non-trivial baselines. We also show that the segmentations are easily interpretable and that SnapNETS scales near-linearly with the size of the input. Finally, we show how to use SnapNETS to detect anomaly in a sequence of dynamic networks.

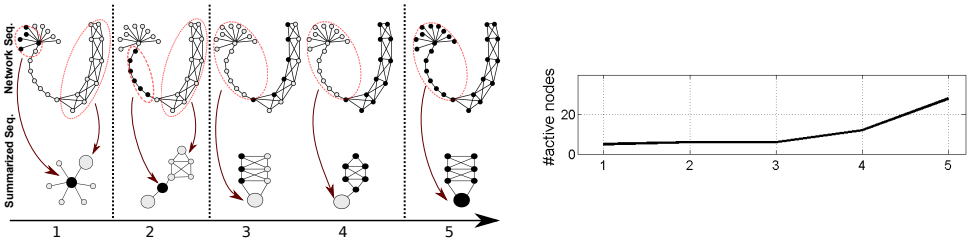


Figure 4.1: Toy Example: SnapNETS automatically identifies four significant steps of the network sequence. The extracted time series (e.g., #active nodes) cannot capture a proper segmentation. Gray nodes are inactive (i.e., label 0), and black nodes are active (i.e., label 1).

4.1 Introduction

Suppose we have a sequence of Ebola infections and the associated contact network of who-can-infect-whom. Can we quickly tell a public health expert when the infection patterns change possibly due to a virus mutation? By itself, it is crucial for public health to understand virus propagation and to design a good immunization strategy. One possible approach is to segment the sequence based on some manually selected features, such as the rate of infections. However, by directly analyzing the underlying social network, and using both the infected and uninfected nodes, we can improve the segmentation as well as its interpretability (e.g. 'disease spread in a tree-like fashion among elderly till Monday, and changed to clique-like fashion among the young' and so on).

Segmenting a graph sequence is an important problem which can help us in better understanding the evolution of the dataset. This problem has numerous applications from epidemiology/public health to social media (rumors/memes on social networks like Twitter), anomaly detection and cyber security (malware on computer networks). In this thesis, we study the problem of segmenting a graph sequence with varying node-label distributions. First, we assume binary labels and fixed structure for graphs. Next, we propose an extension to our algorithm to handle dynamic graphs with varying nodes and edges then leverage it to detect anomalies and important events. For diseases/memes, the labels can be 'infected'/'active' (1) & 'healthy'/'inactive' (0), and the network can be the underlying contact-network. Our problem is:

Problem 1: Segmentation

***Given*:** a sequence $\mathcal{G}$ of networks $G_1, G_2, \ldots, G_T$ with labeled nodes,
***Find*:** best segmentation c^*, which captures different patterns of node labels in $\mathcal{G}$ such that adjacent segments have different characteristics of nodes with the same label.

Toy Example. Suppose $\mathcal{G} = \{G_1, G_2, G_3, G_4, G_5\}$ with 0, 1 labeled nodes (Fig. 4.1 top row). There are four main steps in $\mathcal{G}$: First a central node in a star and some of its spokes have the

Table 4.1: Comparison of SnapNETS with alternative approaches. A dashed cross means most approaches does not satisfy the property; similarly for the dashed check.

Properties	**SnapNETS**	**Feature eng. and time series** [106, 107, 78]	**Plain-graph-based** [144, 95, 56, 131]
Parameter free	✓	✗	✗
Comprehensive	✓	✓	✘
Scalable	✓	✘	✓

label 1; next, low degree nodes in a chain-shaped manner get label 1 (structural change). In the third segment, the label moves to another community of the graph (community change). Finally, the whole graph gets label 1 which indicates an activation rate increase in the network (rate changes). Hence $c^* = \{1, 2, 3, 5\}$. Note that even though the 'active' sub-graphs in time-step 2 and 3 are both chains, their roles in the entire graph are different and so they should belong in different segments. In time-step 2 the active chain is a bridge between two parts while the chain in time-step 3 is part of a near-clique community (role change). Therefore, to get c^* we must consider the entire graph at each time stamp.

Based on the above example, any ideal segmentation algorithm should have the following desired properties:

P1. Parameter free: Find the best number of segments and segmentation without use of parameters i.e. change threshold, time window, and number of desired segments.
P2. Comprehensive: Use the entire snapshot for segmentation, instead of merely active subgraphs.
P3. Scalable: The method must be scalable (i.e. scales sub-quadratically with the input size which can be millions of edges and nodes in the sequence).

This problem has been barely (if at all) studied in literature. Unlike most methods that we can adopt to solve this problem, SnapNETS has all the above mentioned properties.

(Table 4.1 shows a brief comparison; more discussion in Section 4.3.1). SnapNETS is a novel multi-level approach which summarizes the given networks/labels in a very *general way* at *multiple different time-granularities*, and then converts the problem into an appropriate *optimization problem* on a data structure. We give a novel efficient algorithm for the optimization problem as well. A strong advantage of this framework is that it allows us to *automatically* find the right number of segments avoiding over or under segmentation in a very systematic and intuitive fashion. Further it gives naturally interpretable segments, enhancing its applicability. Also, we easily extend SnapNETS to segment dynamic graph sequences which can be used to detect anomalies and important events. Finally, we demonstrate SnapNETS's usefulness via multiple experiments for segmentation and anomaly detection on diverse real-datasets.

4.2 Preliminaries

In this section, we go over useful preliminaries, in particular diffusion models and plain graph summarization algorithms. Table 4.2 summarizes the notations that we use throughout this Chapter to describe the segmentation problem and SnapNETS algorithm.

Table 4.2: Some of the notations used

Symbol	Description
$G_.$, $G^c_.$	The original and the summarized graph.
$\mathcal{G}$	A sequence of T *Act-snapshots* (i.e. *AS-Sequence*)
T	The last time stamp in *AS-Sequence* $\mathcal{G}$
c	A valid segmentation of time interval $[1, T]$
c^*	The best segmentation of time interval $[1, T]$
$\mathcal{C}$	The set of all possible segmentations.
$s_{i,j}$	A segment $[i, j)$.
$\mathcal{S}$	The set of all possible segments.
$\mathbf{F}_.$	The feature vector of each *C-graph*
$\hat{\mathbf{F}}_.$	The feature vector of each segment.
$d(\cdot, \cdot)$	Distance between two segments.
$\mathsf{G_s}$	Segmentation graph
s , t	Source and target nodes of the segmentation graph
λ_G	The leading eigenvalue of graph G
l_x·	The label of a node x
ρ	The compression ratio to summarize graph G
nop	Number of processors

4.2.1 Diffusion models

Node labels (i.e. active:1 and inactive:0) usually come from diffusion process in networks. Different models are distinguished based on how they define the activation cycle on a node in a network G. We categorize diffusion models into two main groups: (1) Progressive: such as **I**ndependent **C**ascade (IC) and **S**usceptible-**I**nfected (SI) models where nodes can not get inactive. (2) Non-progressive: such as the 'flu-like' **S**usceptible-**I**nfected-**S**usceptible (SIS), and **S**usceptible-**I**nfected-**R**ecovered (SIR) models where active node can get inactive again. [18, 88].

In diffusion models, initially, a node is susceptible (inactive). Overtime, that node can get infected (active) according to some probability. Finally, depending on the diffusion model, the node may remain infected or become susceptible (inactive) again or be removed

(immunized). For example, in the IC model a vertex $v \in V$ is called active if it has been influenced and inactive otherwise. Each active node has a single chance to activate each currently inactive neighbor with an independent probability over the edge. This diffusion process continues until no more activations are possible.

4.2.2 Summarizing plain graphs

We leverage graph summarization in our proposed algorithm. Our summarization is inspired by CoarseNet [130] method, an algorithm for coarsening a graph while preserving the propagation characteristics of the graph as much as possible. The algorithm takes a weighted graph $G(V, E, w)$ and a target fraction $0 < \alpha < 1$ as input, and coarsens a fraction $1 - \alpha$ of total nodes such that the coarsened graph $G' = (V', E', w')$ approximates graph G with respect to its diffusive properties.

CoarseNet uses the leading eigenvalue of the adjacency matrix as an indicator of the graph's diffusion properties, and greedily merges adjacent nodes such that the drop in the first eigenvalue is minimized. It evaluates the quality of merging each edge based on a score they define as follows,

$$score(a, b) = |\lambda_{G-(a,b)} - \lambda_G| = \Delta\lambda_{(a,b)} \tag{4.1}$$

Where, λ in the leading eigenvalue of the adjacency matrix of a graph. $score(a, b)$ measure the change of leading eigenvalue after merging a and b. It is shown that the estimation of $score(a, b)$ is as follows,

$$score(a, b) \simeq \frac{-\lambda(u_a v_a + u_b v_b) + v_a \overrightarrow{u}^T \overrightarrow{c^o} + \beta_2 u_a v_b + \beta_1 u_b v_a}{\overrightarrow{v}^T \overrightarrow{u} - (u_a v_a + u_b v_b)} \tag{4.2}$$

Where $\overrightarrow{u}$ and $\overrightarrow{v}$ are the right and left eigenvectors of graph G. u_a denotes the component of $\overrightarrow{u}$ corresponding to vertex a and $\overrightarrow{c^o}$ is the vector of outgoing edges from the supernode c (i.e. the node is generated after merging a and b).

4.3 Overview and main ideas

For sake of simplicity, we focus on the case when nodes have binary labels[1] i.e. active: 1 or inactive: 0. Also, for ease of description, first we assume that the network remains constant through time and we treat the problem as one with a series of graph snapshots. Next, we show a natural extension of SnapNETS to work with dynamic graph sequences (See sections 4.5 and 4.6.8).

[1]Extending to multiple labels is interesting future work.

Finally, we allow that nodes can even *switch* between labels freely (c.f. Figure 4.1). This means that we can handle both progressive/non-progressive scenarios: e.g. SI, IC, and SIS models. Next we give some useful definitions:

Definition 8 (*Act-snapshot*) *$G(V, E, \mathbf{L})$ is an Act-snapshot. $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ are sets of nodes and edges of G. $\mathbf{L} = [l_1, l_2, \ldots, l_n]$ shows the labels of nodes. l_x is 1 if v_x is active and 0 otherwise.*

Definition 9 (*AS-Sequence*) *$\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ is sequence of T Act-snapshots with G_i at time-step i.*

Definition 10 (Segment) *A segment $s_{i,j}$ is a time interval between Act-snapshots G_i and G_j i.e. $s_{i,j} = \{[i, j) \mid i < j\}$. Set of all possible segments is $\mathcal{S} = \{s_{1,2}, s_{1,3}, \ldots\}$.*

Definition 11 (Segmentation) *A segmentation c of size q is a partition of time interval $[1, T]$ with q time stamps i.e. $c = \{a_1, a_2, \ldots, a_q\}$ where $a_i \in \{1, 2, \ldots, T\}$. The set of all possible segmentations is $\mathcal{C}$.*

Definition 12 (*Act-set$_i$*) *Act-set$_i$ contains the active nodes in Act-snapshot G_i i.e. Act-set$_i$ $= \{v_x | l_x = 1\}$.*

Hence our problem is to automatically segment a given *AS-Sequence*. We next explain the shortcomings of alternative approaches, and then give the big picture of our framework.

4.3.1 Shortcomings of alternative approaches

Two natural ways to adapt existing algorithms for this task are: (a) extract complex features from *Act-snapshots* and use time-series segmentation; and (b) extract *Act-set*s and use plain-graph-based methods.

Feature Eng. and Time Series. Converting graphs sequences to time series has several drawbacks. First of all, it needs laborious feature-engineering: designing the right features to capture the pattern of graphs is a complicated task, usually requires a large set of expensive features and the best choice of features may differ for different sequences [78]. For example, we may need to extract the number of stars in a sequence of social networks to identify the structure of graphs. However, we may need to count the number of ladders in infrastructure networks. Second, typical time series segmentation algorithms, which use "local" change detection, do not satisfy our desired properties. They usually need a threshold [107] (which usually depends on knowing the number of desired segments) to detect a change; or they fix *one* aggregation time period for the tracking [106]. All of these can be problematic, as it is fundamentally hard to set these parameters.

Plain-graph-based analysis. Instead of manually designing complicated features, an alternative is to use plain-graph-based methods on induced subgraphs from *Act-snapshots*. However, these approaches do not satisfy **P2**, as they typically track only the *Act-set* in each *Act-snapshot* [144, 95, 131]. As we show in Figure 4.1, using only the active sub-graphs leads to less meaningful segmentation: the active sub-graphs at time-step 2 and 3 are both a chain of a same size, nevertheless, as discussed before the roles of these chains are different in the two snapshots. If we just track active sub-graphs, we cannot detect this difference.

4.3.2 Overview of our method SnapNETS

In order to overcome the disadvantages we discussed before, we propose a "global" framework which looks at the entire *AS-Sequence* $\mathcal{G}$ and computes the correct segmentation. Due to **P1**, we want to examine all possible segmentations $\mathcal{C}$ over all granularities. How to do this efficiently? Our first main idea is to use a graph data structure (called the *segmentation graph* $\mathsf{G_s}$) to efficiently represent the exponential number of all segmentations in space polynomial with respect to the sequence length. The nodes mainly represent the segments in $\mathcal{S}$, while the edge weights indicate the distance ('difference') between adjacent segments. Hence any segmentation is mapped to a path between start and end time in $\mathsf{G_s}$.

How to compute the distance between any adjacent segments $w(s_{i,j}, s_{j,k})$ (each segment will contain sets of *Act-snapshots* G_i)? Note that we want to use the entire graph due to **P2**. This is related to the problem of graph isomorphism. It is natural to compare the segments by examining extracted features such as number of nodes, cliques, diameter and so forth. This will require complex feature construction to correctly and sufficiently represent each graph snapshot. Nevertheless, despite the size of the graphs, patterns in the real-world are usually much less complex. Hence, our second main idea is to develop a *smaller* summary G_i^c which maintains important information in an efficient manner. As a result, we only need a few standard features to represent these summaries.

Finally, how to find the best path in $\mathsf{G_s}$? We need to define this best path and design an efficient algorithm to find it in $\mathsf{G_s}$. Our third main idea is to use the average longest path optimization problem on $\mathsf{G_s}$, as it intuitively regularizes the length of the path (number of segments) with the weight (difference between segments). We also develop the efficient novel algorithm Layered-ALP to find this path.

In short, we pursue 3 main goals:

Goal 1. *Summarizing* G_i: summarize each G_i to capture the structural and label dependent properties (Section 4.4.1).

Goal 2. *Constructing* $\mathsf{G_s}$: extract the features of summarized graphs and compute the distance between adjacent segments; and then construct $\mathsf{G_s}$ (Section 4.4.2).

Goal 3. *Defining and extracting the best segmentation*: define the best segmentation (path) and develop an efficient algorithm to find it (Section 4.4.3).

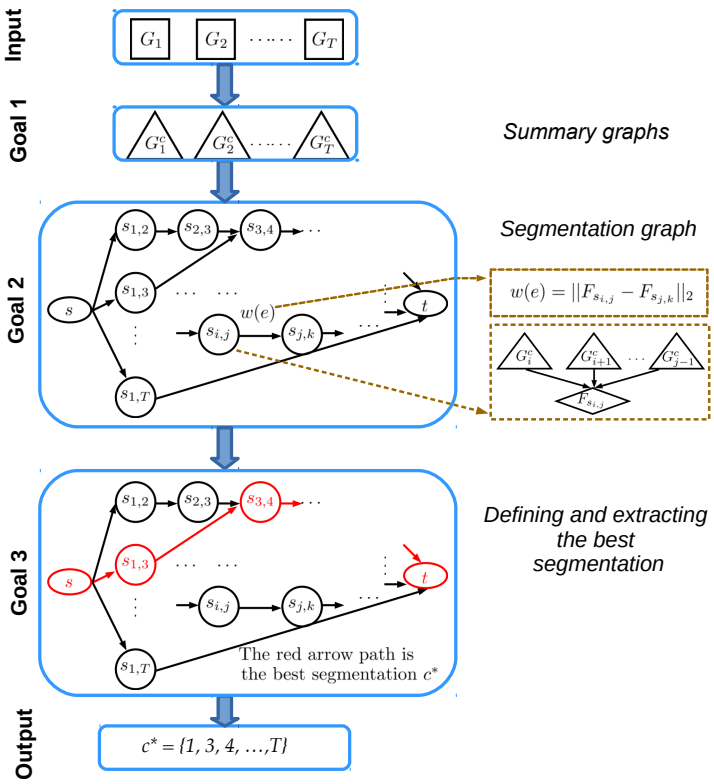


Figure 4.2: SnapNETS overview (Best viewed in color).

Figure 4.2 shows an overview of SnapNETS (**Snap**ping **NET**work **S**equences). Our careful design and parallel implementation help to satisfy **P3**: SnapNETS is sub-quadratic in running time with respect to the size of the sequence and our experiments show its effectiveness and scalability.

4.4 SnapNETs: Details

4.4.1 Goal 1: Summarizing Act-snapshots

We first propose finding a *C-graph* (i.e. G_i^c), which summarizes the structural properties and the nodes labels of each *Act-snapshot* G_i. Popular methods for graph summarization include graph sparsification [110] which try to carefully remove edges to reduce the graph's density while maintaining some properties. Nevertheless, these methods are typically designed for plain graphs and it is not straightforward to modify them for *Act-snapshots*. So we adopt a different, *merging-based* approach which reduces the *number of nodes* instead while maintaining an intuitive and important property.

Role of Eigenvalues: In many real datasets node labels come from a diffusion/propagation process. Recent work [128] shows that important diffusion characteristics of a graph (including the so-called 'epidemic threshold') are captured by the leading eigenvalue of the adjacency matrix, for *almost all* cascade models. This naturally suggests that if the leading eigenvalue of the adjacency matrix of the summarized graph G_i^c and *Act-snapshot* are close, G_i and G_i^c will have similar properties.

Summarizing Act-snapshots via Coarsening: Motivated by the above, we want to successively merge connected nodes into 'super-nodes' (i.e. 'coarsen') while maintaining the leading eigenvalue of the adjacency matrix. Also, we want to keep the same set of labels (0/1) in the *C-graph* to keep it consistent with the *Act-snapshot* and help interpretability and make it easier to compare *C-graphs*. Thus, we define the summarization problem as follows,

PROBLEM 2: *Act-snapshot* SUMMARIZATION

***Given*:** an *Act-snapshot* $G_i(V_i, E_i, L_i)$, and remained fraction of nodes ρ.
***Find*:** a coarsened graph $G_i^c(V_i^c, E_i^c, L_i^c)$ such that

$$\underset{|V_i^c|=\rho|V_i|}{\textit{minimizes}} \; |\lambda_{G_i} - \lambda_{G_i^c}| \;\; \textit{subject to} \;\; \sum\nolimits_{(x,y)\in E_i \textit{is merged}} |l_x - l_y| = 0$$

Here λ_G is the leading eigenvalue of graph G and l_x is the label of node x. This formulation allows us to be model-free and not assume any specific model (such as IC/SIS, etc.). The constraint in PROBLEM 2 maintains the 'frontier' between active and inactive nodes to help consistency and interpretability. PROBLEM 2 is similar to the graph coarsening problem (GCP [130]) whose only goal is to maintain λ_G, but without any constraint—they give an efficient algorithm for this purpose which merges edges based on a quality score. Hence, we modify that algorithm by prohibiting merging of node-pairs with different labels. Algorithm 4 shows our algorithm to solve the PROBLEM 2. We compute the scores of edges based on Equation 4.2 and sort them in lines 1-3. Then, gradually merge edges with end nodes with same labels from the ordered list (lines 5-9). This algorithm works very well in practice and gives near-linear running time. Note that a better algorithm for PROBLEM 2 will only improve our results. A desired value for ρ is the lowest value which maintain λ_G. We use the same amount of coarsening ($\rho = 0.1$) as in [130]. If the ρ value is too large, the summary will be basically similar to the original graph, and we need more complicated features to identify its properties (similar to the original graphs). Also, too small ρ value (i.e., lower than 0.1) is not desirable since all the nodes collapse into few super-nodes and the λ will deteriorate fast.

Figure 4.1 shows our summaries via PROBLEM 2 for the TOY EXAMPLE: the *C-graphs* clearly show the important non-trivial pattern changes in both the structural and label properties of the original graphs succinctly.

Algorithm 4 *Act-snapshot* summarization

Require: *Act-snapshot* $(V, E, \mathbf{L})$, ρ
1: **for** $(x, y) \in E$ **do**
2: Compute $score(x, y)$
3: $\pi \leftarrow$ Sort the edges base of their scores in increasing order
4: $r = 0$, $G^c_\cdot = G$
5: **while** $r \leq (1 - \rho) \cdot |V|$ **do**
6: $(x, y) = \pi(r)$
7: **if** $l_x == l_y$ **then**
8: $G^c_\cdot \leftarrow merge_{G^c}(x, y)$
9: $r \leftarrow r + 1$
10: **return** summary graph *C-graph*

4.4.2 Goal 2: Constructing the segmentation graph

After summarizing the *Act-snapshots*, each segment in the *AS-Sequence* contains a set of *C-graphs*. How to find the distance between two such segments? In general, computing distances between *unlabeled* graphs is itself a challenging problem [95]. Fortunately, in our case, we can just extract *simple* features from the *C-graphs* due to their small size and complexity; and use them to compute the distance. Subsequently, we build the segmentation graph $\mathsf{G_s}$ to store the segments and distances information. Recall that $\mathsf{G_s}$ can efficiently represent all the exp. number of possible segmentations in poly. space.

Feature extraction of C-graphs Extracting features from G^c_i is much more efficient primarily because of their smaller size. Further our summarization maintains the relevant important properties effectively. So we do not need complex features such as "number of particular substructures" (e.g. stars, maximal cliques, ladders, etc.) used in related work.

We extracted multiple *standard* features [105] and eliminate correlated ones to get eight features for each G^c_i (See Table 4.3 for a description). Feature vector $\mathbf{F}_i$ contains: *Structural* features (f_1-f_4); and *Label dependent* features (f_5-f_8) (label-dependent properties). Finally, we normalize them by range normalization for a meaningful comparison between the features [105]. Thanks to our careful design, we can use very simple features for our task.

Segmentation graph We now describe how to construct $\mathsf{G_s}$ to compactly store and represent segmentations. $\mathsf{G_s}(V_s, E_s)$ is a unique weighted DAG where:

Nodes (V_s): For each segment $s_{i,j} \in \mathcal{S}$, there is one node in the graph $\mathsf{G_s}$. We also add two extra nodes to the graph: a source node s and a target node t. Therefore, $V_s = \mathcal{S} \cup \{s, t\}$.

Edges (E_s): There is a directed edge from node $s_{i,j}$ to any node $s_{j,k}$. Also, the source node s links to all nodes with starting time stamp 1 and all nodes with ending time stamp T links to the target node t. Hence, $E_s = \{e(s_{i,j}, s_{j,k})\} \cup \{e(s, s_{i,j})|i = 1\} \cup \{e(s_{i,j}, t)|j = T\}$.

Table 4.3: Features extracted to represent each summarized *Act-snapshot* (i.e. *C-graph*).

Type	ID	Name	Features description
Structural	f_1	Largest eigenvalue of the adjacency matrix	It indicates the structural and label dependent properties of *C-graphs*. We expect changes in eigenvalues of *C-graphs* in *AS-Sequence* due to the restriction of maintaining the frontier (Sec 4.4.1). Hence this feature will encode how the labels are distributed with respect to the *Act-snapshot*.
	f_2	Number of edges	It measures the density of each *C-graph* which indicates how nodes got merged due to the label distribution in the *Act-snapshot*
	f_3	Entropy of the edge weight distribution	It encodes which edges in the original graph got merged. During *CoarseAct*, the edge weights change to maintain the leading eigenvalue (see Sec 4.4.1). Therefore, the distribution of edge weights contains information about which edges are merged and how much merging happens.
	f_4	Average clustering coefficient	It measures the relative frequency of triangles in a *C-graph*. It shows when a new community get activated.
Label based	f_5	Number of active nodes	It counts the remaining active nodes after summarizing an *Act-snapshot*.
	f_6	Average PageRank of active nodes	It measures the structural importance of active nodes in *C-graphs*.
	f_7	Average degree of active nodes	It measures centrality among active nodes in *C-graphs*.
	f_8	Average degree of active neighbors of active nodes	It indicates the role and importance of active nodes.

Edge Weights ($w(e)$): The weight of all edges from s or to t are zero. The weight of an edge from $s_{i,j}$ to $s_{j,k}$ is equal to the distance between sets of *C-graphs* in their corresponding segments i.e. $w(e(s_{i,j}, s_{j,k})) = d(s_{i,j}, s_{j,k})$.

How to get this distance? Using the $\mathbf{F}_i$ for each G_i^c, we compute the average feature vector over all the *C-graphs* in a segment as the segment's representative i.e. $\widehat{\mathbf{F}}_{s_{i,j}} = \frac{\sum_{a=i}^{j} \mathbf{F}_a}{(j-i+1)}$, where $\mathbf{F}_a$ is the feature vector of G_a^c in $s_{i,j}$. This representation has a natural interpretation as it captures the average 'pattern' of *C-graphs* of the segment. Then the distance $d(s_{i,j}, s_{j,k})$ between '$s_{i,j}$' and '$s_{j,k}$' can be defined as $d(s_{i,j}, s_{j,k}) = ||\widehat{\mathbf{F}}_{s_{i,j}} - \widehat{\mathbf{F}}_{s_{j,k}}||_2$. Figure 4.3 shows the $\mathsf{G_s}$ for our Toy Example. Edge weights are not shown for clarity. Note that $\mathsf{G_s}$ is a DAG since its edges are directed and there is no cycle in it (as we cannot go back in time). Further, note that we can compute the distance for every edge in $\mathsf{G_s}$ independently and in parallel. Also, we need to compute the summary just once for each G_i, *not* for each segment in $\mathsf{G_s}$.

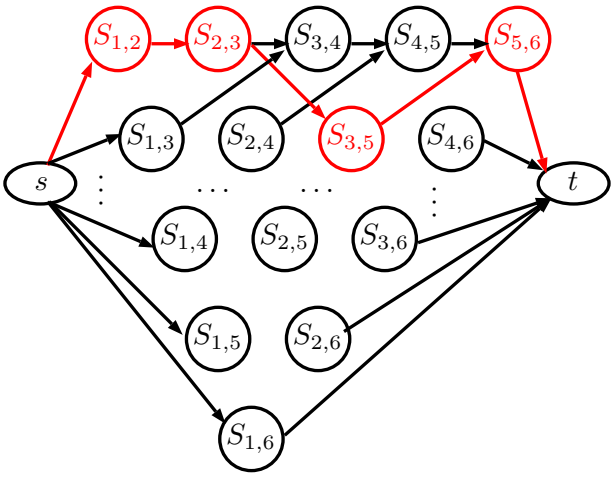


Figure 4.3: The segmentation graph $\mathsf{G_s}$ of the Toy Example. The red path from s to t represent the best segmentation c^*.

4.4.3 Goal 3: Finding the best segmentation

Let $\mathcal{P}$ be the set of all paths in $\mathsf{G_s}$ from s to t. Then,

Lemma 9 *Each path $p \in \mathcal{P}$ corresponds to a valid segmentation $c \in \mathcal{C}$ and for each $c \in \mathcal{C}$ there is a path $p \in \mathcal{P}$.*

Proof 1 *It is obvious based on construction of $\mathsf{G_s}$.*

Hence to get the best segmentation, we only need to *define* and *find* the best path in $\mathsf{G_s}$; which we discuss next.

Average longest path Note that defining the best path is a different and independent question to that of defining edge weights. We define the best segmentation as follows:

Problem 3: Finding the best segmentation

***Given*:** a segmentation graph $\mathsf{G_s}$
***Find*:** the average longest path from s to t in $\mathsf{G_s}$ i.e.

$$c^* = \arg\max_{c \in \mathcal{P}} \frac{\sum_{s_{i,j}, s_{j,k} \in \mathcal{S}} w(e(s_{i,j}, s_{j,k}))}{|c|} \tag{4.3}$$

Thus, Problem 3 is the *Average*-Longest Path (ALP) problem on $\mathsf{G_s}$ (restricted to the path set $\mathcal{P}$). ALP defines the path (segmentation) quality as the average value of edge weights in

the path (distance between its segments). Note that ALP is parameter-free and importantly, it also naturally *balances* the 'length' (weight) of the path (difference between segments) with number of nodes in the path (number of segments).

An alternative 'parameter-free' optimization would have been the Longest Path (LP) problem: which will try to find the *longest* (heaviest) path in $\mathcal{P}$ (Equation 4.3, without the denominator). However, the LP formulation will suffer from *over-segmentation*—it is biased by the number of segments in the path, in the sense that it tends to prefer longer paths with more nodes, irrespective of the edge weights [136]. In practice our observations confirm that LP contains unnecessary edges with low weight (see Sec 4.6). Our ALP objective is intuitive and overcomes the disadvantage of LP. Figure 4.3 shows the ALP for Toy Example in red.

Layered-ALP

ALP can be solved in polynomial time on DAGs (recall $\mathsf{G_s}$ is a DAG)[2]. Current state-of-the-art algorithm [136] can solve Problem 3 in $O(V_s^2 \cdot E_s)$. This is too slow for our purposes; hence, we propose a new and more efficient $O(E_s)$ algorithm for ALP on DAGs called Layered-ALP.

The main observation we use is that the ALP from s to t is the *longest* ('heaviest') path among all paths with the same number of nodes (the 'length') as the ALP. This fact leads us to calculate all the heaviest paths with different lengths in $\mathcal{P}$ and find the one which gives the maximum average edge weight. In Layered-ALP (Algorithm 6) we build a queue of layers of $\mathsf{G_s}$ (lines 5 - 7). Each layer i contains nodes which can reach t by i steps. When we iterate through layers (lines 10 - 15), we maintain the weight ($P_i(v,t)$) of the heaviest path from v to t in i steps, as well as the next node of v in this path (κ_v^i). If s is in layer i, it means there is a path from s to t. We store the $P_{CL}(s,t)$ as the *longest* ('heaviest') path with i nodes from s to t. After iterating over all layers, we have a set of *longest* paths with various number of nodes. The ALP is a path among this set which maximizes its weight divided by its number of nodes. Algorithm 6 shows the pseudo-code of Layered-ALP.

Lemma 10 *The* Layered-ALP *algorithm correctly finds the average longest path* $\mathsf{G_s}$.

Proof 2 *Assume $P_i(a,b)$ is a path from node a to b with i steps.* First *we show that* Layered-ALP *finds all longest paths with different lengths from $\mathcal{P}$. The longest path problem in $\mathsf{G_s}$ has optimal sub-structures because it is a DAG: If we decompose the path $P_h(s,t)$ into $P_{h-h'}(s,v)$ and $P_{h'}(v,t)$ where v is in layer h' in the Queue, and if $P_h(s,t)$ is the longest path form s to t with h steps, then $P_{h'}(v,t)$ will be the longest path from v to t with h' steps.* Second, *ALP is a path in the set of longest paths of different lengths. Assume not, which means there is a path with the same number of nodes as one of the LPs and higher weight which is a contradiction.* □

[2]ALP is NP-hard on general graphs

Lemma 11 *Time complexity of* LAYERED-ALP *is $O(|E_s|)$, where $|E_s|$ is number of edges in $\mathsf{G_s}$.*

Proof 3 *The time complexity of* LAYERED-ALP *is equal to the number of edges visited in the algorithm which can be calculated as follows:*

$$\begin{aligned}
&\text{Layer 1: } (T-1) && = (T-1)\\
&\text{Layer 2: } (T-2)+(T-3)+\ldots+1 && = \frac{(T-1)(T-2)}{2}\\
&\text{Layer 3: } (T-3)+(T-4)+\ldots+1 && = \frac{(T-2)(T-3)}{2}\\
&\text{Layer 4: } (T-4)+\ldots+1 && = \frac{(T-3)(T-4)}{4}\\
&\vdots && \vdots\\
&\text{Layer } T\text{-1: } 1 && = 1
\end{aligned}$$

Therefore, the number of visited edges is,

$$(T-1)+\underbrace{\frac{(T-1)(T-2)}{2}+\frac{(T-2)(T-3)}{2}}_{(T-2)^2}+\underbrace{\frac{(T-3)(T-4)}{2}+\frac{(T-4)(T-5)}{2}}_{(T-4)^2}+\ldots+1$$

$$=(T-1)+\sum_{i=1}^{\frac{T}{2}}(T-2i)^2=O(T^3)=O(|E|). \quad \square$$

4.4.4 Parallelization

Summarizing different *Act-snapshots* is an entirely independent process for each *Act-snapshot* (Algorithm 5, line 1). Also, we can extract the segments features and compute the edge weights of the segmentation graph $\mathsf{G_s}$ independently (Algorithm 5 lines 2-3). These observations motivates us to propose a parallel framework for SnapNETS to further scale it up. We divide this frameworks into two steps: (1) Summarizing *Act-snapshots*, and extracting features of each *C-graph* (2) Constructing the segmentation graph— Constructing nodes, and computing edge weights. In the following we explain each step in detail.

Get *C-graphs* and their features

If the *Act-snapshots* in the sequence are large, even the near-linear time complexity of summarization (Algorithm 4) will be expensive and time consuming. Therefore, we want to

Algorithm 5 SnapNETS

Require: *AS-Sequence* $\mathcal{G}$

1: For each $G_i \in \mathcal{G}$ coarsen and get G_i^c once (Section 4.4.1)
2: Compute feature $\widehat{\mathbf{F}}$ vector of segments in $\mathcal{S}$ (Section 4.4.2)
3: Generate the segmentation graph $\mathsf{G_s}$ (Section 4.4.2)
4: c^* = LAYERED-ALP ($\mathsf{G_s}$, $\mathcal{G}$, s, t) (Section 4.4.3)
5: **return** The optimal segmentation c^*

parallelize this step in Section 4.4.1 to further scale up SnapNETS. The summarization process and extracting features of summary graphs are independent for each *Act-snapshot*. Therefore, we consider the following parallelization scheme to get *C-graphs* and their feature vectors: Assume there are *nop* processors available. (1) Assign $\frac{T}{nop}$ snapshots of the *AS-Sequence* to each processor. (2) In each processor, summarize *Act-snapshots* into *C-graphs*. (3) In each processor, extract the features of *C-graphs*. Note that the parallelization scheme needs no synchronization step which makes it extremely efficient.

Generate the segmentation graph $\mathsf{G_s}$

If the *AS-Sequence* is long and has many *Act-snapshots*, merely using the parallelization suggested in Section 4.4.4 will not be enough to get the best segmentation fast. Thus, we propose a method to generate the segmentation graph $\mathsf{G_s}$ in parallel. We consider the following parallelization scheme to generate $\mathsf{G_s}$: Assume that *C-graphs* features are global information, there are *nop* processors available and $\mathcal{S}$ is the set of all possible segments. (1) Assign $\frac{|\mathcal{S}|}{nop}$ segments to each processor. (2) Next, compute the feature representative $\hat{\mathbf{F}}$ of each segment in each processor. The $\hat{\mathbf{F}}$ of a segment $s_{i,j}$ is the average vector of all the *C-graphs* features that are in the segment $s_{i,j}$ (4) Synchronize the results and compute the edge weights of the segmentation graph $\mathsf{G_s}$.

4.4.5 The complete algorithm

In summary SnapNETS has four main steps:

(1) Summarize *Act-snapshots* in *AS-Sequence*.

(2) Extract features from summarized graphs (*C-graphs*).

(3) Construct the segmentation graph $\mathsf{G_s}$ and compute the distance $d(s_{i,j}, s_{j,k})$ between any adjacent segment based on their representatives ($\widehat{\mathbf{F}}_{s_{i,j}}$ and $\widehat{\mathbf{F}}_{s_{j,k}}$) in a parallel manner.

(4) Compute the best segmentation by finding the ALP.

Algorithm 5 shows the final pseudo code of SnapNETS.

Lemma 12 *SnapNETS has sub-quadratic time complexity $O(E \cdot \log E + \frac{E}{nop} + E_s)$, where E is the total number of edges in $\mathcal{G}$ and nop is number of processors.*

Proof 4 *In step 1 and 2, feature extraction is very fast due to small size of the C-graphs compared with Act-snapshots. Hence the computational cost comes from coarsening which is $O(E \log E + \frac{E}{nop})$: $O(E \log E)$ to sort edges [130] and $O(\frac{E}{nop})$ for parallel coarsening. Step 3 and step 4 take $O(E_s)$. Generating the segmentation graph $\mathsf{G_s}$ is $O(\frac{E_s+V_s}{nop})$ and finding the average longest path is $O(E_s)$ which is $O(T^3)$. In most large real world datasets (e.g. see [144]), the size of each graph (orders of tens of thousands) is usually much higher than the length of the sequence (order of 100s). Therefore, in practice, $O(E_s)$ is not the bottleneck of SnapNETS and we can assume the last two steps take $O(E)$. Total time complexity is $O(E \cdot \log E + \frac{E}{nop} + E_s)$. Note that it does not change even if the size and structure of Act-snapshots change in the AS-Sequence.* □

Even though, the time complexity of step 3 and 4 step is $O(T^3)$ due to generating the E_s and Layered-ALP, the computation is fast in practice (i.e., in order of seconds). Hence, it is not the bottleneck of the SnapNETS and taking the algorithm as a whole SnapNETS is scalable. The reason is in real-world graph sequences $T << E$ (i.e. $T \sim O(100)$ while $E \sim O(1e+6)$). Hence, summarizing real-world graphs takes minutes or hours to complete while generating the $\mathsf{G_s}$ and Layered-ALP take only a few seconds.

4.5 Extending to dynamic graphs

SnapNETS gives *Cut-points* which capture the change of pattern in the *AS-Sequence* $\mathcal{G}$. Even though we assumed the structure of *Act-snapshots* in the *AS-Sequence* is fixed, we propose a procedure to extend SnapNETS to handle dynamic structures as well. In the following we explain this procedure in detail.

4.5.1 Technical details

If the structures of *Act-snapshots* are dynamic (i.e. *Act-snapshots* have various number of nodes and edges), summarizing them with the same reduction factor gives *C-graphs* with different sizes. Therefore, the feature values (table 4.3) of *C-graphs* may be biased by the size of *Act-snapshots*. Thus, we can not compare *C-graphs* and compute distance between segments by simply comparing their feature vectors. We want to find an efficient way to make the *C-graphs* be in a same size and their feature vectors comparable. Towards this goal, we need different compression ratios ρ for each *Act-snapshot*: We summarize smaller graphs with smaller reduction factor and larger ones with larger ρ. Algorithm 7 shows how to compute the compression ratio of each *Act-snapshot*. First, we find the minimum and

Algorithm 6 LAYERED-ALP

Require: $\mathsf{G_s}$, $\mathcal{G}$, s, t

1: Initialize Queue
2: $Layer_0 = \{t\}$ and Queue.push($Layer_0$)
3: $Layer_1 = \{s_{i,j} | j = T\}$
4: Queue.push($Layer_1$)
5: **for** k = 2 **to** T **do**
6: $\quad Layer_k = \{s_{i,j} | T - j + 1 \geq k\} \cup \{s\}$
7: $\quad$ Queue.push($Layer_k$)
8: $Layer_{T+1} = s$ and Queue.push($Layer_{T+1}$)
9: LL = Queue.pop(), $\kappa_t^0 = \varnothing$
10: **while** Queue is not empty **do**
11: $\quad$ CL = Queue.pop()
12: $\quad$ **for** $s_{i,j} \in$ CL **do**
13: $\quad\quad \kappa_{s_{i,j}}^{CL} = \arg\max_{s_{j,k}} P_{LL}(s_{j,k}, t) + w(e(s_{i,j}, s_{j,k}))$
14: $\quad\quad P_{CL}(s_{i,j}, t) = P_{LL}(\kappa_{s_{i,j}}^{CL}, t) + w(e(s_{i,j}, \kappa_{s_{i,j}}^{CL}))$
15: $\quad$ If s is in CL then $lp_{CL} = P_{CL}(s, t)$
16: $\quad$ LL = CL
17: $ALP = \arg\max(\frac{lp_1}{1}, \ldots, \frac{lp_T}{T})$
18: Extract the P_{alp} using κ_s^T to κ_t^0
19: **return** P_{alp}

maximum number of nodes in the $\mathcal{G}$. The largest *Act-snapshot* in the $\mathcal{G}$ is coarsened the most to have the same size as the other *Act-snapshots*. The 'if' statement in line 2 of the algorithm, makes sure that the size of all *C-graphs* will be the same. Finally, in lines 6 to 7 we set the ρ_i values to get the number of desired nodes. More complicated methods to compute the ρ_i of the dynamic graph snapshots may improve our results but our approach is a simple and straightforward way to achieve this goal, and the experiments show it works properly.

Lemma 13 *Time complexity of Algorithm 7 to set the ρ values is $O(T)$.*

Proof 5 *Finding the minimum and maximum number of nodes in the $\mathcal{G}$ (line 1) takes $O(T)$. Also, the if statement (lines 2-5) take $O(1)$. Finally, the 'for' loop runs in $O(T)$. Therefore, in overall the time complexity of Algorithm 7 is $O(T)$.* □

In summary, to extend SnapNETS and handle dynamic $\mathcal{G}$, we compute the ρ values (Algorithm 7) in the first step of SnapNETS and the rest will be the same as in Algorithm 5.

Algorithm 7 SET-ρ

Require: $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$

$V_{max} = \max_{i=1,\ldots,T} |V_i|, \quad V_{min} = \min_{i=1,\ldots,T} |V_i|$

if $0.1 \cdot V_{max} \leq V_{min}$ **then**

  $V_{goal} \leftarrow 0.1 \cdot V_{max}$

else

  $V_{goal} \leftarrow V_{min}$

for $i = 1, 2, \ldots, T$ **do**

  $\rho_i = 1 - \frac{V_{goal}}{|V_i|}$

return $\rho_1, \rho_2, \ldots, \rho_T$

Algorithm 8 ANOMALY-SNAPNETS

Input: $\{G_1, G_2, \ldots, G_T\}$ **Output:** R $\rho_1, \rho_2, \ldots, \rho_T$ =SET-$\rho(G_1, G_2, \ldots, G_T)$ For each G_i coarsen with ρ_i and get G_i^c in parallel Compute feature $\widehat{\mathbf{F}}$ vector of segments in $\mathcal{S}$ in parallel Generate the segmentation graph $\mathsf{G_s}$ c^* = Parallel-LAYERED-ALP ($\mathsf{G_s}$, $\mathcal{G}$, s, t)

4.5.2 Sample application: Anomaly and event detection

An example application of the segmentation problem is to solve the anomaly (and event) detection problem which is defined as follows,

Given a sequence of graphs $\{G_1, G_2, \ldots, G_T\}$

Find a list R of time points in $1 \leq i \leq T$, as anomalies. In the above problem, nodes do not have labels and there is no restriction on the structure of the graphs in the sequence. As a consequence, graph size may change over time. We build the following framework to solve the same problem using SnapNETS:

> **(1) Summarize *Act-snapshots*.** The size of *Act-snapshots* can be different. Hence, as we explained in Section 4.5 we compute the compression ratio of each graph according to Algorithm 7. Next, we summarize each graph, assuming all nodes have the same label, to get *C-graphs* with the same size.
> **(2) Find c^* as R.** We follow the same procedure as SnapNETS to find c^* as R.

Algorithm 8 indicates the anomaly detection algorithm using SnapNETS.

Justification: The detected *Cut-points* by SnapNETS is an accurate solution because SnapNETS detects important time points in the *AS-Sequence* where the structure of graphs change which in turn can be considered as anomalies (or important events).

4.6 Experiments

We design various experiments to evaluate SnapNETS. We implemented SnapNETS in Python. Our experiments were conducted on a 4 Xeon E7-4850 CPU with $512GB$ of $1066Mhz$ main memory and we released our code and datasets for research purposes[3].

4.6.1 Setup

We design experiments to answer the following questions:

Q1. Is the coarsened graph a good summary for our segmentation problem?
Q2. How is the quality of our feature set?
Q3. What is the difference between ALP and other path optimization algorithms?
Q4. What are the segmentations found by SnapNETS? Do they discover useful and interesting patterns?
Q5. Can Anomaly-SnapNets find important events and anomalies in dynamic graph sequences?
Q6. Is SnapNETS scalable to run on real large datasets?

4.6.2 Datasets

For our experiments, we collected a number of real-world and synthetic datasets from various domains such as social and news media, epidemiology, autonomous system, and co-authorship network to evaluate SnapNETS. See Table 4.4 for a summary description.

The ground truth segmentations in these datasets are non-trivial. They are induced from complex structural changes such as activation in different parts of the *Act-snapshots* (AS Oregon and Higgs), and varying role of active nodes e.g. change of active nodes centrality (BA-degree), or activation rate changes (Portland). Moreover, detecting the number of correct cut points is also a difficult task itself. In datasets without ground truth, we would like to find how memes/news were adopted by social media users (IranElect and Memetracker) and when the co-authorship relation on a specific topic changes over time (DBLP).

(1) BA-degree. We activate highest degree and then lowest degree nodes on a random Barabasi Albert graph [22].

(2) AS Oregon contains an Autonomous Systems (AS) peering information network collected from the Oregon router views[4]. We generate three simulations: AS Oregon-PA,

[3]http://github.com/SorourAmiri/SnapNETS
[4]http://topology.eecs.umich.edu/data.html

Dataset	#Nodes	#Edges	Timesteps	GT
BA-DEGREE	500	4,900	240 units	✓
AS-PA	633	1,086	400 units	✓
AS-COM	4431	7,609	530 units	✓
AS-MIX	1899	3261	1000 units	✓
HIGGS	456,626	14,855,843	7 days	✓
PORTLAND	1,575,861	19,481,626	25 days	✓
MEMETRACKER	960	5,001	165 days	
IRANELECT	126,915	5,589,083	30 days	
DBLP	369,855	1,109,452	13 years	
WORLDCUP	140,336	674,077	30 days	✓
SECURITY	308,499	1,182,021	90 days	✓
ENRONINC.	82,797	349,780	111 weeks	✓

Table 4.4: Datasets details. (GT == Ground Truth)

AS OREGON-COM and AS OREGON-MIX.: (a) AS-PA: First we activate nodes with **P**referential **A**attachment process and then uniformly randomly. In the next two simulations, we created three copies of AS OREGON and connected them with bridge edges. We assume each copy is a community in the combined graph. (b) AS-COM: we connect 7 copies of the original network and combine them into a large network with 7 communities. We start activating 3 communities at different times with the same PA activation process. (c) AS-MIX: It is a combination of the above two. Each community gets activated at different time, and the activation in each community has two stages with different patterns (i. e. PA and random stages).

(3) HIGGS [44] is a related tweets dataset (with the follower-followee network) when the Higgs particle was discovered. Between 1st and 7th July 2012 several announcements were made about the discovery of the Higgs particle. Nodes of the graph (i.e. twitter users) are active when they participate in related activities (e.g. retweeting, replying). The ground truth cut is the official release date of the discovery.

(4) PORTLAND. has a contact network among a synthetic population of Portland, and this dataset has been used in national smallpox modeling studies [53]. We simulate a chain shape diffusion and then a star shape one in a synthetic contact network of Portland[5].

(5) IRANELECT contains the tweets and the follower-followee network of Twitter [96], during the 2009 Iran election. Users are the nodes of the network and they are active if they post a related tweet. We want to find how the news were adopted by twitter users.

(6) DBLP is a co-authorship network [98] related to the topic "network". Authors are nodes, and edges show the co-authorship relations. An author is active if she co-authors a related

[5]http://ndssl.vbi.vt.edu/synthetic-data/

paper with "network" or its derivations in its title. We want to find how the co-authorship of this topic develops over time.

(7) MEMETRACKER. It is the who-copies-from-whom blog and website network [101]. We consider a blog/website as active if it adopts the phrase 'hey can I call you joe'. We want to find how the adoption pattern changes.

To evaluate Anomaly-SnapNets we collect three real world temporal graph datasets. The ground truth events of all datasets are available [6].

(1) WORLDCUP It is the World Cup twitter dataset (Jun 12 - Jul 13, 2014). Nodes represent entities (i.e. URLs, hashtags, mentions) and edges show the co-mention relation [132].

(2) SECURITY It is a twitter data of four months (May 2 - August 1, 2014) related to the security and terrorism. Nodes represent entities (i.e. URLs, hashtags, mentions) and edges show the co-mention relation [132].

*(3)*ENRONINC. It is the Enron email communication network from Jan 2000 to March 2002. Nodes represent email addresses and edges shows the sent/received relations [132].

4.6.3 Baselines

To the best of our knowledge, there is no existing algorithm which has all the desired properties (Section 4.3.2). Hence, we adapt three alternative approaches that we also mentioned in Section 4.3.1 as baselines: time series, clustering, and graph summarization algorithms.

(1) Dynammo uses the spike of reconstruction errors to find change points in time series [106]. It has also been used in other time series problems [111]. We adapt it for our problem: Extract features in Table. 4.3 for each *Act-snapshot*. Then, use these features to form a time series and feed Dynammo to get the segmentation. The algorithm requires number of cut points as an input and we set it to the one SnapNETS finds.

(2) K-means finds segmentations based on an online "local" approach [107]. We make the same time series as in Dynammo, and feed it to K-means. We report a new segment when a new cluster is detected. We consider it as the emergence of a new pattern and include the corresponding cut point in our final segmentation.

(3) VOG finds succinct descriptions of graphs in terms of common substructures [95, 144]. It does not work on labeled graphs. Hence, we extract active *sub-graph* of *Act-snapshots* and use VOG to find the 10 most important sub-structures. We output a segment if this set of sub-structure changes sufficiently (i.e is above a threshold which is set to be the one which gives the best results). Also, we use VOG as a baseline to evaluate the Anomaly-SnapNets. In this case, the input of VOG is each snapshot of the temporal graphs.

[6] http://shebuti.com/SelectiveAnomalyEnsemble/

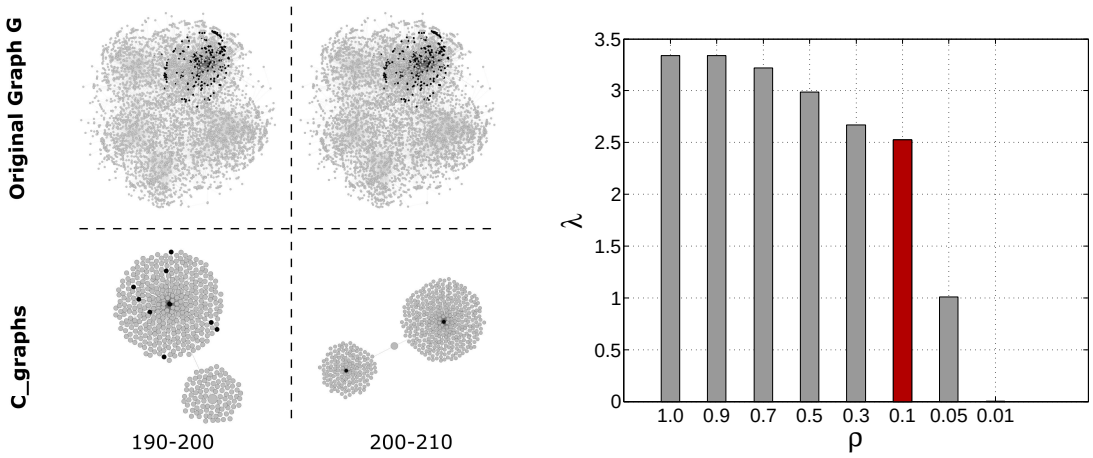


Figure 4.4: (a) *C-graphs* capture patterns that are not captured by the original graphs. (b) λ vs ρ shows 0.1 is the smallest ρ which maintains λ.

(4) SELECT is an ensemble approach for anomaly mining. It leverage novel techniques to rank anomalies in temporal graphs. We take top $k = 5$ events as the detected anomalies and compere its result with Anomaly-SnapNets.

Variations: Besides the above baselines, we design the following variations to test the functionality of each SnapNETS's component. These variations are different from SnapNETS only in one step.

(1) SN-Orig extracts features from *Act-snapshots* and use the same $\mathsf{G_s}$ and ALP optimization to get the segmentation.

(2) SN-LP finds the **L**ongest **P**ath instead of ALP.

(3) SN-Greedy greedily selects edges with the largest weight from s to t, instead of ALP.

Metrics. For datasets with ground truth, we measure the performance by calculating the F_1 score and precision of the set of detected cut-points with the ground-truth set (using the same methodology as [111]). For others, we perform case studies. We show how SnapNETS reveals interesting patterns by matching the results with external news/events, and show how they are easily interpretable.

4.6.4 Q1: Usefulness of C-graphs

We compare the performance of SnapNETS with SN-Orig to show the effectiveness of our summarization. See Table 4.5, SnapNETS outperforms SN-Orig: In most datasets SnapNETS discovers the correct cut points (F_1 *score* $= 1$), while SN-Orig misses them. Hence clearly the *C-graphs* lead to better segmentation than the original *Act-snapshots*. Also, the *C-graphs*

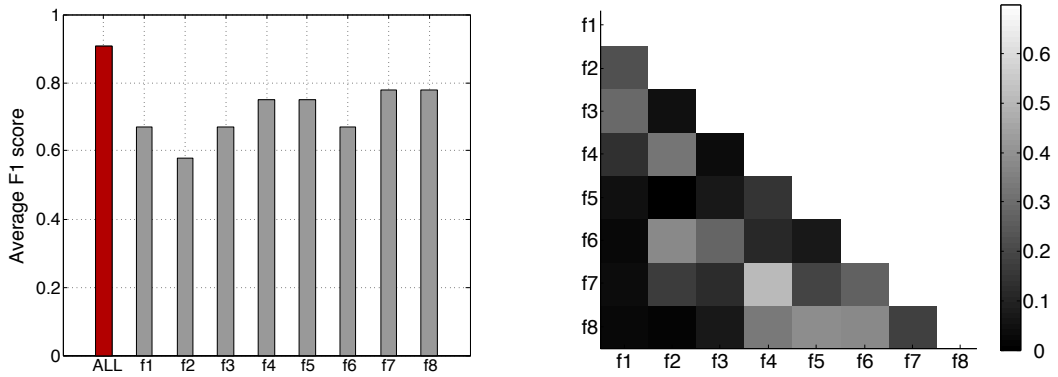
not only maintain the same pattern of the original *Act-snapshots*, but also discover important new patterns and help interpretation via the much simpler graph structure. For example, see figure 4.4a: the *C-graphs* of AS-COM (bottom row) capture clearly the pattern: first one community gets active, then two communities are active after the ground truth cut point 200. This pattern is hardly observable in the original *Act-snapshots* (top row).

Also, we evaluate the largest eigenvalue of summary graphs of the MEMETRACKER dataset with various compression ratio to show the effect of ρ on λ. Figure 4.4b, confirms our intuition behind selecting the ρ value in section 4.4.1 that 0.1 is a desired value for ρ: It is the smallest ρ which does not change the largest eigenvalue of the graph much.

4.6.5 Q2: Quality of feature set

Here we show that the current feature set $\mathbf{F}$ we use is of high quality. We did the ablation test [149] to study the impact of each feature in the feature set $\mathbf{F}$. First we run SnapNETS with the complete $\mathbf{F}$, we then remove one feature at a time from $\mathbf{F}$ to see the change of performance. In addition, we compute the correlation between each pair of features in $\mathbf{F}$.

The ablation test on the datasets with ground truth shows an average F_1 decrease of 0.2 (See Figure 4.5a). We also tried the same test on datasets without ground truth, and we observe unreasonable performance when we remove one of the features. For example, removing any of f_2, f_5, f_6 would lead to over-segmenting in DBLP; removing f_3 leads to an unreasonable cut point at the end of the sequence in IRANELECT. These results indicate the importance of each feature in $\mathbf{F}$. In addition, Figure 4.5b shows the correlation between pairs of features in $\mathbf{F}$ in all datasets (i.e. with or without ground-truth). It shows that most of the pairs has near zero correlation in at least one dataset and there is no pair with more than 0.5 correlation and on average the correlation is 0.17. These results show the high quality of $\mathbf{F}$.



(a) (b)

Figure 4.5: (a) The ablation test. (b)The correlation between features among all datasets. Darker color means less correlation.

4.6.6 Q3: Finding the best path

Here, we evaluate the quality of ALP by comparing it to SN-LP and SN-Greedy. See Table 4.5: SnapNETS is much better than SN-LP and SN-Greedy in all the datasets. We also check the number of segments found by SN-LP. As we expected (Section 4.4.3), it leads to over-segmentation. For example, in MEMETRACKER and IRANELECT SN-LP detects cut point at every snapshot, whereas SnapNETS gets the right answer every time.

4.6.7 Q4: Segmentation Results

Here we show the quality of the final segmentations found by SnapNETS.

Quantitative analysis

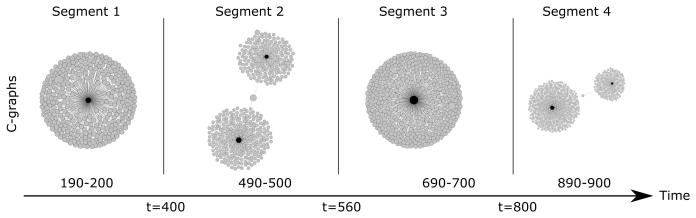
Table 4.5: F_1 score of the segmentation detected by SnapNETS, variations, and baselines on datasets with ground truth. SnapNETS has the best performance among all competitors.

Data w. GT	F_1 score						
	SnapNETS	SN-Orig	SN-LP	SN-Greedy	Dynammo	K-means	VOG
BA-DEGREE	**1**	**1**	0.08	**1**	0	0.4	0.67
AS-PA	**1**	0	0.05	0.67	0	0.4	**1**
AS-COM	**0.67**	0	0.07	0.5	0.5	0.22	0
AS-MIX	**0.86**	0	0.07	0.57	0.32	0.4	0
HIGGS	**1**	0	0.15	**1**	0	0.67	0
PORTLAND	**1**	0	0.4	**1**	**1**	0.67	**1**

We see in Table 4.5 that SnapNETS has the best performance among baselines and variations of SnapNETS. Note that all the baselines require some input parameters: such as the number of cut points (Dynammo), threshold for new cluster creation (K-means), and difference threshold for outputting a cut point (VOG). We test a wide range of these input values and pick the ones that give the best results. Still, their performance is clearly worse.

BA-degree: SnapNETS detects the ground truth segmentation while three other baselines do not perform as well. , when the activation starts to target low degree nodes instead of high degree nodes. As shown in Table 4.5, the F_1 *score* of SnapNETS is 1. In comparison, the three other baselines do not correctly find the cut points where the pattern changes.

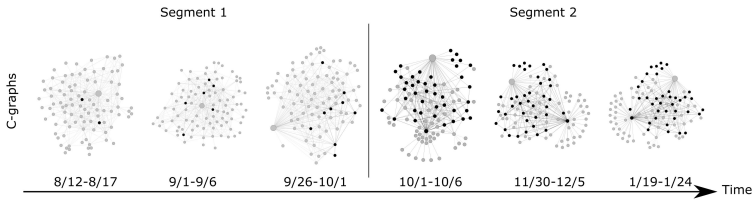
AS Oregon: SnapNETS performs very well to differentiate between random and preferential attachment style activation (AS-PA) and we can accurately detect when different communities of the network get active (AS-COM and AS-MIX). Figure 4.6a visualizes the *C-graphs* in the c^*, and shows that our results are easily interpretable. In AS-PA dataset, SnapNETS



(a) Visualization of *C-graphs* in the segmentation for AS Oregon-MIX.

Feature	Seg. 1	Seg. 2	Seg. 3	Seg. 4
f1	0.32	0.28	0.90	0.76
f2	0.12	0.36	0.57	0.77
f3	0.50	0.65	0.83	0.87
f4	0	0	0	0
f5	0	1	0	1
f6	0.91	0	1	0
f7	0.26	0.14	0.87	0.42
f8	0	0	0	0

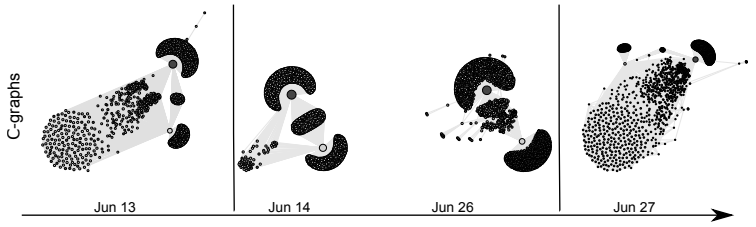
(b) The average feature values in each segment for AS Oregon-MIX.



(c) Visualization of *C-graphs* in the segmentation for Memetracker.

Feature	Seg. 1	Seg. 2
f1	0.93	0.06
f2	0.97	0.02
f3	0.09	0.97
f4	0.12	0.87
f5	0.10	0.95
f6	0.22	0.91
f7	0.45	0.76
f8	0.20	0.95

(d) The average feature values in each segment for Memetracker.



(e) Visualization of *C-graphs* in the segmentation for IranElect.

Feature	Seg. 1	Seg. 2	Seg. 3
f1	0.79	0.11	0.85
f2	0.56	0.04	0.81
f3	0.11	0.73	0.047
f4	0.92	0.14	0.82
f5	0.05	0.33	0.92
f6	0.19	0.70	0.01
f7	0.69	0.06	0.63
f8	0.21	0.20	0.89

(f) The average feature values in each segment for IranElect.

Figure 4.6: Interpretation of the segmentation results found by SnapNETS. (a), (c), (e) show the *C-graphs* for the segmentations found for AS Oregon-MIX, Memetracker, and IranElect. The vertical lines are the detected cut points. Black nodes are active and gray ones are inactive. (b), (d), and (f) show the average feature values in each of the segment for AS Oregon-MIX, Memetracker, and IranElect.

finds the ground truth cut point (F_1 *score* = 1). In the more complicated AS-COM and AS-MIX, it also does very well, getting F_1 *score* of 0.67, 0.86 respectively.

SnapNETS also aids in interpretation: we visualize the AS-MIX segmentation in Figure 4.6a. We see clearly that the *C-graph* in the segment 2 captures the fact that there are two communities activated. As we activate more nodes in bridge, the active nodes in the two communities are merged to one node in the *C-graph* (segment 3). And finally when the third community gets activated, the *C-graph* again captures it as a new community (segment 4). The corresponding feature values of *C-graphs* also correctly reflect these changes (while the features from original graphs do not). Note that although the active subgraphs are similar in segments 1 and 3, and segments 2 and 4, the segments are actually different if we look at the whole graph including the inactive nodes (this difference is also seen in the feature values). These results show that SnapNETS can detect non-trivial changes, including node-level, process-level or community-level changes.

Higgs: SnapNETS finds the exact ground truth Jul. 4 ($F_1 = 1$). Note that according to external news, there were rumors about the Higgs boson discovery from Jul. 2-4, these rumors make the ground truth harder to detect (for example VOG finds a cut point on Jul. 2 instead of Jul. 4). Further in the first segment, *C-graphs* have multiple near-clique structures of *active* nodes ($f_1 \simeq 0.9$, $f_5 = 0.2$) which demonstrates the existence of a rumor in the network (multiple small groups adopt the news). In comparison in the second segment, *C-graphs* become chain-like with many active nodes ($f_1 \simeq 0.02$, $f_5 = 0.9$). It suggests that many communities adopt the news (with nodes in the same community merged), and few bridge (inactive) nodes connect different communities, matching our expectation since the official announcement is evidently more influential.

Portland: SnapNETS again detects the ground truth segments ($F_1 = 1$). Other baselines have worse performance except VOG and Dynammo since the change of the infection pattern in this dataset is visible in the active subgraphs: an infected chain first, and then many infected stars (as the disease goes viral). SnapNETS shows its power in finding the pattern change in disease propagation.

Case studies

SnapNETS finds meaningful segments in multiple datasets from various domains with varied patterns of evolution (both structurally and in labels) while none of the baselines perform as well (for example, VOG finds no cut-point in DBLP, K-means essentially gives one at the end, and Dynammo finds no cut-point in MEMETRACKER).

Memetracker: SnapNETS finds a cut point on Oct 01, 2008, which matches the date of the televised debate between Joe Biden and Sarah Palin. In the first segment, *C-graphs* (Figure 4.6c) are close to the case when all nodes have the same label—suggesting that few nodes randomly got infected ($f_5 \simeq 0.1$, $f_8 \simeq 0.2$). Few active nodes in the first segment of

Figure 4.6c confirms this fact and its because the meme is not viral yet. In the second segment, *C-graphs* are substantially sparser (i.e. f_2 dramatically dropped to 0.02) and contain large stars and leaf nodes with active centers. The size of the centers in the second segment of Figure 4.6c and average PageRank (f_6) in the *C-graphs* shows that important nodes such as "CNN" and "BBC" websites spread the meme to many nodes, and they are merged to form hubs.

IranElect: We detect two cut points at Jun. 14 (presidential election) and Jun. 27 (vote recount). In the first segment (Figure 4.6e), when the election did not get much attention, multiple small *near-clique* structures (high $f_1 \simeq 0.8$ and low $f_2 \simeq 0.5$) of *C-graphs* shows small and highly connected groups of political activists were active. In the second segment, important nodes (e.g. news media such as "NYtimes") in different areas of the graph reported possible collusion in counting votes and they became active and formed multiple large stars of active nodes in *C-graphs* (low $f_1 \simeq 0.1$ and high $f_6 \simeq 0.7$). Finally, after the vote recount and news reports more people became interested and tweeted about the election. Hence, *C-graphs* became densely connected ($f_1 \simeq 0.85$, $f_2 \simeq 0.8$) because the remaining hub and bridge nodes became active and merged, while a few small/sparse subgraphs remained inactive.

DBLP: We detect a cut point in the year 1997, which matches the publication time of the ground-breaking papers in network science [163, 54, 53, 22]. In the first segment, *C-graphs* are relatively connected, and few less-central nodes are active ($f_1 = 0.7$, $f_5 = 0.03$, $f_6 = 0.2$) which indicate the "network" topic did not get much attention. In the second segment, the number of active nodes in *C-graphs* increases dramatically, and significant nodes with high degree are among the active ones ($f_5 = 0.6$, $f_6 = 0.8$, $f_7 = 0.83$); which suggests influential people became interested in "network" and made it into a "hot topic"—high $f_8 = 0.9$ also confirms this fact that many nodes got active since one high degree neighbor of them is active.

4.6.8 Q5: Anomaly detection: Another application

We measure the quality (precision) of the segmentation by Anomaly-SnapNets baseline in three real-world datasets to evaluate its performance in detecting anomalies and important events. Table 4.6 shows that Anomaly-SnapNets has the best performance compared to baselines. Note that SELECT is specifically designed for anomaly detection and VOG needs a threshold as an input (Also, VOG did not finish and ran out of memory in two cases). However, we detect the anomaly automatically without any preset parameter or user interaction, showcasing SnapNETS's usefulness.

WorldCup Anomaly-SnapNets detects Jun 16, July 7 and July 10 as cut-points. Jun 16 is the match between Germany and Portugal, one of the most important games in group matches according to the ground truth. The July 7 and 10 cut-points distinguish the semi-final games and the final game.

Security Anomaly-SnapNets accurately finds the important events at Jun 11 and July 29, 2014. Jun 11, 2014, matches the date when large regions of Iraq was seized by Iraqi militants and the attack of 'Boko Haram' in Nigeria and the following kidnapping of schoolgirls took place. July 29, 2014, matches the date of Ebola virus outbreak in Liberia.

EnronInc. We detect a cut point on Oct. 15 which matches the major anomaly when the Enron company announced a third-quarter loss. This is also recognized in SELECT [132] as a major anomaly.

Dataset	Anomaly-SnapNets	VOG	SELECT
WorldCup	**1**	0	**1**
Security	**1**	-	0.8
EnronInc.	**1**	×	0.8

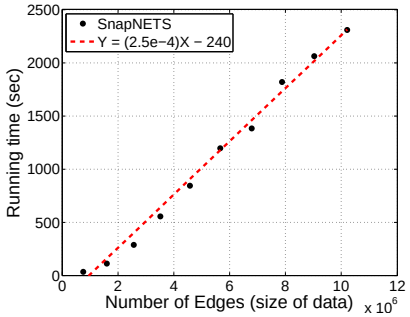
Table 4.6: The precision of detected anomalies by Anomaly-SnapNets and other baselines. '-' means the method ran out of memory and '×' means it did not finish after 4 days.

4.6.9 Q6: Scalability

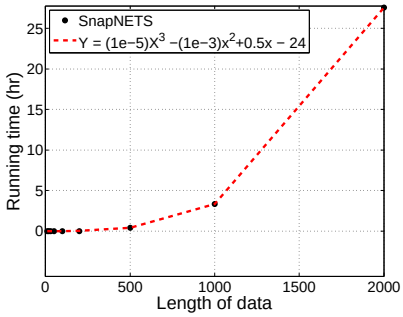
Each step of SnapNETS is carefully designed to be memory efficient and sub-quadratic or near linear with respect to the size of data. We extract subgraphs with different sizes from DBLP (up to 10M edges), then we run SnapNETS on them. Figure 4.7a shows that as expected SnapNETS scales near-linearly with respect to the number of edges in the sequence. The running time of SnapNETS shown in Figure 4.7a is reasonable and practical, especially considering that dynamic graph analysis are typically time consuming. For example, [144] takes more than 100 hours to run on a dataset with size $10M$ while SnapNETS takes *less than an hour* (i.e. it is ~ 10 *times* faster).

Also, we evaluate the scalability of generating $\mathsf{G_s}$ and Layered-ALP in Figure 4.7b. We generate a series of $\mathcal{G}$ of DBLP data with various lengths (up to 2000 snapshots), then we run SnapNETS on them. Figure 4.7b confirms that generating $\mathsf{G_s}$ and Layered-ALP are not the bottleneck of SnapNETS. Also, it shows that our algorithm is much faster than the state-of-the-art algorithms.

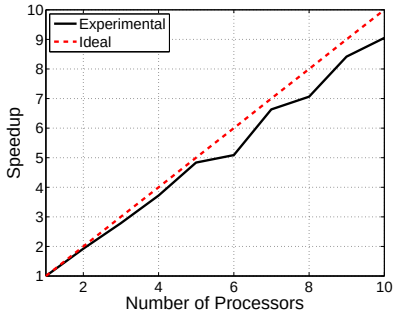
Figure 4.7c shows the excellent speedup ($\sim$ 9X faster using 10 processors) we get after parallelizing the *Act-snapshots* summarization and feature extraction. Also, Figure 4.7d shows the speedup of making the segmentation graph in parallel ($\sim$ 8X using 10 processors). In summary, SnapNETS makes it possible to process a large network sequences with many snapshots.



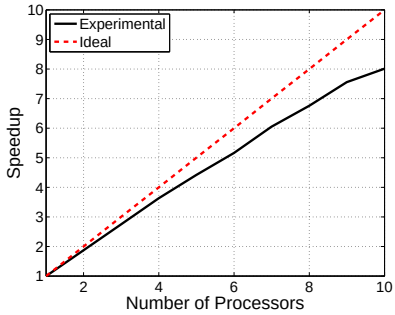
(a) Scalability W.R.T the size of $\mathcal{G}$



(b) Scalability W.R.T length of $\mathcal{G}$



(c) Parallel summarization speedup.



(d) Parallel $\mathsf{G_s}$ speedup.

Figure 4.7: (a) and (b) shows the scalability of SnapNETS with respect to the size and length of the $\mathcal{G}$. (c) and (b) show the speedup by parallelizing summarizing *Act-snapshots* and constructing $\mathsf{G_s}$.

4.7 Discussion and Conclusions

We presented SnapNETS, an intuitive and effective method to segment *AS-Sequences* with binary node labels and extended it to work with dynamic graph sequences as well. It is the *first* method to satisfy all the desired properties **P1**, **P2**, **P3**. It efficiently finds high-quality segmentations, detects anomalies and events and gives useful insights in diverse complex datasets. Also, we propose Anomaly-SnapNets as an expansion of SnapNETS to detect anomalies and important events in dynamic graph sequences. Finally we parallelize SnapNETS and Anomaly-SnapNets to accelerate the computations.

Patterns it finds: In short, SnapNETS finds segmentations where adjacent segments have different characteristics of nodes with the same label i.e. the 'placement' and 'connection' of active/inactive nodes are different. This includes both structural (e.g. community/role/centrality) and rate changes. As a non-trivial example, we can detect both a random-vs-targeted activation and a faster-vs-slower one. Also, Anomaly-SnapNets detects segments with different structural characteristics such as the density, the growing speed of

the network, etc.

Global method: It is useful to note that SnapNETS is a 'global' method and not simply a change-point detection method. We are not just looking for local changes; rather we track the 'total variation' using $\mathsf{G_s}$. Hence this allows to find important cut-points automatically and without any specification (which is useful for anomaly detection as well).

Flexibility: The SnapNETS framework is very flexible, as our formulations are very general. Thanks to our careful design we easily expand our method to handle dynamic graphs with varying nodes and edges and propose Anomaly-SnapNets. The eigenvalue characterization is general; similarly, the $\mathsf{G_s}$-ALP formulation should be also useful for other segmentation-like problems; and Layered-ALP can be of independent interest too.

Future work: Parallelizing Layered-ALP and extending our work to streaming graphs, and handling more general node/edge level features and partially observed graphs will be interesting. Also, analyzing the effect of using different compression ratio in Anomaly-SnapNets is an interesting future work.

Chapter 5

Attributed networks for influence based tasks

We continue to summarize networks with optimization-based approaches. In this chapter, we study the summarization of attributed diffusion networks. Diffusion networks with user attributes such as friendship, email communication, and people contact networks are increasingly commonplace in the real-world. Given a large attributed social network, can we find a compact, diffusion-equivalent representation while keeping the attribute properties? We first formally formulate a novel problem of summarizing an attributed diffusion graph to preserve its attributes and influence-based properties. Next, we propose ANeTS, an effective sub-quadratic parallelizable algorithm to solve this problem: it finds the best set of candidate nodes and merges them to construct a smaller network of 'super-nodes' preserving the desired properties.

Extensive experiments on diverse real-world datasets show that ANeTS outperforms all state-of-the-art baselines (some of which do not even finish in *14 days*). Finally, we show how ANeTS helps in multiple applications such as Topic-Aware viral marketing and sense-making of diverse graphs from different domains.

5.1 Introduction

Suppose we are monitoring network-traffic in a computer system. Legitimate user inputs such as keyboard and mouse events start a series of network requests. Some of these network requests are malicious and they are sent when a user visits a compromised website or the host is infected by malware. This data can be represented by an attributed diffusion graph [177], where nodes are user input and network requests and their attributes are their types and time-stamps. There is an edge between two nodes when one node requests the other one (See Fig. 5.1(a) and (c)). Can we find malicious network requests as anomalies in this network?

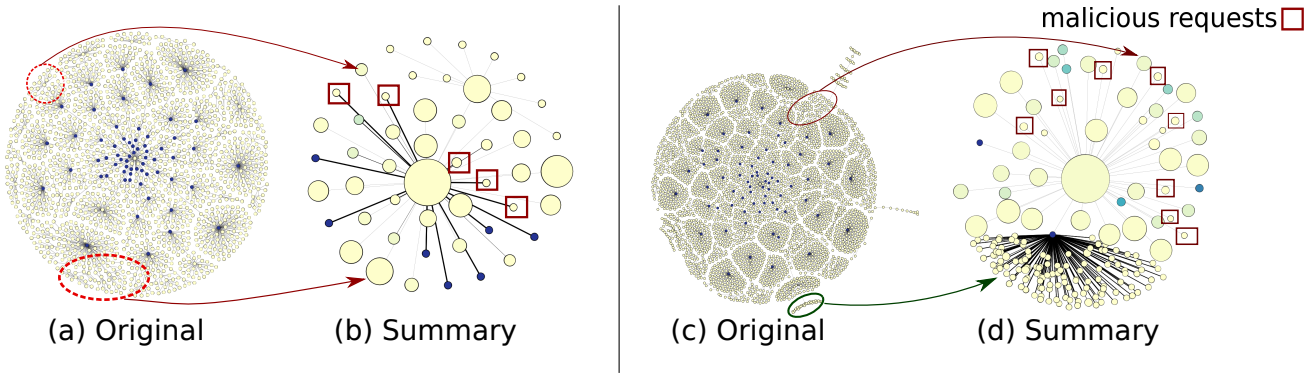


Figure 5.1: Finding malicious requests. (a, c) are original NETWORK-TRAFFIC graphs of two different users. Dark blue nodes are type 'user-input' and yellow are type 'network-request'. Note we do not show the time-stamp attributes here. (b, d) show summary graphs generated by ANeTS. Color of each supernode is the combination of the color of nodes inside it. Size of super-nodes $\propto$ # merged nodes and red squares highlight some of the malicious nodes found.

Further, can we also understand the connection structure of these malicious nodes? These insights can help security analysts to efficiently evaluate the network-traffic dependency and perform further forensics. However, malicious nodes can be triggered anytime and anywhere in the network. This combined with the large size of the graph and sparsity of the malicious requests (0.3% of the total requests [176]) make finding them extremely hard in the Network-traffic graph.

Attributed influence/diffusion networks are commonly seen in other domains as well. Email communication, citation, product co-purchase, and social networks are some other examples. Understanding and analyzing such networks is essential for many applications including viral marketing, immunization, anomaly detection, pattern finding and making sense of propagation. However, the increasingly larger sizes of such networks make this difficult. While for many tasks there are very efficient algorithms for plain networks (such as the influence maximization problem [36]) scaling to millions of nodes, the corresponding state-of-the-art algorithms (e.g. [35]) on attributed networks are slow and can not scale even to graphs with $50k$ nodes (see experiments). Summarizing these attributed networks to get a smaller representation can help us to easily scale-up these data-mining tasks. By finding such summaries, we can better visualize and understand the dynamics of the network, and speed-up attribute and influence-related analysis. The idea would be to run the analysis on the much smaller summary, saving running time, instead of doing it on the entire graph. We study the novel problem of summarizing influence-networks where nodes have features. Despite their importance, there is surprisingly less work on summarizing attributed graphs (see Table 5.1 and related work). Also, dealing with attributed networks is challenging due to their high dimensional nature. We present an efficient and scalable algorithm ANeTS which takes these graphs and returns a smaller (in nodes and edges) attributed diffusion network

of 'super-nodes' and 'super-edges' as the summarization satisfying the following properties:

P1. Diffusion consistency: Summary graph must have similar diffusion properties as the original one.

P2. Soft Attribute consistency: Summary graph must have similar attribute characteristics as the original one i.e. the nodes in the summary should be as homogeneous as possible.

P3. Scalability & Parallelizability: The summarization must be scalable to the input size and easily parallelizable.

Table 5.1: Feature-based comparison of ANeTS with alternative approaches. ANeTS has all the desirable features. Dotted check mark indicates only some work in that category satisfy the property.

Properties	**ANeTS**	VEGAS [145]	SNAP [153]	Plain graph sum. e.g. [130, 131]	Graph community detection e.g. [169]
Generate a summary	✓	✓	✓	✓	
P1. Diffusion consistency	✓			✓	
P2. Soft Attribute consistency	✓	✓			✓
P3. Scalability & Parallelizability	✓			✓	✓

In our NETWORK-TRAFFIC example, Figs. 5.1 (b) and (d) show the corresponding summary graphs (composed of super-nodes and super-edges) generated by our algorithm ANeTS. We also show some of the correspondance between the summary nodes and regions of the original network via circles. Note that these summaries help highlight the main structure of the original graph. Desirably, the super-nodes are homogeneous with respect to the nodes "type" attribute. Further these summaries also help us to find malicious network requests as anomalies. Unmerged singleton nodes in the summary seem interesting; we realized that they are usually of type 'user-input'. However, there are a number of singleton nodes which are structurally very similar to these user inputs but their attributes type is 'network-request'. For example in Fig. 5.1(b) there are single nodes that are directly connected to the center of the graph with a high weighted edge, just as user-inputs, but their attribute type is different. These are suspicious and turns out that indeed they are malicious requests based on the ground truth [177]. Our summary graphs can correctly detect them and also show their overall connection structure. Note that this also clearly demonstrates the need to take into account *both* structure and attributes to identify these anomalies. Hence a method like ANeTS which preserves both of these properties is required.

The main contributions of our work are:

(I) Problem Formulation: We formulate a novel Attributed Graph Summarization Problem (AGS) to find a smaller diffusion and attribute equivalent summaries.

(II) Efficient Algorithm: We propose an effective sub-quadratic algorithm ANeTS which easily scales to datasets, $\sim 20X$ larger than the datasets our main competitor used [145], while in many cases our baselines do not finish in *14 days* or run out of memory. We also parallelize it to further speed it up.

(III) Extensive Experiments: Our extensive experiments on diverse datasets show the effectiveness and efficiency of our method. Further we also show how to use ANeTS to dramatically speed-up the topic-aware influence maximization task, while maintaining the quality of solutions. Finally, our case studies also show that ANeTS can help make sense of complex attributed networks.

5.2 Problem formulation

We formulate our novel problem on DAW-graphs.

Definition 13 (DAW-graph) $\mathbf{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{W})$ *is a* ***D****irected* ***A****ttributed* ***W****eighted Graph with* n *nodes* ($\mathcal{V}$) *and* m *edges* ($\mathcal{E}$). $\mathcal{F} = \{\mathbf{f}_1, \ldots, \mathbf{f}_n\}$ *are the node attributes* ($\in \mathbb{R}^{d\times 1}$) *and* $\mathcal{W} = \{w_1, \ldots, w_m\}$ *are the edge weights (wlog,* $\in [0, 1]$*).*

Goals. Our goal is to generate a summary DAW-graph satisfying **P1** and **P2**. [130] proposed a plain-graph summarization problem based on merging edges and minimizing a structural distance (between the original graph and the summary) based on influence-based properties. First, we start with their structural distance and merging process. Next, we show how to modify the problem formulation and merging process to preserve both attributes and influence.

Starting Point. *Structural distance:* Motivated by recent work [128], we can maintain the influence based properties by keeping the leading eigenvalue λ of the adjacency matrix of the summary DAW-graph $\mathbf{G^s}$ close to the original $\mathbf{G}$. Hence we can define the *normalized* structural distance between $\mathbf{G^s}$ and $\mathbf{G}$ as follows,

$$\mathcal{D}_{\text{str}}(\mathbf{G}, \mathbf{G^s}) = \frac{|\lambda_{\mathbf{G}} - \lambda_{\mathbf{G^s}}|}{\lambda_{\mathbf{G}}} \tag{5.1}$$

where λ_G is the first eigenvalue of the adjacency matrix of G.

Merge operation: We want to *merge* edges inside each super-node to get the summary. Suppose we merge two connected nodes 'x' and 'y' with no attributes to form a new super-node 'c'. The new edge weights should maintain the *local* influence around the old nodes. Formally,

Definition 14 (Merge operation) *Assume $N^i(c) = N^i(x) \cup N^i(y)$ and $N^o(c) = N^o(x) \cup N^o(y)$ indicate the set of in-neighbors and out-neighbors of a super-node 'c' respectively. Assume $w_{t,c}$ and $w_{c,t}$ denote the weight of the corresponding edges. If the (super-) node-pair (x, y) is now contracted to a new super-node c, and $w_{(x,y)} = \omega_1$ and $w_{(y,x)} = \omega_2$, then the new edges are weighted as:*

$$w_{(t,c)} = \begin{cases} \frac{(1+\omega_1)w_{t,x}}{2} & \forall t \in N^i(x) \setminus N^i(y) \\ \frac{(1+\omega_2)w_{t,y}}{2} & \forall t \in N^i(y) \setminus N^i(x) \\ \frac{(1+\omega_1)w_{t,x}+(1+\omega_2)w_{t,y}}{4} & \forall t \in N^i(x) \cap N^i(y) \end{cases} \tag{5.2}$$

$$w_{(c,t)} = \begin{cases} \frac{(1+\omega_2)w_{x,t}}{2} & \forall t \in N^o(x) \setminus N^i(y) \\ \frac{(1+\omega_1)w_{y,t}}{2} & \forall t \in N^o(y) \setminus N^i(y) \\ \frac{(1+\omega_2)w_{x,t}+(1+\omega_1)w_{y,t}}{4} & \forall t \in N^o(x) \cap N^o(y) \end{cases}$$

Adding Attributes. Next, we extend the above formulation.

Attribute distance: We also want to maintain the attribute-based properties of the original graph. It implies that the nodes inside each super-node of the summary must be homogeneous to be able to accurately represent them by a super-node. Hence, motivated by K-means, we formulate our problem to minimize the distance of node attributes in each super-node to its centroid. Eq. 5.3 formally defines the normalized attribute-based distance between $\mathbf{G}$ and $\mathbf{G^s}$ (here $|v^s|$ indicates the number of nodes in super-node v^s):

$$\mathcal{D}_{\text{att}}(\mathbf{G}, \mathbf{G^s}) = \frac{\sum_{v^s \in \mathcal{V}^s} \sum_{v \in v^s} ||\mathbf{f}_v - \bar{\mathbf{f}}_{v^s}||_2^2}{\sum_{v \in \mathcal{V}} ||\mathbf{f}_v - \bar{\mathbf{f}}||_2^2}, \quad \textit{Where, } \bar{\mathbf{f}}_{v^s} = \frac{\sum_{v \in v^s} \mathbf{f}_v}{|v^s|} \textit{ and, } \quad \bar{\mathbf{f}} = \frac{\sum_{v \in V} \mathbf{f}_v}{n} \tag{5.3}$$

Note, $\mathcal{D}_{\text{att}}$ measures the homogeneity of super-nodes in the summary graph: lower $\mathcal{D}_{\text{att}}$ leads to higher homogeneity.

Merge operation: We also extend the merging process by adding the nodes attributes to it. Motivated by the $\mathcal{D}_{\text{att}}$, the attributes of c should be the centroid of x and y: if $\mathbf{f}_x$ and $\mathbf{f}_y$ are the attribute vectors of x and y, then $\bar{\mathbf{f}}_c = \frac{|x| \cdot \mathbf{f}_x + |y| \cdot \mathbf{f}_y}{|x|+|y|}$.

Attributed Graph Summarization (AGS) problem. We want to find a summary $\mathbf{G^s}$ which is similar to the original $\mathbf{G}$ in structure ($\mathcal{D}_{\text{str}}$) and attributes ($\mathcal{D}_{\text{att}}$). Formally:

***Given*:** a strongly connected DAW-graph $\mathbf{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{W})$, a reduction fraction $0 < \alpha < 1$, and the merge operation.

***Find*:** a summary DAW-graph $\mathbf{G^{s*}}(\mathcal{V}^s, \mathcal{E}^s, \mathcal{F}^s, \mathcal{W}^s)$ by merging edges, with $n^s = (1-\alpha) \cdot n$, and such that:

$$\mathbf{G^{s*}} = \arg\min_{\mathbf{G^s}} \mathcal{D}(\mathbf{G}, \mathbf{G^s}) \tag{5.4}$$

$$\textit{where, } \mathcal{D}(\mathbf{G}, \mathbf{G^s}) = \frac{1}{2} \cdot [\mathcal{D}_{\text{str}}(\mathbf{G}, \mathbf{G^s}) + \mathcal{D}_{\text{att}}(\mathbf{G}, \mathbf{G^s})] \tag{5.5}$$

REMARK: If there are no attributes in $\mathbf{G}$, we only need to minimize the $\mathcal{D}_{\text{str}}$. Hence, AGS problem reduces to the influence-based plain-graph summarization [130]. Also, if $\mathbf{G}$ is a fully connected graph, then all nodes are structurally equivalent. So, nodes will be merged merely based on their attributes and AGS essentially becomes a variant of the classic K-Means problem [82] and only minimizes $\mathcal{D}_{\text{att}}$.

5.3 Our Solution

Both of the special cases of our problem are NP-hard [130, 84]. Nevertheless, there are heuristic methods to solve each of them based on spectral methods or iterative algorithms. However, existing spectral methods for eigenvalue problems rely on matrix perturbation theory, while those for attribute-based data clustering rely on eigenvectors of similarity graphs. It is not clear how to combine these two approaches to make them work on DAW-graphs. In this section, we propose a sub-quadratic algorithm that succesively refines a solution while maintaining feasibility. We first describe the serial algorithm and then propose a careful parallel framework for our solution.

5.3.1 Overview

We present our framework ANeTS (**A**ttributed **Net**work **S**ummarization) to tackle the AGS problem. Our main idea is to combine local matrix perturbations with iterative algorithms for K-means style problems (e.g. Lloyd's algorithm) effectively and efficiently for DAW-graphs. For each $\mathbf{G^s}$ there is a super-nodes assignment vector $\mathbf{\Phi}$ such that $\mathbf{\Phi}[v] \leftarrow v^s$ indicates node v belongs to super-node v^s. Also, $\mathbf{G^s_\Phi}$ is the corresponding summary graph of the assignment $\mathbf{\Phi}$. Our approach updates the super-nodes assignments iteratively while reducing the cost function of its corresponding $\mathbf{G^s}$ until convergence. The framework is as follows:

Step 1 *Initialization*: Start with a super-nodes assignments $\mathbf{\Phi}_0$.

While not converged **do** t++

Step 2 *Find the best assignment*: Perturb $\mathbf{\Phi}_t$, evaluate the quality of the corresponding summary graph according to Eq. 5.5 and select the best assignment.

Step 3 *Update assignments*: Update the super-nodes assignments as $\mathbf{\Phi}_{t+1}$ and the centroids of super-nodes. Also, measure the distance of the summary graph corresponding to the current assignment to the original graph (i.e. $\mathcal{D}(\mathbf{G}, \mathbf{G^s})$ Eq. 5.5).

end

Step 4 *Merging*: Use $\mathbf{\Phi}_{final}$ to extract $\mathcal{V}^s$ and merge nodes in the same super-node and update the edge weights to get $\mathbf{G^s}$.

Even though this framework is fairly straightforward, we face various challenges: (1) Initialization: Since AGS is on graphs, we can not randomly initialize $\mathbf{\Phi}_0$ unlike the clustering algorithms which can. Moreover, we want to find a good starting point that converges to a high-quality solution in few iterations. (2) Running time: Naïvely following the above framework is very expensive, since in each iteration we must compute the largest eigenvalue of all possible summary graphs and update the centroids. following, we describe each step of ANeTS in more detail.

First, we explain Steps 2, 3, and 4 and then how to speed them up. Then, we design an initialization in Step 1.

5.3.2 Step 2: Finding the best assignment

In Step 2, we perturb $\mathbf{\Phi}_t$ to find a better assignment. *First*, we find the best assignment for each node x in the graph (i.e. $v^s_{best}(x)$), while keeping the assignment of all other nodes fixed to $\mathbf{\Phi}_t$—we only move x. The $v^s_{best}(x)$ is the super-node such that moving x into it drops the $\mathcal{D}$ the most (i.e. minimizes the $\mathcal{D}$). Moving x naïvely may lead to some inconsistency in the solution. We will explain the issue later and propose a 'cleanup' method to tackle the problem. Next, we choose the perturbation as the $\mathbf{\Phi}_{t+1}$ which gives the smallest $\mathcal{D}$ among all nodes x.

Finding the next super-node assignment for each node

As mentioned, $v^s_{best}(x)$ is the best new super-node of x which maximizes the drop of $\mathcal{D}$. We define the *drop function* $\Delta(x, B; \mathbf{\Phi})$ as the change in $\mathcal{D}(\mathbf{G}, \mathbf{G^s})$ when x moves from A to B given the super-node assignment $\mathbf{\Phi}$. Formally,

$$\Delta(x, B; \mathbf{\Phi}) = \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[x]\leftarrow A}) - \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[x]\leftarrow B}) \tag{5.6}$$

where $\mathbf{\Phi}[x] \leftarrow B$ indicates node x is assigned to super-node B and $\mathbf{G^s}_{\mathbf{\Phi}[x]\leftarrow B}$ is its corresponding summary graph. Hence, $v^s_{best}(x) = \arg\max_{v^s \in \mathcal{V}^s} \Delta(x, v^s; \mathbf{\Phi}_t)$. Since we want to *localize* the search space to find $v^s_{best}(x)$, we only examine the super-nodes that contain at least one neighbor of x. Furthermore, the $\mathbf{\Phi}_{t+1}$ in Step 2 is the assignment maximizing the largest drop of each node i.e. the next best assignment maximizes $\Delta(x, v^s_{best}(x); \mathbf{\Phi}_t)$ over all x.

Cleanup

We merge edges of $\mathbf{G}$ to get super-nodes. Hence,

Observation 1 (Connectivity of super-nodes) *The corresponding sub-graph of any super-node $v^s \in \mathcal{V}^s$ in the original graph $\mathbf{G}$ is weakly connected.*

If we perturb $\mathbf{\Phi}_t$ naïvely we may end up with a $\mathbf{\Phi}_{t+1}$ with disconnected corresponding super-nodes, violating Obs. 1. Hence, we propose an additional step to 'cleanup' the assignment of nodes in the graph. Formally the validity concept is:

Definition 15 (Valid $\mathcal{V}^s$) *A super-node set $\mathcal{V}^s$ is valid iff*

- ***(Condition 1)*** $|\mathcal{V}^s| = (1-\alpha)\cdot|\mathcal{V}|$
- ***(Condition 2)*** *Each $v^s \in \mathcal{V}^s$ is a weakly connected subgraph.*

$\mathbf{\Phi}_{\text{valid}}$ is the corresponding assignment of a valid $\mathcal{V}^s$. Fig. 5.2(a) shows an example of such a situation: There are two super-nodes A and B in the graph shown by dotted circles. If we just move node x to B, super-node A will be divided into two connected components (Fig. 5.2(b)), which violates Condition 2 above. Also, if we assign each connected component to a new super-node we will violate Condition 1.

How can we resolve this problem? An intuitive solution is to move more nodes to B along with x to keep A connected. As shown in Fig. 5.2(c) moving cc_1 and x to B keeps the super-nodes connected and valid. So, it implies that we must move a group of nodes instead of just one. We also want to get a $\mathcal{V}^s$ with high quality. So, we need to select the best group of nodes to move and have a valid and high-quality $\mathcal{V}^s$.

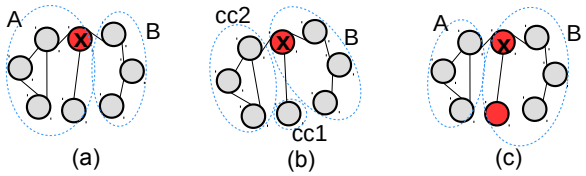


Figure 5.2: Clean up example

To maintain the validity of the super-node set we propose the Clean-up algorithm (see Alg. 9). Assume we want to move node x from super-node A to its best super-node B, which makes A disconnected. We now select additional nodes to move along with x to B. *First*, we extract all connected components $\{cc_i\}_{i=1}^{q}$ generated in A after moving x by using BFS algorithm. *Second*, we move all nodes of A to B except one connected component. It guarantees that both super-nodes A and B remain connected. Now the best connected component cc_i to keep in A is the one which gives the lowest drop if all of its nodes move to B. Formally, $cc^* = \arg\min\ \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[cc_i]\leftarrow B})$, for all $\{cc_i\}_{i=1}^{q}$, where $\mathbf{\Phi}[cc_i] \leftarrow B$ indicates all nodes in connected component cc_i are assigned to B and, $\mathbf{G^s}_{\mathbf{\Phi}[cc_i]\leftarrow B}$ is its corresponding summarized graph. We compute $\mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[cc_i]\leftarrow B})$ by measuring the drop of $\mathcal{D}$ for moving all nodes of cc_i to B. We call it *group drop function* and calculate it as,

$$\Delta(\mathbf{cc}, B; \mathbf{\Phi}) = \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{cc}]\leftarrow A}) - \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{cc}]\leftarrow B}) \tag{5.7}$$

where $\mathbf{cc}$ is a connected subgraph in super-node A. Finally, the valid assignment of x is, $\mathbf{\Phi}^x_{\text{valid}} = \mathbf{\Phi}[A/cc^*] \leftarrow B$. Hence, we move as many nodes as necessary to keep the assignment valid in each iteration.

Algorithm 9 Clean-up algorithm

Require: $\mathbf{\Phi}$, $A \in \mathcal{V}^s$, $B \in \mathcal{V}^s$, $x \in \mathcal{V}$
1: Initialize $\mathbf{\Phi}^x_{\text{valid}} = \mathbf{\Phi}$
2: Move x form super-node A to B
3: find $CC = \{cc_i\}_{i=1}^q$ using BFS on A
4: $cc^* = \underset{cc_i \in \{cc_1, cc_2, \ldots, cc_q\}}{\arg\min} \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi[cc_i]} \leftarrow B})$
5: $\mathbf{\Phi}^x_{\text{valid}} = \mathbf{\Phi}[A/cc^*] \leftarrow B$
6: **return** $\mathbf{\Phi}^x_{\text{valid}}$

Lemma 14 *The time complexity of Clean-up algorithm (Alg. 9) is $O(m_A)$ where m_A is the number of edges of the super-node A.*

In summary, the Clean-up algorithm is the best local approach to keep the validity of the $\mathcal{V}^s$ in *linear time*.

The next best assignment

The next best assignment is the one which minimizes the distance i.e.

$$\mathbf{\Phi}_{t+1} = \underset{\mathbf{\Phi}^x_{\text{valid}}}{\arg\min}\, \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}^x_{\text{valid}}}) \qquad \forall x \in \mathcal{V}. \tag{5.8}$$

5.3.3 Step 3: Updating and Step 4: Merging

After finding $\mathbf{\Phi}_{t+1}$ in Step 2 we update the super-nodes' information. To that end, we update the attribute vectors of all the corresponding super-nodes using Eq. 5.3, and $\mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{G^s}_{\mathbf{\Phi}_{t+1}}})$.

We iterate over Steps 2 and 3 until convergence, and obtain the best assignment $\mathbf{\Phi}_{final}$ that optimizes Eq. 5.4. In Step 4, we *create* the actual summary $\mathbf{G^s}$ by repeatedly applying the merge operation (Def. 14) on edges inside the super-nodes corresponding to $\mathbf{\Phi}_{final}$.

5.3.4 Speeding up

In Step 2, we need to evaluate the change in $\mathcal{D}$ (Eq. 5.6 and 5.7) in the drop functions, while moving a set of nodes from super-node A to B. Consider Eq. 5.6:

$$\begin{aligned}\Delta(\mathbf{x}, B; \mathbf{\Phi}) =& \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow A}) - \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow B}) = 1/2[\big(\mathcal{D}_{\text{str}}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow A}) - \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow B})] \\ &+ [\mathcal{D}_{\text{att}}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow A}) - \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[\mathbf{x}] \leftarrow B})\big)] = 1/2[\Delta\mathcal{D}_{\text{str}}(\mathbf{x}, B; \mathbf{\Phi}) + \Delta\mathcal{D}_{\text{att}}(\mathbf{x}, B; \mathbf{\Phi})]\end{aligned}$$

as $\mathcal{D} = \frac{1}{2}[\mathcal{D}_{\text{str}} + \mathcal{D}_{\text{att}}]$. Our idea is to reduce the time-complexity by approximating $\Delta\mathcal{D}_{\text{str}}$ and $\Delta\mathcal{D}_{\text{att}}$ in Eqs. 5.6 and 5.7 instead of computing them naively. We also give a practical improvement which reduces the constant factors.

Fast computation of structural distance

Computing $\Delta\mathcal{D}_{\text{str}}$ involves computing the change in the leading eigenvalue of possible $\mathbf{G^s}$ and $\mathbf{G}$. Doing this naively will take $O(mnn^s)$ to compute the λ of all possible assignments (assuming we use the Lanczsos method which takes $O(m)$ edges time). To speed this up, we approximate $\Delta\mathcal{D}_{\text{str}}$ instead. Using matrix-perturbation theory, Purohit et al. [130] derive a constant-time first-order approximation $\Delta\widehat{\lambda}_{x,y}$ for $\Delta\lambda_{x,y}$ after merging one edge (x, y) in a non-attributed graph. However $\Delta\mathcal{D}_{\text{str}}$ involves multiple edges, and generalizing $\Delta\widehat{\lambda}_{x,y}$ for merging multiple edges is still costly (see appendix). Hence, we intuitively estimate the change of λ by just aggregating the $\Delta\widehat{\lambda}_{x,y}$ of all merged edges in every super-node. Consider Eq. 5.6 again: as we are only moving node x from A to B, we no longer merge the edges between x and other members of A. Instead we merge edges between x and nodes in B. Using this fact $\Delta\mathcal{D}_{\text{str}}$ for Eq. 5.6 can be estimated as:

$$\Delta\mathcal{D}_{\text{str}}(\mathbf{x}, B; \mathbf{\Phi}) \simeq \frac{\lambda_{\mathbf{G^s}_{\mathbf{\Phi}[x]\leftarrow B}} - \lambda_{\mathbf{G^s}_{\mathbf{\Phi}[x]\leftarrow A}}}{\lambda_{\mathbf{G}}} \simeq \frac{\sum_{y\in B}(\Delta\widehat{\lambda}_{x,y}) - \sum_{z\in A}(\Delta\widehat{\lambda}_{x,z})}{\lambda_{\mathbf{G}}} \tag{5.9}$$

We can similarly estimate $\Delta\mathcal{D}_{\text{str}}$ for Eq. 5.7 as well. See the complete derivation in appendix. This estimation reduces the time complexity of evaluating $\Delta\mathcal{D}_{\text{str}}$ from $O(mnn^s)$ to $O(m)$.

Fast computation of attribute distance

We need to compute $\Delta\mathcal{D}_{\text{att}}$ as well from above. For example, in Eq. 5.6 it means that we need to reevaluate the $\overline{\mathbf{f}}_B$ for each possible move of x to a super-node B which costs $O(dn^2)$ in each iteration. Therefore, to speed-up ANeTS, we disregard the change of centroids while computing $\mathcal{D}_{\text{att}}$. It is reasonable to assume that the centroids themselves change smoothly between the moves. Hence, we approximate as follows:

$$\Delta\mathcal{D}_{\text{att}}(\mathbf{x}, B; \mathbf{\Phi}) \simeq \frac{||x - \overline{\mathbf{f}}_A||_2^2 - ||x - \overline{\mathbf{f}}_B||_2^2}{\sum_{v\in\mathcal{V}} ||\mathbf{f}_v - \overline{\mathbf{f}}||_2^2} \tag{5.10}$$

In the same way, we can estimate $\Delta\mathcal{D}_{\text{att}}$ for the group drop function in Eq. 5.7 too. See the complete derivation in appendix. This estimation reduces the complexity of evaluating $\Delta\mathcal{D}_{\text{att}}$ from $O(dn^2)$ to $O(dn)$.

A practical improvement

Note that $\mathbf{\Phi}_t$ and $\mathbf{\Phi}_{t+1}$ are differ in only two super-nodes (we move a group of nodes from one super-node to another one). So, in Step 3, we only need to update the centroids of the two updated super-nodes. Also, we can divide the nodes of $\mathbf{G}$ into two groups: (group 1) Nodes in the updated super-nodes and (group 2) the rest. In each iteration, we need to recalculate the drop function of nodes in group 1 for all possible super-nodes. However, we need to recalculate the drop function of nodes in group 2 for the two updated super-nodes only. Although this does not change the time-complexity, it does help considerably in avoiding unnecessary calculations in ANeTS.

5.3.5 Step 1: Initialization

One issue still remains: we need to initialize the assignments at the beginning of the ANeTS. We can not randomly assign nodes into super-nodes it may not give a feasible solution. Our intuition is to gradually merge *unimportant* edges with *similar* (attributed) end-nodes to end up with a high-quality starting point as follows: (1) Compute a 'score' for each edge in $\mathbf{G}$. It estimates the $\mathcal{D}_{\text{str}} + \mathcal{D}_{\text{att}}$ of merging the end nodes of each edge in $\mathbf{G}$. (2) Gradually assign end nodes of edges with the lowest score to the same super-node until we get the target number of super-nodes. We compute the score of an edge $(x, y) \in \mathcal{E}$ as, $score(x, y) = \frac{\Delta\widehat{\lambda}_{x,y}}{\lambda_{\mathbf{G}}} + \frac{||\bar{\mathbf{f}}_x - \bar{\mathbf{f}}_y||_2^2}{\sum_{v\in\mathcal{V}} ||\mathbf{f}_v - \bar{\mathbf{f}}||_2^2}$. Note that although our initialization is very intuitive, it does not directly minimize Eq. 5.5. So, we need to follow our steps of ANeTS to improve the results for the AGS problem.

5.3.6 The complete (serial) algorithm

Alg. 10 is the pseudo-code of the complete serial version of ANeTS algorithm. Also, we have the following lemmas,

Lemma 15 ANeTS *converges* *in finite number of iterations and reduces $\mathcal{D}$ in each iteration.*

Lemma 16 *Time complexity of serial* ANeTS *is sub-quadratic $O(m \cdot \log m + n_{itr} \cdot (\mathfrak{n}_x \mathfrak{m}_x + n^s \cdot n \cdot d))$. Also, the memory complexity of ANeTS is linear $O(n \cdot d + m)$. The n, d, and n_{itr} are the number of nodes, attributes and iterations respectively. E_o is the number of edges between nodes in different processors and n_x and m_x is the number of nodes and edges of the largest super-node.*

Algorithm 10 The ANeTS framework

Require: DAW-graph $\mathbf{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{W})$, α
1: //Step 1: Initializing super-nodes
2: $t \leftarrow 0$, $\mathbf{\Phi}_0 =\leftarrow$ INITIALIZATION$(\mathbf{G}, \alpha)$ (Sec. 5.3.5)
3: **while not** converged **do**
4: $t \leftarrow t + 1$
5: //Step 2: Find the next best super-node assignment $\mathbf{\Phi}_{t+1}$ (Sec. 5.3.2)
6: **for** $x \in \mathcal{V}$ **do**
7: $\mathbf{\Phi}^x_{\text{valid}} = \textsf{Clean-up}(\mathbf{G}, v^s_t(x), v^s_{best}(x), x)$
8: Compute $\mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}^x_{\text{valid}}})$
9: $\mathbf{\Phi}_{t+1} = \arg\min_{\mathbf{\Phi}^x_{\text{valid}}} \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}^x_{\text{valid}}})$ (Eq. 5.8)
10: //Step 3: Update super-nodes (Sec. 5.3.3)
11: Update $\overline{\mathbf{f}}_{v^s}$ for each $v^s \in \mathcal{V}^s$ corresponded to $\mathbf{\Phi}_{t+1}$ Update $\mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}_{t+1}})$ (Eq.
12: 5.5).
//Step 4: Merging (Sec. 5.3.3) Merge nodes of each super-node to get $\mathcal{E}^s$, and $\mathcal{W}^s$
13: **return**DAW-graph $\mathbf{G^s}(\mathcal{V}^s, \mathcal{E}^s, \mathcal{F}^s, \mathcal{W}^s)$

5.3.7 Scaling-up ANeTS: Parallelization

Although the time complexity of serial ANeTS is better than state-of-the-art competitors, it does not scale up to graphs with more than $20k$ nodes. The bottleneck is in finding $\mathbf{\Phi}_{t+1}$ in each iteration (i.e. Step 2: lines 6 to 8) and updating the attributes of super-nodes (line 12). This motivates us to propose an efficient parallel framework for ANeTS to make it faster. However, designing it is not straightforward due to the complexity of the Clean-up step. For example, a first-cut approach would simply distribute the nodes to processors. Each processor finds the best super-nodes, and does the clean-up separately. The main drawback is that, as real networks are skewed, there would be unbalanced workloads due to different super-node sizes.

(3) Our approach. We run Clean-up in parallel and make sure the processors have balanced workload. Instead of having $\mathbf{G}$ as global information, each processor accesses a part of $\mathbf{G}$ associated with its assigned nodes. We do this process for all nodes in parallel–see Alg. 11. Assume $N_A = \{v_{A_1}, v_{A_2}, \ldots, v_{A_h}\}$ is the set of nodes in super-node A after moving x. In the serial algorithm (i.e. $p = 1$), we run BFS to detect CCs in the subgraph of A. When $p > 1$, nodes in A can be in different processors. Hence, we propose the following procedure to find CCs in parallel: Assume N^j_A is the subset of nodes in A which are in processor j. (I) Extract CCs of each N^j_A using BFS. So for super-node A we will have p sets of CCs computed in each processor. (II) Next, merge these p sets to get CCs of A. In this step, merge two CCs into one iff there is at least one edge between them. Continue merging CCs until we can not merge more. Finally, we compute the drop of distance for each component by accumulating the drop of the nodes inside them.

Algorithm 11 Parallel Clean-up

Require: $\mathbf{\Phi}$, a set of triples $Tr = (A \in \mathcal{V}^s,\ B \in \mathcal{V}^s,\ x \in \mathcal{V})$, processors node assignments

1: Initialize $\mathbf{\Phi}^x_{\text{valid}} = \mathbf{\Phi}$ for each triple in Tr in parallel
2: Move each x in Tr form its super-node A to B in parallel In each processor j find CC_j of N^j_A associated with each triple in parallel
3: //Merge connected components
4: **for** each triple in Tr **and** $(a, b) \in \mathcal{E}$ where a and b are in different processors i and j **do**
5: merge CC_i and CC_j
6: **for** each triple in Tr **do**
7: $cc^* = \underset{cc_i \in \{cc_1, cc_2, \ldots, cc_n\}}{\arg\min} \mathcal{D}(\mathbf{G}, \mathbf{G^s}_{\mathbf{\Phi}[cc_i] \leftarrow B})$
8: $\mathbf{\Phi}^x_{\text{valid}} = \mathbf{\Phi}[A/cc^*] \leftarrow B$
9: **return** Set of $\mathbf{\Phi}^x_{\text{valid}}$

Lemma 17 ***Time complexity of parallel*** **ANeTS** *is* $O(m \cdot \log m + n_{itr} \cdot \left(\frac{\mathfrak{n}_x \mathfrak{m}_x + n^s \cdot n \cdot d}{p} + E_o + p \cdot d\right)$*;* p *is #processors and* E_o *is #edges with end-points in different processors.*

5.4 Sample Application: Topic-aware Diffusion

Our summary $\mathbf{G^s}$ can be used to speed-up other applications while maintaining performance. Next we show how to use ANeTS to scale the topic-aware influence maximization (TIM) problem [35]. This problem is useful in many applications such as predicting activity or product/opinion adoption, in various kinds of datasets such as tweets, DBLP, etc.

Problem setting: Consider a social graph $\text{SG}(\text{V}_{\text{SG}}, \text{E}_{\text{SG}}, \mathbf{P})$, where V_{SG} is a set of people and E_{SG} with directed influence relations (edges) between people. $\mathbf{P}(u, v) = [\text{p}_1, \ldots, \text{p}_T]$ is the weight vector of each edge where p_i indicates the probability that node u can influence v in topic i (T is the number of topics). The Topic-aware Influence Propagation (TIP) model [23] is a cascade model used to model influence-propagation of a topic-aware content Q along the edges of a social graph SG. The content $Q = [q_1, \ldots, q_T]$ is represented by the topic distribution of the content where q_i is the probability of topic 'i' in content Q. A vertex $v \in \text{V}_{\text{SG}}$ is active if it has been influenced by (or adopted) Q and inactive otherwise. There is an initial set of 'seed' S active nodes. Each active node u has a single chance to activate each of its currently inactive neighbor v independently with probability $\mathbf{P}(u, v) \cdot Q'$. This cascade process continues until no more activations are possible. The final number of active nodes is the 'influence spread' of seed set S given the content Q. Formally the TIM problem is:

Given a social graph $\text{SG}(\text{V}_{\text{SG}}, \text{E}_{\text{SG}}, \mathbf{P})$, its TIP model, a content $Q = [q_1, \ldots, q_T]$, and a budget 'k',

Find $S^*_{TIM} = \underset{S_{TIM} \subseteq \mathrm{V_{SG}}, |S_{\mathsf{TIM}}|=k}{\arg\max} \sigma_{TIP}(S_{\mathsf{TIM}}|Q)$.

Our approach. The TIM problem is NP-hard, and current algorithms [35] are slow (e.g. they do not finish for graphs larger than $50k$ in ten days). We use the state-of-the-art algorithm TIM [35], and propose ANeTS-based AnT to scale it up.

Firstly, SG is not a DAW-graph. So we first convert it to a DAW-graph $\mathbf{G}$ by mapping the edge attributes of SG to node attributes. Next, we summarize $\mathbf{G}$ and map it back to a smaller social graph SG^s of super-nodes and super-edges. Then, we run the TIM algorithm on (the much smaller) SG^s to get a seed-set of super-nodes. Finally, we just randomly select a node in each super-node of selected seeds as the final solution 'S'. Intuitively, the selected seed set using this approach is an accurate solution because we merge nodes into super-nodes with similar topic influence probabilities to their neighbors in the original graph. Hence randomly picking any node within the super-node as a seed should give the same expected influence spread (i.e. $\sigma_{TIP}(S^*_{tim}|Q) \simeq \sigma_{TIP}(S^*_{\mathsf{AnT}}|Q)$). Hence our framework AnT is:

(1) Map weights to attributes. Convert the social graph $\mathrm{SG}(\mathrm{V_{SG}}, \mathrm{E_{SG}}, \mathbf{P})$ to a DAW-graph $\mathbf{G}(\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{W})$ with same number of nodes and edges using the following simple scheme: (I) For each node $u \in \mathcal{V}$ we have a T dimensional attribute vector $\mathbf{f}_u \in \mathcal{F}$. This $\mathbf{f}_u$ is the average value of all edges starting from it corresponding node $u_{\mathrm{SG}} \in \mathrm{SG}$. The mapping captures the average influence behavior of a node $u_{\mathrm{SG}} \in \mathrm{SG}$ as the attribute of u in $\mathbf{G}$. (II) Assign an arbitrary uniform probability to all edges of DAW-graph as $\mathcal{W}$.

(2) Get a summary. Summarize $\mathbf{G}$ to $\mathbf{G^s}$ using ANeTS.

(3) Map attributes to weights. Since $\mathbf{G^s}$ is a DAW-graph we convert it back to a social graph $\mathrm{SG^s}(\mathrm{V^s_{SG}}, \mathrm{E^s_{SG}}, \mathbf{P^s})$. In $\mathbf{G^s}$, for each super node we have T attributes. The edge weight from super-node $u^s_{\mathrm{SG}} \in \mathrm{V^s_{SG}}$ to $v^s_{\mathrm{SG}} \in \mathrm{V^s_{SG}}$ is simply the attribute of its corresponding node $u^s \in \mathbf{G^s}$ (i.e. $\mathbf{P}(u^s_{\mathrm{SG}}, v^s_{\mathrm{SG}}) = \bar{\mathbf{f}}_{u^s}$). This mapping maintains the average influence behavior of super-nodes.

(4) Solve. Run the state-of-the-art TIM algorithm [35] on the smaller graph SG^s to get k seed super-nodes.

(5) Pull back. To get the seed nodes of the original graph we randomly select a node inside the seed super-nodes $s_1, \ldots, s_k$.

5.5 Empirical Study

We design various experiments to evaluate ANeTS in this section. We implemented ANeTS in Python. Our experiments were conducted on a 4 Xeon E7-4850 CPU with $512GB$ of $1066Mhz$ main memory and we will release our code for research purposes.

Datasets. We collected a number of datasets with different scales and domains such as social

networks, system network traffic, movie rating, co-authorship and email communication network for our experiments. See Table 5.2 for details. Note, the size of the attributed graphs is $O(\#Nodes \times \#Attr + \#Edges)$ in contrast with the size of plain-graphs with no attributes which is only $O(\#\ Edges)$.

Table 5.2: Details about our 8 datasets.

Dataset	#Nodes	#Edges	#Attr	Size = #Nodes × #Attr + #Edges
1. Disney	124	335	28	3,807
2. Memetracker	960	5,000	76	77,960
3. Citeseer	2,111	3,669	3,703	7,820,702
4. Facebook	4,039	88,234	1,402	5,750,912
5. Google+	3,820	280,742	6	303,662
6. Enron	13,533	176,967	20	447,627
7. PubMed	19,717	44,336	500	9,902,836
8. YouTube	77,381	367,151	300	23,581,451

(1) Disney [124]: Nodes are movies and links indicated a co-purchase relation. Attributes are price, average rating, etc.

(2) Memetracker[1] Nodes are blog/website and links means who-copies-from whom. Attributes are the type (blog/website), the number of posts and the rest indicate the time node adopts a popular meme.

(3) Citeseer/ PubMed [134, 142]: Nodes are publications, and edges indicate citation relationships. Attributes in Citeseer are binary and show stemmed words from the publications. In PubMed attributes are a TF/IDF weighted word vector of each publication.

(4) Facebook/ Google+/ YouTube[2] [124]: Nodes are users and links indicate the friendship. Attributes are extracted from users profiles.

(5) Enron [67]: Nodes are email addresses and edges shows email transmissions in Enron Inc.. Attributes are the average number of recipients, average content length, etc.

Baselines. We compare ANeTS to the most related approaches, including a matrix decomposition based algorithm for summarizing attributed graphs (VEGAS [145]), influence-based summarization algorithm for plain graphs (CoarseNET [130]), attributed-graph clustering/community algorithms (BAGC [169], CODICIL [134]), and a min-cut partitioning approach (METIS [86]).

Note that BAGC, CODICIL, and METIS only find node clusters/communities: to adapt them to find a summary graph, we merge nodes in the same cluster/community using Def. 14.

[1] snap.stanford.edu
[2] snap.stanford.edu

Also, note that the datasets we used are much larger than the datasets that our main competitor VEGAS used (i.e. $\sim 20X$ larger) and similar to the other graph community detection algorithms such as BAGC.

5.5.1 Performance of ANeTS

Effectiveness: We calculate $\mathcal{D} = \frac{1}{2}[\mathcal{D}_{\text{str}} + \mathcal{D}_{\text{att}}]$ to show the performance of different algorithms. On average ANeTS took 200 itrs. to converge and reduces the initial $\mathcal{D}$ by 68% and Clean-up triggered around half of the times. It shows that even with our intuitive initialization, multiple iterations and Clean-up are necessary to find a high-quality summary Note in many cases VEGAS, BAGC and CODICIL do not even finish after fourteen days! or run out of memory. It shows, AGS is challenging and clearly ANeTS can fill the gap between attributed and plain graph summarization.

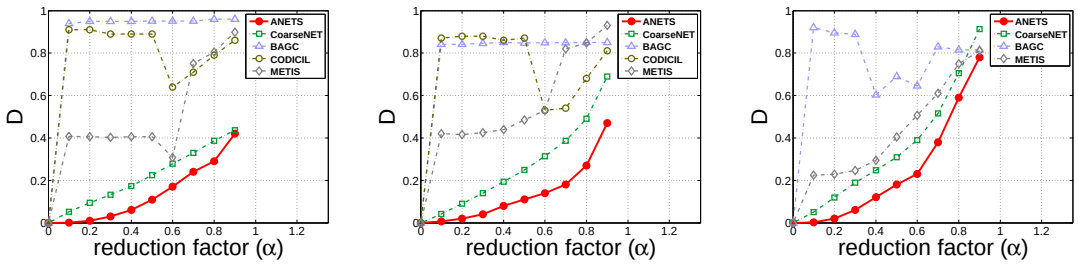
Minimizing $\mathcal{D}$. We show the results of some datasets in Tab. 5.3. It shows the $\mathcal{D}$, $\mathcal{D}_{\text{str}}$, and $\mathcal{D}_{\text{att}}$ with $\alpha = 0.7$ and Figs. 5.3 shows the results with different α values (since VEGAS is very slow, we could not run it for multiple α values). ANeTS has constantly the best performance for all datasets getting a 44% on avg. improvement which is significant considering the challenges of the problem. ANeTS reduces the $\mathcal{D}$ as much as the graph allows: e.g. GOOGLE+ (Fig. 5.3b) is a dense graph with avg. deg. 146.9 and $\mathcal{D}$ of ANeTS is much lower than the baselines even with high reduction factors. On the other hand, ENRON (Fig. 5.3c) is a sparser graph with avg. deg. 26.1. So, performance degrades mainly with high reduction factors similar to baselines.

Balanced summary and homogeneity. Tab. 5.3 clearly shows that unlike other approaches, ANeTS minimizes the $\mathcal{D}_{\text{str}}$ along with $\mathcal{D}_{\text{att}}$. For example, CoarseNET only optimizes $\mathcal{D}_{\text{str}}$, and usually produces a much larger $\mathcal{D}_{\text{att}}$ than ANeTS, hence giving less meaningful summaries. Note, the very sparse structure of PUBMED (Avg. deg. = 4.4) results in a similar score for ANeTS and CoarseNET. However, ANeTS gives a more balanced summary (i.e. $\mathcal{D}_{\text{str}}$ and $\mathcal{D}_{\text{att}}$ are evenly minimized). Similarly, CODICIL has good performance on $\mathcal{D}_{\text{att}}$ while does not preserve the diffusive property (a larger $\mathcal{D}_{\text{str}}$ value). The low $\mathcal{D}_{\text{att}}$ of ANeTS implies the super-nodes of $\mathbf{G^s}$ are homogeneous. To examine it we computed the entropy of nodes attributes in the same super-node: ANeTS gives 7 times lower entropy than the other baselines – detailed results omitted due to lack of space.

Scalability: To examine the running time of ANeTS we extract subgraphs with different sizes from YOUTUBE ($\sim 20M$ dataset size), and run it. As expected from the complexity, we observed that ANeTS scales near-linear w.r.t the size of the DAW-graph $\mathbf{G}$ (See Fig. 5.4(g)). Also we got $\sim 9X$ speedup after parallelizing ANeTS using 10 processors (See Fig. 5.4(h)).

Table 5.3: The results of ANeTS and baselines with $\alpha = 0.7$. Bold numbers are winners, and ANeTS performs best in all datasets. '-' means the method does not finish after 14 days. '×' means the method runs out of memory.

Data		ANeTS	VEGAS	CoarseNET	BAGC	CODICIL	METIS
DISNEY	$\mathcal{D}$	**0.30**	0.49	0.34	0.82	0.59	0.69
	$\mathcal{D}_{\text{str}}$	0.06	0.38	0.02	0.71	0.55	0.76
	$\mathcal{D}_{\text{att}}$	0.54	0.60	0.66	0.93	0.63	0.62
MEMETRACKER	$\mathcal{D}$	**0.34**	0.69	0.41	0.64	0.39	0.67
	$\mathcal{D}_{\text{str}}$	0.22	0.86	0.19	0.70	0.23	0.65
	$\mathcal{D}_{\text{att}}$	0.46	0.52	0.63	0.58	0.57	0.70
CITESEER	$\mathcal{D}$	**0.32**	0.78	0.34	0.56	0.43	0.70
	$\mathcal{D}_{\text{str}}$	0.00	0.98	0.00	0.25	0.23	0.79
	$\mathcal{D}_{\text{att}}$	0.64	0.58	0.70	0.63	0.63	0.62
FACEBOOK	$\mathcal{D}$	**0.24**	0.48	0.33	0.95	0.71	0.75
	$\mathcal{D}_{\text{str}}$	0.00	0.61	0.00	0.97	0.80	0.87
	$\mathcal{D}_{\text{att}}$	0.48	0.35	0.66	0.93	0.62	0.63
GOOGLE+	$\mathcal{D}$	**0.18**	0.66	0.38	0.84	0.54	0.81
	$\mathcal{D}_{\text{str}}$	0.13	0.99	0.07	0.99	0.58	0.85
	$\mathcal{D}_{\text{att}}$	0.23	0.33	0.70	0.69	0.50	0.78
ENRON	$\mathcal{D}$	**0.38**	-	0.51	0.83	×	0.60
	$\mathcal{D}_{\text{str}}$	0.36	-	0.26	0.97	×	0.67
	$\mathcal{D}_{\text{att}}$	0.40	-	0.76	0.69	×	0.54
PUBMED	$\mathcal{D}$	**0.34**	-	**0.34**	0.80	0.43	0.52
	$\mathcal{D}_{\text{str}}$	0.04	-	0.00	0.75	0.23	0.34
	$\mathcal{D}_{\text{att}}$	0.64	-	0.69	0.86	0.63	0.70
YOUTUBE	$\mathcal{D}$	**0.26**	-	0.34	×	×	0.68
	$\mathcal{D}_{\text{str}}$	0.03	-	0.02	×	×	0.69
	$\mathcal{D}_{\text{att}}$	0.47	-	0.66	×	×	0.68



(a) FACEBOOK (b) GOOGLE+ (c) ENRON

Figure 5.3: The $\mathcal{D}(\mathbf{G}, \mathbf{G}^{\mathbf{s}})$ achieved by ANeTS and baselines with different α. ANeTS is best with any α in all datasets.

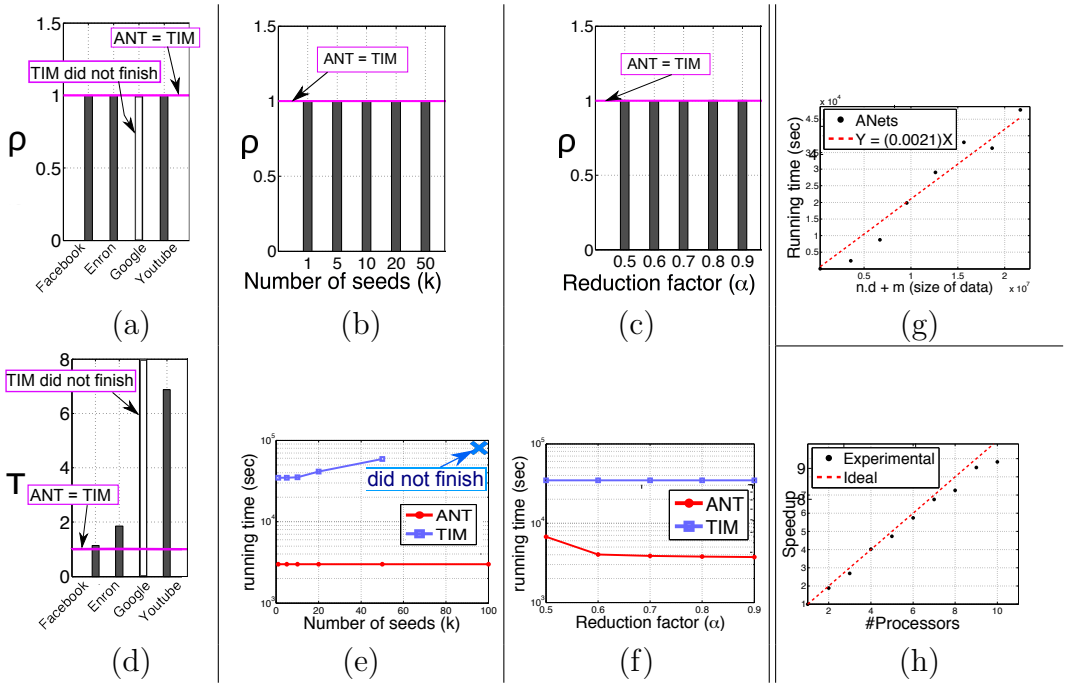
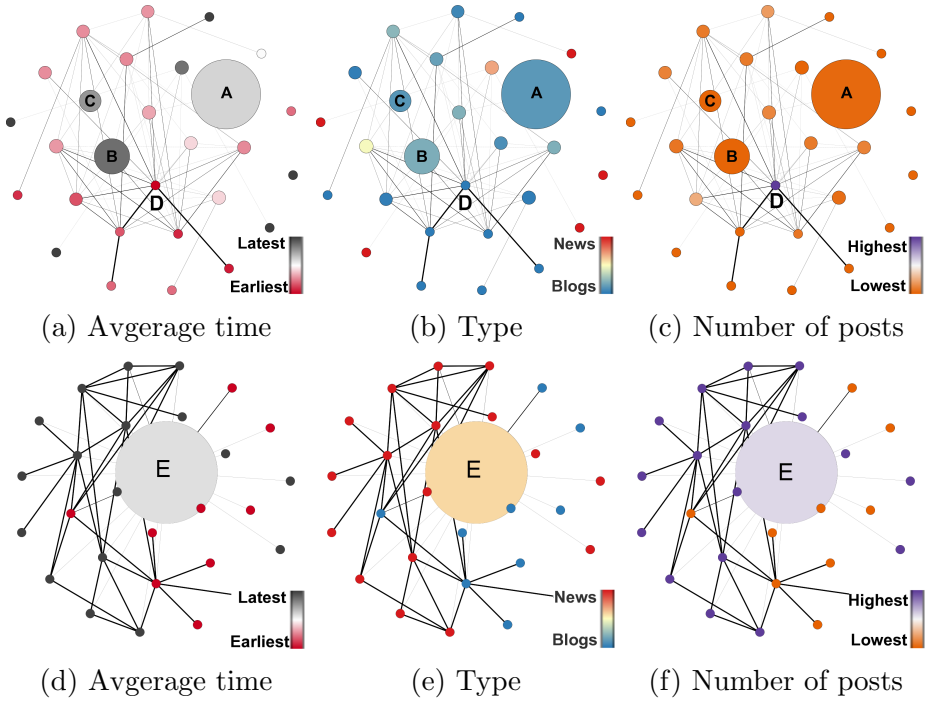


Figure 5.4: (a-f) Performance of AnT. (a) expected influence spread ratio ρ and (d) speedup. (b) and (e) ρ and run time of AnT and TIM vs k. (c) and (f) ρ and run time of AnT and TIM vs α. AnT is significantly faster than TIM, giving similar results. (g-h) Efficiency of ANeTS. (g) running time of ANeTS and, (h) parallelization speed-up. ANeTS is *near-linear* in running time and in parallelization speed-up as well.

5.5.2 Application 1: Topic-aware Diffusion

Here we apply ANeTS to speed-up the original TIM task (see the AnT framework in Sec. 5.4). Our experiments show that we are able to achieve up to $20X$ speedup on large networks while maintaining the quality of solutions. We can use any off-the-shelf algorithm to solve the TIM problem on original and summarized graphs. We used state-of-the-art online TIM [35] algorithm. Finally, as the topic aware propagation probabilities are not available for our datasets, following literature [35], we generate a random probability vectors for each edge and we assume there are 10 topics (i.e. $T = 10$) in each dataset.

Effectiveness of AnT: We examine the expected influence spread of AnT and TIM across various datasets, number of seed nodes and reduction factors (i.e. α). Fig. 5.4(a) shows the expected influence spread ratio $\rho = \frac{\sigma_{\mathsf{AnT}}(S|Q)}{\sigma_{\mathsf{TIM}}(S|Q)}$ with $k = 5$ and $\alpha = 0.7$. Fig 5.4(b) and Fig. 5.4(c) show the influence spread ratio ρ with various number of seed nodes and reduction factors. In *all* the experiments, the influence spread of the AnT is within 1% of influence spread of TIM approach. In some cases such as ENRON, we even perform slightly better. Note that TIM did not finish on GOOGLE+ after **10 days** (AnT finished in around *an hour*). Also, Fig. 5.4(c) indicates that even with extremely compact summary (i.e. $\alpha = 0.9$) AnT



(a) Avgerage time (b) Type (c) Number of posts

(d) Avgerage time (e) Type (f) Number of posts

Figure 5.5: MEMETRACKER—Node colors in each graph represent an attribute of the super-nodes. ANeTS (top) and CoarseNET (bottom) summary: (a), (d) The average time of infection, (b), (e) the type of the websites, and (c), (f) # posts related to the memes. Size of super-nodes $\propto$ the # merged nodes (Best viewed in color).

works as well as TIM.

Speed-ups by AnT: Here we show the running time and speed-up of using AnT versus TIM. We define the speed-up as $\tau = \frac{\textit{Running time of TIM}}{\textit{Running time of AnT}}$.

W.R.T. graph size: Fig. 5.4(d) shows that τ increases as the size of graph grows. E.g. in YOUTUBE we get $7X$ speedup while the quality of the results is same as the plain TIM algorithm.

W.R.T. number of seeds (k): Fig. 5.4(e) shows the results for experiments on YOUTUBE with $\alpha = 0.7$ and number of seeds varies from 1 to 100. The running time of AnT increases only slightly as k increases, while TIM scales very poorly: with $k = 100$ TIM does not even finish within *ten days*.

W.R.T. (α): Fig. 5.4(f) shows that the running time of AnT decreases with more smaller summaries (i.e. higher α).

5.5.3 Application 2: Making sense of graphs and anomaly detection

Here we show how to use ANeTS to understand and explore complex networks and detect anomalies (NETWORK-TRAFFIC, MEMETRACKER and PORTLAND).

Network-traffic: Fig. 5.1 shows two examples of network-traffic graphs. As explained earlier, it is hard to recognize malicious requests by simply inspecting the original graphs. However, our summaries highlight these anomalous nodes, matching the ground-truth [177] and also showing that they are structurally similar to user inputs–Fig 5.1(b) and (d) (which makes them hard to find in the first place). Hence our summary helps provide structural evidence for system and network assurance and is useful for human experts cognition, and decision making in cyber security.

Memetracker: We summarize the network with $\alpha = 0.97$. MEMETRACKER is a blog/News network, and the edges indicate who copied from whom. Attributes are the type of nodes (i.e. blog or news website), the number of posts about memes, and the first time that the node posted anything about memes. The Fig. 5.5 shows the $\mathbf{G^s}$ of the MEMETRACKER dataset. Node colors in each graph represent different attributes of the super-nodes.

ANeTS should ensure that nodes in a super-node are similar w.r.t the influence structure and attributes. Super-node 'A' contains mainstream news media sites such as CNN, BBC, Guardian, etc. It shows that these websites are structurally similar. Also, their behavior in reporting news (i.e. time of report and # posts) is similar. Fig. 5.5a shows that mainstream news media sites usually report memes earlier than their connected blogs/news sites. Also the high degree centrality (= 16) of 'A' in the summary graph demonstrates that other nodes copy content from them.

'B' and 'C' are immediate neighbors of super-node 'A'. Super-node 'C' mostly contains news blogs and 'B' has websites and blogs related to sports and entertainment such as ESPN. Fig. 5.5a shows the 'B' and 'C' usually get infected through mainstream news media. According to Fig. 5.5a and 5.5b blogs publish memes very quickly but not many websites follow them and report the news later. Their low degree (~ 2) and low PageRank (~ 0.02) show that many websites specially mainstream news websites do not follow them.

Our summary can also help in anomaly detection. See the *single* purple node 'D'. It is a news website which has the highest number of posts and earliest time of reporting. It is suspicious as it was not merged with the other group of mainstream websites by ANeTS. Turns out that this website belongs to "gopusa.com" which is a spam news website and not affiliated to the Republican National Committee. Our summary clearly shows it as an anomaly. In contrast, CoarseNET does not highlight any of the above patterns as it preserves only the structural properties.

Portland: Fig. 5.6 shows the PORTLAND summary. It is a people contact network in Portland area: nodes are people and links are their interactions. Attributes are age, location

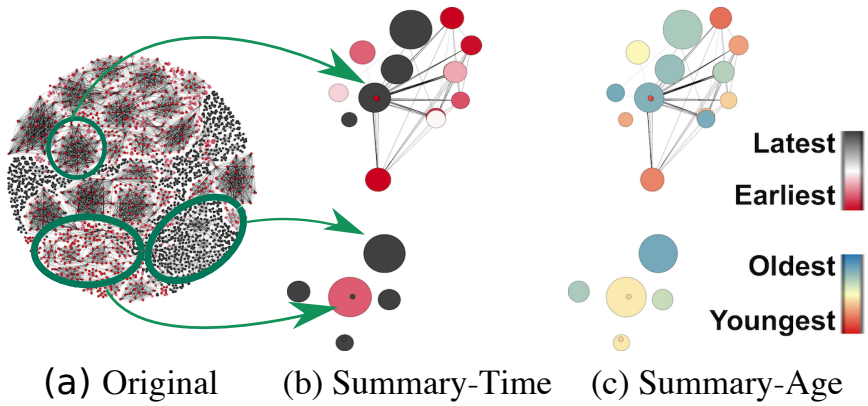


Figure 5.6: PORTLAND. Node colors represent atts. (a) Original Graph. (b) Avg. time of infection. (c) Avg. age. Size of super-nodes $\propto$ # merged nodes (Best viewed in color).

and time of infection of people. We ran a *Targeted* flu-infection scenarios on the dataset [18]: The infection starts at a location, and older individuals have higher probability of getting infected through their young neighbors. Colors represent attributes and the location of nodes represents their actual lat-long.

Our summary gives a better insight about how the disease propagates. For example, 'the disease starts from a location infecting young people, and gradually moves to other areas of the network infecting the elderly' (See Fig. 5.6). These kinds of insights help epidemiologists in understanding and eventually controlling contagious diseases better. Such a pattern cannot be easily observed on the large original network due to the complex interplay between the attributes and network structure. However, our summary can easily highlight them.

5.6 Discussion and Conclusions

We proposed ANeTS, a new unsupervised, scalable and parallelizable algorithm which gets high-quality summaries of attributed influence graphs, in *near-linear* time in practice (in contrast to non-trivial competitors). The summary can be used for many applications: e.g. it speeds-up the TIM task and detects malicious nodes in a network; and it intuitively highlights structurally and attributed homogeneous regions in the network—helping in sense-making of complex network datasets.

Effect of attributes. Most real-world graphs are skewed; so typically many nodes will not be structurally important. Hence plain-graph methods such as CoarseNET will likely give unbalanced summary graphs (like in Fig. 5.5). However, as we showed in the experiments, considering attributes makes a significant difference and gives more semantically meaningful and balanced summaries.

Alternative approaches. For this challenging task, ANeTS improves the state-of-the-art in multiple ways: (a) it easily outperforms competitors for the AGS problem giving nodes with higher homogeneity; (b) enables summarization of $\sim 20X$ larger DAW-graphs (current methods like VEGAS do not even finish in 14 days); and (c) it helps scale existing algorithms for other tasks like TIM $\sim 20X$ times.

Flexibility. ANeTS naturally generalizes both plain-graph summarization and attributed data-clustering in a flexible framework. For example, we can replace the euclidean distance with any distance metric suitable for K-means style algorithms. Extending our method to a streaming setting (as it is iterative already), and to incorporate overlapping K-Means to give more complex summaries will be interesting.

Part II

Learning-based approaches

5.7 Overview

In the second part of the thesis, we develop learning approaches to summarize graphs for given tasks. We start with univariate graphs and develop a unified learning-based-framework. We summarize a graph for any graph optimization problem (Chapter 6). As the next step toward our goal to summarize multivariate networks, we design feedback based summarization which integrates the goals of the given task with human feedback to generate meaningful summaries for multivariate networks (Chapter 7).

In Chapter 6, we generalize over multiple threads of prior work and propose a novel Task-Based Network Summarization (TBNS) problem to automatically learn how to generate task-specific network summaries. We proposed an effective method NetGist by leveraging the deep Q-learning framework. As shown by our experiments on both real and synthetic data, NetGist is able to learn meaningful summaries for various tasks like *Influence maximization* and *Community detection* even when we reduce the network size by 90%. Further, we show how we can use our framework to help develop practical new algorithms for challenging open problems (like GUIDED-TBNS) and also sensemaking via visualization by proposing Guided-NetGist. Also, since Guided-NetGist is entirely automatic and reusable, it can generate summaries for multiple networks at the same time and use the learned model in other similar graphs. Hence, it has a significant impact in speeding up creating high-quality visualization for multiple document datasets.

In Chapter 7, we go further and explored the problem of learning interactive network summarizations of multivariate networks which helps in learning generalizable visualization models for text analysis. We proposed a novel and effective algorithm NetReAct which leverages a feedback-based reinforcement learning approach to incorporate human input as well as the visualization task to produce high-quality summaries and visualizations. Our extensive experiments show that NetReAct is able to summarize and visualize a document network meaningfully to reveal hidden stories in document corpora and connect the dots between different documents.

Chapter 6

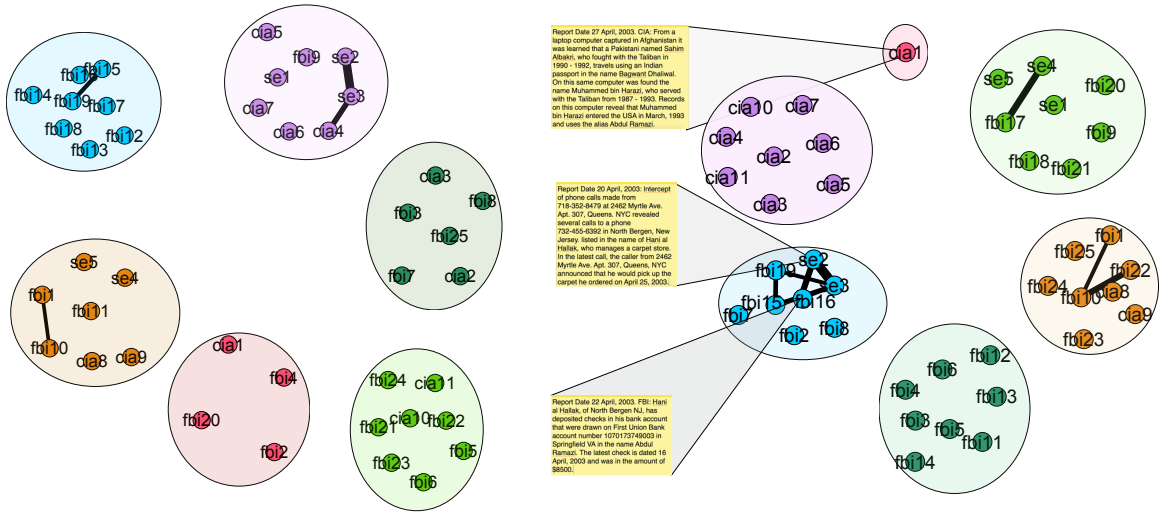
Learning to generate network summaries

As the first step toward our goal to develop learning approaches to summarize graphs, we investigate univariate networks where nodes have no label or attribute. As mentioned earlier such networks are widespread, and hence summarizing and visualizing them is of primary interest for many applications such as viral marketing, extracting communities, immunization, and sense-making. However, most prior work focuses on generic structural summarization techniques or on developing specific algorithms for specific tasks. This is both tedious and challenging. As a result, for several popular or new tasks, there do not exist readymade summarization methods.

We explore a promising alternative approach instead. We propose NetGist, a framework which automatically learns *how* to generate a summary for a given task on a given unlabeled network. In addition to generating the required summary, this also allows us to *reuse* the learned process on other similar networks. We formulate a novel task-based graph summarization problem and leverage reinforcement learning to design a flexible framework for our solution. Next, we propose Guided-NetGist to leverage human suggestions to further improve the quality of the network summaries, for sense-making tasks. Via extensive experiments, we show that NetGist and Guided-NetGist robustly and effectively learn meaningful summaries, and help solve challenging problems, and aid in complex task-based sense-making of networks.

6.1 Introduction

As discussed previously, *task-based* summaries can help in solving various problems. For instance, consider collaborative visualization of document networks [20] where nodes represent documents and edges shows their similarities. Generating high-quality visualizations of



(a) Before considering human suggestions

(b) Our framework

Figure 6.1: Visualization of Crescent intelligence-related documents [31] (a) Before considering human suggestions and (b) After it by our framework. Each node represents a document in the dataset. Nodes with the same color are in the same supernodes. Edges show the satisfies user suggests that their end nodes are related to each other. Our framework satisfies more user suggestions and gives a summary that reflects the user's interests more.

these networks helps users to locate and read relevant documents and accomplish their goals faster. This is a challenging problem. There are ways to visualize such networks based on users interests [31]. However, they rely more on intuition, are not principled and are expensive. On the other hand, if someone gives us a summary of a document network, where nodes that are close in the eventual visualization are grouped together in the summary, then it is easy to intuitively imagine that we may be able to generate a high-quality visualization. So, given a task-based summary for the task which takes the user interests into account, we can potentially generate a high-quality visualization. Figure 6.1 shows a document network visualization given by our method using such an approach; indeed, as discussed in experiments later, it consistently generates better visualizations than state-of-the-art approaches.

Similarly, task-based summaries can help in other sense-making and visualizing hidden patterns in networks, e.g., for identifying bridges in different types of networks. Finally, task-based summaries can also help in getting a better quality solution for tasks with known algorithms [109].

Usually, past work has looked into constructing summaries preserving 'important' characteristics of the network (e.g., structural characteristics). Based on previous work, we make three observations. (1) Different tasks give rise to different network summaries. For example, the desired summary for the community detection task (which tries to find cohesive regions) is very different from the summary for a task like influence maximization (which seeks to find

central nodes). (2) There are tasks with no summarization algorithms such as collaborative visualization problem discussed above, traveling salesman problem, etc. Also, new tasks are getting defined on networks which may make it impractical to develop specific algorithms for each of them. (3) In real-world, we usually solve the same task repeatedly on similar networks that belong to the same distribution. For example, generating layouts for similar graphs is helpful in the context of protein networks [17], network-traffic data [177], etc.

Similarly, an advertiser may target the same social network repeatedly with probabilistic influence patterns [90]. Samples of networks from the same distribution are frequently observed in intelligence analysis [40], epidemiology [2] and many other settings.

Hence leveraging these three observations, in such scenarios, the inherent similarity between networks and the variety of tasks opens a space for *learning* task-based network summaries for networks coming from a distribution. This motivates the following informal learning problem:

Problem: Task-based Summaries. Given a problem *Prob*, and a distribution D of unlabeled network instances, can we learn *how* to generate meaningful network summaries that generalizes to unseen instances from D?

Leveraging recent advances in reinforcement learning and deep learning [26, 115], we develop NetGist, a flexible approach which automatically learns *how* to generate a summary for a given set of tasks. To the best of our knowledge, past work has not looked into this novel problem. Our contributions in this Chapter are as follows:

(I) Problem formulation: We formally formulate a novel problem of task-based graph summarization (TBNS) to find a 'good' smaller representations of unlabeled networks. We also formulate an extension of it as the guided task-based network summarization (GUIDED-TBNS) to learn a 'good' network summary incorporating human suggestions.

(II) Designing effective learners: We propose a general framework for this problem. Our framework learns the process of summarization as a 'meta-algorithm' for networks in the same distribution. Our framework can be used for a large set of graph optimization problems (GOP).

(III) Extensive experiments: We show the effectiveness of our framework NetGist on multiple diverse datasets. NetGist beats all the competitors and shows robust performance for different tasks and sizes of summary graphs. Also, we show that our framework makes it possible to generate high-quality solutions for the GUIDED-TBNS problem automatically. Finally, our case studies show NetGist gives high-quality visualizations which helps in sense-making for specific tasks.

6.2 Problem Formulation

Following prior work in network summarization [130, 46, 145], in a summary, it is natural to *group* ('merge') similar nodes to construct a graph of 'super-nodes' and 'super-edges' with a smaller size than the original network. In this way, each super-node represents homogeneous regions [46] in some sense and the connection between super-nodes highlights the overall structure of the network and important regions of the original graph [130]. Intuitively, our goal is to find a good network summary which guides us to find the solution for a given task.

Tasks: The first question is what kind of tasks we want to handle? As our first step towards learning task-based summaries for networks, we choose to solve a set of problems we call **G**raph **O**ptimization **P**roblems (GOP) which includes many popular graph mining problems. Suppose we have a graph $G(V, E)$ where V and E are sets of nodes and edges of the network. Also, assume Θ is the set of parameters of the given task. In a problem $prob \in$ GOP, the goal is to select $O \subseteq U$ (i.e. a set of objects O from the universe U of objects determined by the problem), that maximizes some quality function $F_{prob}(O; G, \Theta)$. For all GOP problems, U is some set of objects defined over the graph G (e.g. nodes, edges, subgraphs, etc.). Note F_{prob} maps a sets of objects to a real value based on the given task *prob* and its parameters Θ on the graph G. We formally define the set of graph optimization problems as follows:

Definition 16 *Graph Optimization Problem (*GOP*)*

Given *a graph $G(V, E)$, a set of input parameters Θ and a quality function $F_{prob}(O; G, \Theta) \in \mathbb{R}$ and set U,*

Find *the best set of objects O^* such that,*

$$O^* = \arg\max_{O \subseteq U} F_{prob}(O; G, \Theta) \tag{6.1}$$

Note that many combinatorial problems such as Minimum Steiner Tree, Maximum Clique, etc. [99] that defined over graphs can naturally be written as a GOP problem. Hence, the set of GOP problems is broad and includes many problems of interest. We use four common GOP problems in this Chapter to showcase our method and results (see Table 6.1).

What is a good summary? As mentioned above, to generate any summary we need to do two things: (Q 1) Evaluate the structure of the summary graph; and (Q 2) Identify homogeneous regions. Our main insight is that both of these issues are *task-dependent*. Consider the example figure left. We show the summary for the *Influence maximization* problem in figure (b). The summary highlights the central region of the graph which is the middle node in the chain (Q 1). It is natural to assume that to get the most central node we should only focus on the most central *region*. So a good summary quickly gives a high level understanding of the most central region in the graph. However, we still need to 'zoom into' the central region to find the most central node. Hence, the central node in the graph

Table 6.1: Four common graph optimization problems (GOP).

	Problem	$\mathbf{F_{prob}(O;G)}$	**U**	$\mathbf{O \subseteq U}$	**Description**
1	Influence maximization	$\sigma_G(O)$	V	$O \subseteq V$, $\|O\| = k$	Given the Independent Cascade model for diffusion, find a set O of k nodes in G as seed adopters which maximizes the expected number of final adoptions $\sigma_G(O)$ [88].
2	Community detection	$k - \sum_{i=1}^{k} \frac{links(C_i, V \setminus C_i)}{links(C_i, V)}$	$S \equiv$ all subgraphs of G	$O \subseteq S$, $\|O\| = k$	Group nodes into k communities which maximizes the # inside-group edges versus # outside-group edges [58].
3	Immunization	$\lambda_G - \lambda_{G \setminus O}$	V	$O \subseteq V$, $\|O\| = k$	Find k best nodes to remove from G to minimize the spread of contagion by maximizing the drop of leading eigenvalue λ of the graph adjacency matrix [34].
4	Maximum Cut	$\sum_{i=1}^{k} links(C_i, V \setminus C_i)$	$S \equiv$ all subgraphs of G	$O \subseteq S$, $\|O\| = k$	Group nodes into k cuts which maximizes the # outside-group edges [99].

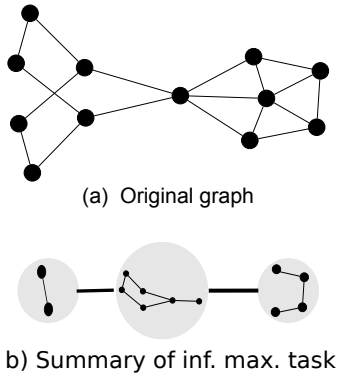


Figure 6.2: *Influence maximization* summary of our RUNNING EXAMPLE.

is central in the central region as well (Q 2) (this can be verified by actually running any influence maximization algorithm [36] as well). Hence the summary should also ensure that the 'homogeneous' nodes/regions which play a role in determining the solution (i.e. central nodes and other nodes which determine their centrality) should be grouped together.

This example shows that, intuitively, the given tasks guide us to both evaluate the structure and characterize homogeneous regions in the summary graph. The summary highlights how different super-nodes (regions) are connected to each other based on the given task. Also, each super-node in the summary shows a region in the graph which contains nodes with the same role (effect on the solution) based on the given task. Note that these properties also lead to visually appealing summaries, as they highlight the structural characteristics of the graph important to the given task.

Measuring Summary Quality: Given the above discussion, how should one precisely evaluate the structure and characterize the homogeneous regions for a particular task? Answering these questions is challenging as the answers are inter-dependent. Defining what characterizes a homogeneous region will necessarily impact the right measure to evaluate the overall structure (which essentially connects these homogeneous regions). Hence our idea is to identify properties and develop a procedure which answers both questions simultaneously.

We first formalize the notions of a 'good summary' having structural similarity to the original network as well as having homogeneous super-nodes. Let $G^*(V^*, E^*)$ be an ideal summary of the original network $G(V, E)$ for an arbitrary GOP problem *prob* with parameters Θ. Also let O^* be the optimal solution for *prob* on G. Then we want the following properties to hold for G^*.

Property 1 *Let* $O'^* = \arg\max F_{prob}(O', G^*, \Theta^*)$. *Then,* $\forall_{o \in O^*}, \exists H(o) = x \in O'^*$.

Property 1 states that there exists some mapping function H which maps all objects in the optimal solution for *prob* on G to a unique element of the optimal solution for the *prob* on G'. This property indicates that G and G' have structural similarity w.r.t. *prob* due to the mapping between the solutions.

Property 2 *For any two pairs of nodes* v_i *and* v_j *in an arbitrary super-node* x *in the summary,*

$$F_{prob}(\{v_i\}, G, \Theta_x) > F_{prob}(\{v_j\}, G, \Theta_x) \implies$$
$$F_{prob}(\{v_i\}, x, \Theta) > F_{prob}(\{v_j\}, x, \Theta)$$

Property two implies that the order of role of nodes inside each x (which are super-nodes), is preserved. This property indicates that the nodes in each super-node have a homogeneous role in determining the solution and they lead to the right solution as their order is preserved.

These two properties indicate if an ideal summary G^* for G w.r.t. to GOP problem *prob* exists, we can reconstruct the optimal solution to *prob*. For finding the most central node for example, due to Property 1, first solving for the *prob* in G' leads us to the correct super-node. And since Property 2 indicates that the order of the quality is maintained inside the super-node, we just need to solve the *prob* in this super-node. Now, the question is what about the sub-optimal summaries? How do we determine how far they are from G^*? To answer this question, we propose a divide-and-conquer Algorithm 12 to measure the quality of any summary G' of G, which follows the same strategy as mentioned earlier: first solve *prob* on G' and recursively solve for *prob* on each solution obtained in the first step. And intuitively the quality of the solution tells us how good the summary is. Lemma 18 indicates that the Divide and Conquer framework we propose returns optimal score for the summary network that satisfies both the properties.

Lemma 18 *Algorithm 12 on a summary network G' returns the optimal solution O^* and optimal score $F_{prob}(O'^*; G', \Theta)$ if G' satisfies both Properties 1 and 2.*

Proof. Each element o of O^* can be mapped to a unique element x in Φ^* obtained in line 3 of Algorithm 12 by Property 1. As per property 2, the order of $F_prob(v, x, \Theta)$ is preserved for each v inside each x. Hence step 8 of the algorithm returns an element o of O^*. Since, the outer loop in line 5 is over all element of Φ^*, each iteration returns an element of O^*. Hence, $O'^* = O^*$, if G' satisfies both Properties 1 and 2. ■

Note that validating whether a summary satisfies Property 2 is computationally expensive as it involves repeatedly measuring F_{prob}. Our approach avoids this step and still manages to output the ideal solution and the corresponding optimal score F_{prob} with respect to *prob* if the summary network satisfies both the properties.

Algorithm 12 Divide and Conquer

Require: $G(V, E)$, $G'(V', E')$, $prob \in$ GOP
1: $O'^* \leftarrow \emptyset$
2: //Divide
3: $\Phi^* \leftarrow \arg\max F_{prob}(\Phi; G', \Theta)$
4: // Conquer
5: **for** $\phi \in \Phi^*$ **do**
6: $G_\phi \leftarrow$ subgraph of ϕ in G
7: $\Theta_\phi \leftarrow$ sub-parameters that related to ϕ
8: $o^*_\phi = \arg\max F_{prob}(o_\phi; G_\phi, \Theta_\phi)$
9: $O'^* \leftarrow O'^* \cup o'^*_\phi$
10: return O'^* and $F_{prob}(O'^*; G, \Theta)$

Problem definition: We are now ready to define our problem formally. Suppose we have a network $G(V, E)$ where V and E are sets of nodes and edges of the network respectively. Also, suppose we are given a graph optimization problem $prob \in$ GOP (Def. 16) which asks to find the best set of objects O^*. The parameters of *prob* and the graph G are drawn from a probability distribution D (i.e. $(G(V, E), \Theta) \sim D$). For example, D can be the set of probabilistic influence patterns (as discussed in the introduction). We want a summary network $G'(V', E')$ for any graph $G(V, E)$ drawn from D such that $|V'| = \alpha \cdot |V|$, where α is the 'reduction factor'. It is expensive to generate the high-quality summary for all the graphs in D. Hence, our idea is to learn the *process* of learning the summaries instead of merely the final summary graphs. Our goal is to learn a graph summarization process such that solving the given problem on G' using our divide-and-conquer strategy (Algorithm 12) results in a set O'^* such that $F_{prob}(O'^*; G, \Theta)$ is similar to the $F_{prob}(O^*; G, \Theta)$. Formally:

Problem 4 *Task-Based Network Summarization (*TBNS*)*

Given *a graph optimization problem (prob* $\in$ GOP*), a distribution of problem instances* D*, and a reduction factor* $0 < \alpha \leq 1$*.*

Learn *a graph summarization process such that for any* $(G(V, E), \Theta) \sim D$*,*

$$\max \rho = \mathbb{E}_{(G,\Theta)\sim D}\left[\frac{F_{prob}(O'^{*}; G, \Theta)}{F_{prob}(O^{*}; G, \Theta)}\right] \tag{6.2}$$

where, O'^{*} *is the solution of the given problem prob on the summary graph* G' *using Algorithm 12.*

Comments: Generalizing a learned model to unseen instances is important from the learning perspective. A good model which is trained on a particular 'training' data is expected to do reasonably well in the 'test' data drawn from the same distribution [113]. Hence, our ultimate goal is to maximize the expected ratio ρ over all the instances (including the unseen ones) of (G, Θ) drawn from D. Note that in some real-world scenarios, D can be a single (G, Θ) (see Sec. 6.4).

6.3 NetGist: Solving TBNS problem

Next, we show how to solve our TBNS problem. We want to precisely learn the steps to be taken for the summarization process, so that we can re-apply the same approach on other graphs from the same distribution. A basic summarization step we take is some sort of a 'merge' operation. The merge operation basically groups nodes in the original network and results in a so-called 'super-node' in the summary network. The question at hand is that once the summarization step is decided, how do we measure the 'goodness' of each step and how do we obtain a quality summary? Reinforcement Learning (RL)—more specifically Q-learning [150]—is a natural solution to the above questions. In general, RL methods learn to take an '*action*' in an '*environment*' to maximize cumulative '*reward*' based on a '*transition*' function.

Recently, the success of deep reinforcement learning [115], where convolutional neural networks are used to learn a representation of 'states', in solving various AI tasks has gained much attention. It is known to perform better and converge faster than the traditional reinforcement learning. Hence, we propose to leverage a deep reinforcement learner to solve our TBNS problem.

6.3.1 Overview of NetGist framework

We now give an overview of our NetGist approach (Algorithm 13). Overall, in our deep reinforcement learning paradigm, the initial 'state' is the network $G(V, E)$ drawn from D

Algorithm 13 Overview of NetGist

Require: $prob \in$ GOP, α, D
1: Randomly Initialize the deep Q-learning parameters
2: **for** $i = 1$ **To** num of samples **do**
3: Sample $G(V, E)$ and Θ drawn from D
4: // learning how to summarize
5: **for** episode=1 **to** T **do**
6: $G'(V', E') \leftarrow G(V, E)$
7: **while** $|V'| > \alpha \cdot |V|$ **do**
8: Merge $G'(V', E')$ // (See Section 6.3.2)
9: // Evaluate
10: $O', F_{prob} \leftarrow$ DivideAndConquer$(G, G', prob)$ Algorithm 12
11: // Optimize
12: Update the deep Q-learning parameters to yield better summary (see Section 6.3.4)
13: **Return** the trained model (which solves the TBNS problem)

and each 'action' the learner takes reduces the size of original network G to produce a new 'state', a summary network $G'(V', E')$. The learner repeatedly takes actions till the summary network $G'(V', E')$ is of desired size. Next, we evaluate the *prob* on G' using Algorithm 12. Based on the quality of the solution, the learner's parameters are updated. The learner repeats the process on more samples from D until it learns to summarize any $(G, \Theta) \sim D$ to G', such that ρ is maximized. To formalize our framework, we need to answer some questions and make design choices. These question are: **Q1** How to define the universe of meaningful actions that summarizes any network $G(V, E)$ drawn from D into $G'(V', E')$? **Q2** What is the universe of all possible 'states'? and **Q3** What are the reward, policy, transition functions and the overall learning procedure? We answer these questions next.

6.3.2 Q1. Universe of Actions

An action a taken at state s basically gives us the next state s'. Since our initial state is a network G from D and our goal is to summarize it, each action a should reduce the size of the network G. As briefly indicated above, we basically define our action as a merge operation. Specifically, a merge operation on a node pair $\{a, b\}$ in the network G would result in a new *super-node* c in G'. Moreover, new edges (v, c) for all nodes v which are neighbors of either nodes being merged would be added to G'. Now to precisely define our action a we need to answer following two questions. What kind of node-pairs $\{a, b\}$ can be merged together? How many node-pairs can be merged at one step? To answer the first question, it is important to note that merging different types of nodes is useful for different graph optimization problems. For example, for tasks like community detection, it

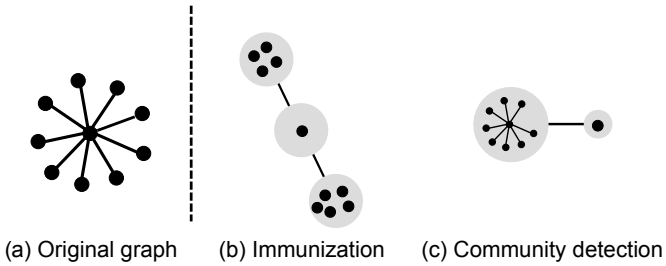


Figure 6.3: A star network (a). Summary graphs of *Immunization* (b) and *Community detection* (c) tasks.

may be more meaningful to merge nodes which have an edge connecting them to ensure that densely connected nodes are grouped together (Figure below (c)). In contrast, for task like immunization, it may be more meaningful to merge nodes around a central node to ensure that the central node, which is more likely to be in the solution, remains central in the summary network (Figure below (b)). It is also important to group fringe nodes, which are well separated, to ensure that the learner is able to focus on the important regions of the network. Hence, merging various types of node-pairs could be important to obtain meaningful summary. Therefore, we allow merging nodes that are any distance apart. To formalize this intuition, we introduce the concept of a k^{th}-order merge. Suppose we are merging nodes a and b in $G(V, E)$. Let $NB(a)$ be the set of all neighbors of node a and $d(a, b)$ be the shortest hop distance between the nodes in G. We define our merge operation as follows:

Definition 17 *(k^{th}-Order Node Pair Merge) A k^{th}-Order node pair merge operation merges nodes a and b into a new node c, such that $a \in V$, $b \in V$, and $d(a, b) = k$. We add new edge (c, k) for all the nodes $k \in NB(a) \cup NB(b)$.*

Now let us answer the second question. Since, we allow k^{th}-order merge operations, merging any two nodes in the network is possible. However, this yields a $O(n^2)$ possible merge operands, which is intractable. Hence, in a single action a, we first decide on the order k of the merge operation. Then we merge all possible node pairs, which can be merged with a k^{th}-order merge. Formally, our universe of actions is defined as follows:

Definition 18 ***Universe of Actions*** *The universe of actions A in NetGist is a set of all possible k^{th}-order merges.*

Note that our single action is a set of k^{th}-order merge operations, for a fixed order k. Hence, we usually merge more than a single pair of nodes. This ensures that our universe of actions is broad enough to incorporate meaningful merge operations for different tasks and yet restrictive enough to be tractable for learning purposes.

Lemma 19 *k^{th}-Order node pair merge is order sensitive.*

Proof. Assume given a set of k^{th}-Order merges, application of all the merges in the set in any order results in the same summary network $G'(V', E')$. Now, let us assume we have a chain $G(V, E)$, where $|V| = n$. Let also assume that n is an odd number. Note that the diameter of G is then $n - 1$ Let the set of k^{th}-Order merges be $\{n - 1^{\text{th}}\text{-Order}, 1^{\text{st}}\text{-Order}\}$.

Consider the first possible sequence: ($n - 1^{\text{th}}$-Order, 1^{st}-Order). Since the diameter of the network is $n-1$, there are only two nodes, which are $n-1$ distances apart. Hence, in the first step only these two nodes are merged. This results is $G_1^1(V_1^1, E_1^1)$. Note that $|V_1^1| = n-1$ and G_1^1 is a cycle. Therefore, the step (1^{st}-Order merge) results in the final summary network $G_2^1(V_2^1, E_2^1)$ where $|V_1^1| = \frac{n-1}{2}$.

Consider the next possible sequence: (1^{st}-Order, $n-1^{\text{th}}$-Order). There are n nodes in G and n is odd. Hence, the application of 1^{st}-Order merge results in $G_1^2(V_1^2, E_1^2)$, where $|V_1^2| = \frac{n-1}{2}+1$ The second step does not have any effect as the diameter of G_1^2 is strictly less than $n - 1$. Hence the final summary network is $G_2^2(V_2^2, E_2^2)$ where $|V_2^2| = \frac{n-1}{2} + 1$

Contradiction: The resulting summary networks from two different orderings of same set of k^{th}-Order merges has different number of nodes. So, the summaries are different. ■

Given the lemma 19 the order of taken actions is essential in generating the summaries. Hence, we should learn the sequence of actions as well as the set of actions to take.

6.3.3 Q2. Universe of States

The initial state in our RL framework is any network $G(V, E)$ drawn from D and each action the learner takes, results in a summary network $G'(V', E')$. Recall that our one action is a set of k^{th}-order merge operations. Any state s other than the initial state is a result of one or more actions on the initial state, the network G. Hence the universe of states, which includes the original network as well, is defined as follows:

Definition 19 ***Universe of States*** *The universe of states S in NetGist is a set of networks $G'(V', E')$, such that G' is a result of zero or more merge operations on the a network G from D.*

Note that our universe of states does not include all possible summary networks of network G from D. This is because our action merges multiple nodes at once. This reduces the size of the universe of states, which in turn helps in learning. However, as discussed earlier, this set is still large enough to explore meaningful summaries across the search space.

6.3.4 Q3. Reward, Policy, Transition and the Learning Procedure

Having described the states and actions earlier, now we can describe our method to learn how to get a meaningful summary.

Reinforcement learning.

As mentioned earlier, we propose to leverage Deep Reinforcement Learning (DRL) to learn the summarization process. Our idea is to use deep Q-learning as it is an off-policy RL which known to be more sample efficient than the policy gradient methods [162]. Before, explaining our Deep Q-learning framework first we should set up our Reinforcement Learning (RL) procedure. Next, we describe how we design our end to end RL framework to learn the graph summarization process.

RL gives the quality of taking an action a in each state s. To formulate the RL framework, we should define state, actions, transition, rewards, and policy function. We have already defined the states and actions in Sections 6.3.3 and 6.3.2 respectively. Now let us start with the transition function. The result of taking an action a on the current state s is determined by the transition function. We design our transition function $\mathcal{T}$ as a deterministic function which takes the current state s and action a and returns the next state— $\mathcal{T}(s,a) = s_{next}$. Having a deterministic transition function suffices in our case as both the universe of state and actions are carefully designed with consideration of effect of each action. Note that the deterministic transition function also helps in faster convergence.

Next we define the reward to be -1 for a state s, unless it is a terminal state. A terminal state in our case is a summary network $G'(V', E')$ which has the desired size. Note that not having a predefined reward function (i.e., -1) would essentially require us to solve the task at hand at each state, which would be a costly process. Formally, we define our reward function as follows:

$$r(s,a) = \begin{cases} \max\limits_{O \subseteq U} F_{prob}(O; s_{next}, \Theta) & \text{if } s_{next} \text{ is a terminal state} \\ -1 & \text{otherwise} \end{cases} \tag{6.3}$$

In Eq. 6.3 we compute $F_{prob}(O; s_{next}, \Theta)$ on the summary s_{next} with our divide and conquer framework in Algorithm 12.

Now that we have defined our reward function, let us define the policy function. The policy function $\pi(a^*|s)$ specifies what action to take at a given state. It is defined as $\pi(a^*|s) = \arg\max \text{Q-value}(s,a)$ where Q-value(s,a) is the Q-value of the state s and action a which estimates the expected cumulative reward we achieve after taking action a at state s [150]. Our goal is to learn optimal Q-value function results in the highest cumulative reward. We leverage the Q-learning algorithm [150] which iteratively updates Q-value(s,a) until

convergence.

Learning algorithm

Next, we elaborate on the whole learning pipeline. Note that our pipeline learns the summarization procedure for graphs in D such that it is generalizable to the unseen graphs. First we define how to estimate the Q-value of the state s and action a, Q-value(s, a), using the Q-learning algorithm. We define Q-value of a state and action as the expected rewards in future as follows,

$$\text{Q-value}(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] \tag{6.4}$$

The optimal Q-value function is the maximum available cumulative reward achievable with state s and action a. That is Q-value$^*(s, a) = \max$ Q-value(s, a). Using Eq. 6.4 we iteratively estimate Q-value$^*(s, a)$ as follows,

$$\text{Q-value}_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} \text{Q-value}_i(s', a') | s, a\right] \tag{6.5}$$

The vanilla Q-learning algorithm is not scalable specially in our case where both the state space and actions space are large. In order to have an end-to-end framework to learn the optimal Q-value(s, a) we combine CNNs [68] with Q-learning as often done in literature. CNN helps us to have a compact representation of each state s in the universe of states. An advantage of using CNN is that it uses fewer number of neurons than the fully connected layers. Hence, we use deep neural networks with CNNs to approximate Q-value$^*(s, a)$.

In our deep RL framework the input state s is fed into the CNN layer. The CNN outputs a feature representation of the state, which is then fed into a fully connected Feed Forward Network, FFN. The output layer of the FFN is d-dimensional, where d is the number of possible actions. Each i^{th} entry in the output vector represents the predicted Q-value function Q-value(s, a_i) for the state s and the action a_i.

Lemma 20 *Time complexity of NetGist is $O(Itr \cdot VE \log V)$, where Itr is the number of iterations.*

Proof. In the first step, we feed the adjacency matrix of G to the first layer of a CNN. The size of the filter in this layer is $8 \times 8 \times 32$ with stride equal to 8. Hence, have $\frac{V^2}{64}$ of 8×8 matrix multiplication. Hence, the computational cost of the layer with be $O(\frac{V^2}{64} \times 328 \times 2.3737) = O(V^2)$. Similarly, we can compute the computational cost of the next 2 layers. Hence, the time complexity of CNN step is $O(V^2)$. Also, the time complexity of fully connected layer is $O(dim^2)$, where dim is the output size of the last layer of CNN. $dim << V$, so the

overall complexity of the neural network is $O(V^2)$. To take a k^{th}-Order merge we compute all-pairs-shortest-path which is $O(VE \log V)$. Note, the number of steps is $<< V$ and can be considered as a constant. Finally, we can assume the running time of Algorithm 12 is constant. Hence, the overall time complexity is $O(V^2)$. ■

6.4 NetGist with human suggestions

We designed NetGist to learn to summarize networks for GOP tasks. Until now we assumed that the GOP tasks have well-defined objectives and F_{prob}. However, in reality, the objective of tasks are not always well defined. It is common that we merely have a vague understanding of a good solution and can give an explanation for it in the form of suggestions. Can we extend NetGist to incorporate human suggestions to summarize networks for such tasks with unclear objectives? In this section, we extend NetGist to incorporate human suggestions as it has many applications. We specifically ground our work on collaborative visualization. This is a type of data visualization in which a human interacts with the computer to generate a graphic illustration of the information effectively [133, 152]. Collaborative visualization is an important area and has seen growing research work in recent years [31, 19, 52]. One of the state-of-the-art methods in collaborative visualization is StarSPIRE [30]. It is a visual text analytics tool which generates a layout for a network of documents. It leverages user semantic interactions on documents to update the layout with a heuristic approach and provides a high-quality relevancy-based document visualization.

Motivation. Generating a high-quality relevancy-based layout can be converted to a GOP problem with a vague relevancy-based objective which we explain later in Section 6.4.2. However, human users can provide apriori suggestions about the ideal location and closeness of documents in an for a visualization at the beginning of the process of generating layouts. We think that NetGist can help StarSPIRE to accomplish the goal of generating high-quality relevance-based layouts. Additionally, in StarSPIRE, users often have to work with diverse yet similar document corpora. This makes it a good fit for our framework as we can learn the right summary/visualization and reuse it. Achieving this goal is not possible by simply using the similarity between documents. However, we can incorporate human suggestions at the beginning of the learning process in a principled way to solve the corresponding GOP task of generating graph layouts. Hence broadly speaking, we want to learn to summarize such document networks with the help of human suggestions, such that we can effectively visualize document networks in a 2-dimensional space.

6.4.1 Overview of StarSPIRE

StarSPIRE [30] can be considered as a content-based recommendation system in a document network $G(V, E, W)$, where V is the set of documents, E is the set of edges representing the

similarity relation between them, and W is the set of edge weights representing the amount of similarity based on their word usage. StarSPIRE uses a weighted, force-directed layout to place similar documents based on user interest nearer to each other in the 2-D visualization of G. The framework learns the user interest from its semantic interactions such as opening, minimizing, removing, annotating and highlighting documents. For example, if the user opens a document and expresses her interest in it through the semantic interactions such as bookmarking or highlighting parts of it, StarSPIRE visualizes similar documents on screen and recommends them to read. So the user can find more relevant documents easier. If $\Upsilon(i,j)$ is a function that shows the relevance of document i to j based on the user interest, the goal is to visualize the relevant documents with higher $\Upsilon(i,j)$ nearer to each other to recommend them and guide the user. More formally we can consider the StarSPIRE task as follows:

Problem 5 *StarSPIRE Document Visualization*

Given *a document network $G(V,E,W)$, and a relevance function $\Upsilon(\cdot,\cdot)$,*

Generate *a graph visualization $\text{VIZ}^* \in \mathbb{R}^{|V|\times 2}$ of G in a 2-d space such that,*

$$\text{VIZ}^* = \arg\min_{\text{VIZ}} \sum_{i,j\in V} \Upsilon(i,j)\cdot[F_{\text{repulsive}}(i,j) + F_{\text{attractive}}(i,j)] \tag{6.6}$$

$$s.t. \qquad \forall_{i,j\in V} ||\text{VIZ}(i) - \text{VIZ}(j)||_2 > 0 \tag{6.7}$$

Where,

$$F_{\text{repulsive}}(i,j) = \frac{-1}{||\text{VIZ}(i) - \text{VIZ}(j)||}$$

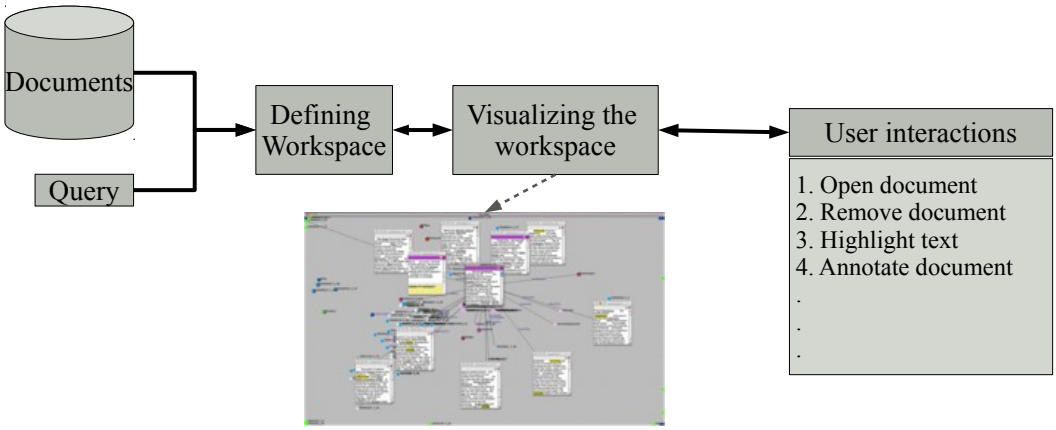
$$F_{\text{attractive}}(i,j) = ||\text{VIZ}(i) - \text{VIZ}(j)||^2$$

(the constraints guarantees that there is not overlapping in the visualization VIZ*)*

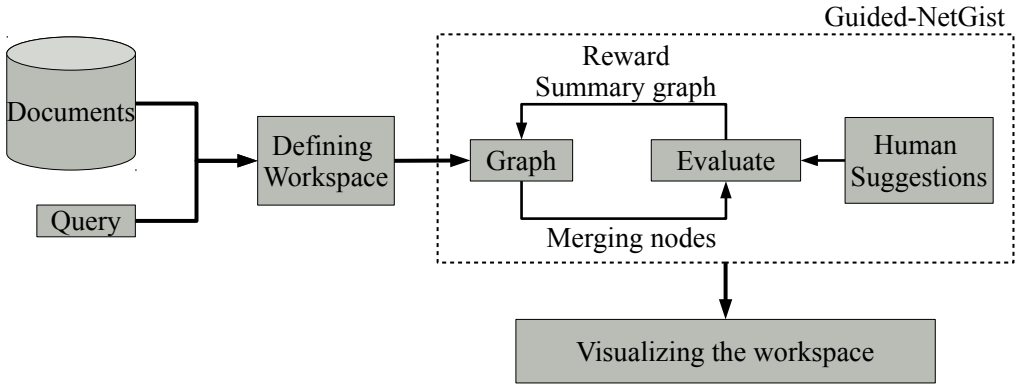
This problem is challenging because it is hard to define the relevance function Υ which characterizes the user's interests. Also, each user has a unique interest which can also depend on the task. Figure 6.4a shows an overview of StarSPIRE pipeline.

6.4.2 Challenges and main idea

Main idea. Our intuition is that a good network summary can help in generating a high-quality visualization for StarSPIRE problem. Consider a summary network where each supernode contains nodes that are most relevant to each other according to the user's interest. Clearly, the structure of the summary graph gives us an idea about the relationship between documents in different supernodes. For example, a 'star' structure means the nodes in the



(a) StarSPIRE pipeline. It updates the visualization by user interactions.



(b) Guided-NetGist pipeline. It incorporates human suggestion in the beginning of the learning process to visualize document networks.

Figure 6.4: An overview of (a) StarSPIRE and (b) Guided-NetGist pipelines.

center of the star are important and related to other nodes while the spokes are not related to each other. If we have such a summary, first we can visualize the summary network. Next, we can visualize the supernode that the user is interested in working with to suggest the most relevant documents to the user. Hence, our main idea is to use NetGist to precisely learn the network summary with these characteristics. Generating a high-quality layout for the StarSPIRE task is not a GOP problem and finding the desired summary cannot be simply formulated as a TBNS problem. However, we can tweak StarSPIRE to shape it into a partitioning problem StarSPIRE-Partition, as follows:

Problem 6 *StarSPIRE-Partition*

Given *a document network $G(V, E, W)$, and a relevance function $\Upsilon(\cdot, \cdot)$ and number of partitions Θ and U =all the sub-graphs of G,*

Find *a set of non-overlapping and exhaustive subsets of nodes in G as the graph partitioning $O^* \subseteq U$ where $|O^*| = \Theta$ such that,*

$$O^* = \arg\max_{O \subseteq U} F_{\textit{StarSPIRE-Partition}}(O; G, \Theta) \tag{6.8}$$

$$F_{\textit{StarSPIRE-Partition}}(O; G, \Theta) = \sum_{o \in O} \sum_{i,j \in o} \Upsilon(i, j) \tag{6.9}$$

It is easy to see that StarSPIRE-Partition $\in$ GOP. Intuitively, StarSPIRE-Partition partitions the graph in a way to maximize Υ for nodes in the same supernodes. Essentially each partition is a super-node of the desired summary for StarSPIRE. Figure 6.4b shows an overview of such pipeline.

Challenges. The main challenge toward our goal is that there is no clear definition of relevance function Υ to guide us. Instead, we only can get some examples of nodes that should/should not be grouped in the desired summary in the form of user suggestions at the beginning of the learning process. Hence, we extend our summarization problem TBNS to be able to handle the human *suggestions* and use this insight to improve the quality of the final summary. Also, based on definition 5 the document graphs are weighted. However, in the vanilla TBNS, we assumed that the input graphs are unweighted. So, we introduce some updates to TBNS to handle summarizing weighted graphs as well.

6.4.3 Aggregating human suggestions: Details

A user can express which nodes should be or should not be grouped in the form of one-time suggestions. If the user suggests that a node pair (i, j) are related, the pair will be in the set of positive suggestions C^+. In the same way, if the user suggests that a node pair (i, j) are not related, the pair will be in the set of negative suggestions C^-. More formally we define user suggestion as follows,

Definition 20 *(Human suggestion C^+/C^-)*

$$C^+ = \{(i,j)|i,j \in V \textit{ and user suggests } \Upsilon(i,j) \textit{ is high.}\} \tag{6.10}$$
$$C^- = \{(i,j)|i,j \in V \textit{ and user suggests } \Upsilon(i,j) \textit{ is low.}\} \tag{6.11}$$

Constraint graphs. We represent the human suggestions with two graphs which we call constraint graphs.

Definition 21 *(Positive/Negative constraint graph G^+/G^-)*

$G^+(V^+, E^+, W^+)$ is a positive constraint graph iff

$$V^+ = V \qquad \textit{and} \qquad \forall_{(a,b)\in C^+}(a,b) \in E^+ \tag{6.12}$$

Where C^+ is the set of positive human suggestions (Equation 6.10).

Similarly, $G^-(V^-, E^-, W^-)$ is a negative constraint graph iff

$$V^- = V \qquad \textit{and} \qquad \forall_{(a,b)\in C^-}(a,b) \in E^- \tag{6.13}$$

Where C^- is the set of negative human suggestions (Equation 6.11).

Problem formulation

Our goal is to use human suggestions as a guide to merge relevant documents with high Υ and satisfy them as much as possible. In other words, we want to group node pairs in C^+ and ungroup the ones in C^-. Formally, our problem can be stated as:

Problem 7 *Guided Task-Based Network Summarization (*GUIDED-TBNS*)*

Given *a graph optimization problem (prob ∈ GOP), a distribution of problem instances D, human suggestions C^+/C^- and a reduction factor $0 < \alpha \leq 1$.*

Learn *a graph summarization process such that for any $(G(V,E,W),\Theta) \sim D$,*

$$\max \rho_{\text{GUIDED}} = \mathbb{E}_{(G,\Theta)\sim D}\left[\frac{F_{prob}(O'^*;G,\Theta) + \frac{\lambda_1\cdot\alpha\cdot|V|}{|E^+|}\sum_{y_i} y_i^T A_{G^+} y_i - \frac{\lambda_2\cdot\alpha\cdot|V|}{|E^-|}\sum_{y_i} y_i^T A_{G^-} y_i}{F_{prob}(O^*;G,\Theta) + 2\cdot\alpha\cdot|V|}\right] \tag{6.14}$$

where, O'^* is the solution of the given problem prob on the summary graph G' using Algorithm 12, and A_G is adjacency matrix of graph G and the GOP *task is the StarSPIRE-Partition problem.*

Extending NetGist

To solve the new problem of GUIDED-TBNS we propose an extension of our summarization algorithm Guided-NetGist. Algorithm 14 is an overview of Guided-NetGist. In Guided-NetGist, first, we sample the initial states from D (Line 3). Then we keep merging nodes using to reduce the size of the graph until the summary graph G' is of the desired size (Lines 7-8). Next, we evaluate the summaries by evaluating Equation 6.14 (Lines 10-11). To calculate Equation 6.14 we use Algorithm 12 and count the number of satisfied suggestions. Finally, the learner updates its parameters and repeats the process for the next episode (Line 13).

Weighted merging. In the original NetGist, we assumed the input graphs are unweighted with no attribute. However, in Definition 7 the nodes can be attributed, and the edge weights should reflect the similarity between nodes. Hence, we update the definition of the k^{th}-order merge in Definition 17 to capture the edge-weights as well.

Definition 22 *(Weighted k^{th}-Order Node Pair Merge) A k^{th}-Order node pair merge operation merges nodes a and b into a new node c, such that $a \in V$, $b \in V$, and $d(a,b) = k$. We add new edge (c,i) for all the nodes $i \in NB(a) \cup NB(b)$ with weight $\frac{W(a,i)+W(b,i)}{2}$.*

Algorithm 14 Overview of Guided-NetGist

Require: $prob \in$ GOP, α, D, G^+, G^-
1: Randomly Initialize the deep Q-learning parameters
2: **for** $i = 1$ **To** num of samples **do**
3: Sample $G(V,E,W)$ and Θ drawn from D
4: // learning how to summarize
5: **for** episode=1 **to** T **do**
6: $G'(V',E',W') \leftarrow G(V,E,W)$
7: **while** $|V'| > \alpha \cdot |V|$ **do**
8: Merge $G'(V',E',W')$ // (See definition 22)
9: // Evaluate
10: $O', F_{prob} \leftarrow$ DivideAndConquer$(G, G', prob)$ Algorithm 12
11: Calculate Equation 6.14
12: // Optimize
13: Update the deep Q-learning parameters to yield better summary (see Section 6.3.4)
14: **Return** the trained model (which solves the GUIDED-TBNS problem)

6.5 Empirical Study

We design various experiments to evaluate NetGist. Specifically, we answer the following questions:

(Q1) Does NetGist give high-quality, summary graphs for a given GOP task and a network distribution D?

(Q2) Can NetGist detect anomalies on mobility graphs?

(Q3) Does NetGist robust to change of parameters of the TBNS problem?

(Q4) Does NetGist give high-quality solution for StarSPIRE problem?

We used Python and PyTorch to implement all the steps of NetGist and our code is available for research purposes[1]. Although traditionally RL can be expensive, we implement the deep Q-learning on GPUs, which makes it effective to already solve practical novel problems (like StarSPIRE and sense-making). Additionally, we set $\alpha = 0.3$ for all our experiments. However, we examined the robustness of NetGist to changes in α and Θ as parameters of the TBNS problem and found it to be stable over wide ranges.

6.5.1 Data

We collected a number of datasets from various domains for our experiments. See Table 6.2 for details.

		Dataset	#Nodes	#Edges
Synthetic	1	ER	20-2000	50-10000
	2	BA	20-2000	40-4000
	3	Block Model	20-2000	40-30000
Real-world	4	Karate Club	34	78
	5	Work-Place	92	755
	6	High-School	327	5,818
	7	Political Blogs	1,490	16,783
	8	Facebook	4,039	88,234
	9	As-Oregon	7,000	15,743

Table 6.2: Datasets details.

• ER, BA, and Block Model are synthetic random networks created using Erods-Reyni, Barabasi-Albert preferential attachment [8] and Block models [47] respectively.

[1] http://github.com/SorourAmiri/NetGist

• KARATE CLUB is a social network among members of a karate club. The nodes represent the members of the club and edges indicate social ties outside the club.

• WORK-PLACE and HIGH-SCHOOL are mobility networks. Nodes represent individuals and edges indicate physical contact between the two[2].

• POLITICAL BLOGS is a widely used network of hyperlinks between web-blogs on US politics. The nodes are blogs website and edges are hyperlinks between them.

• FACEBOOK[3] is a social network. Nodes are users and edges indicate friendships.

• AS-OREGON[3] is an autonomous systems peering information inferred from Oregon route-views. Nodes are autonomous systems and edges show peering relationships.

6.5.2 Baselines

We compare the performance of NetGist with other summarization techniques which are listed below.

1. **Random** selects random node-pairs and merges them until the size of the summary G' is small enough.
2. **CoarseNET** [130] summarizes the network G to G' while ensuring that the propagation properties of G are preserved.
3. **METIS** [87] and Spectral [158] partition the network G while ensuring the summary is useful for the *Community detection* task. We take each partition as the super-node of the summary network and connect two super-nodes a and b with an edge if there is at least one edge between the nodes in a and b.

6.5.3 Q1. Quality of NetGist on distribution D

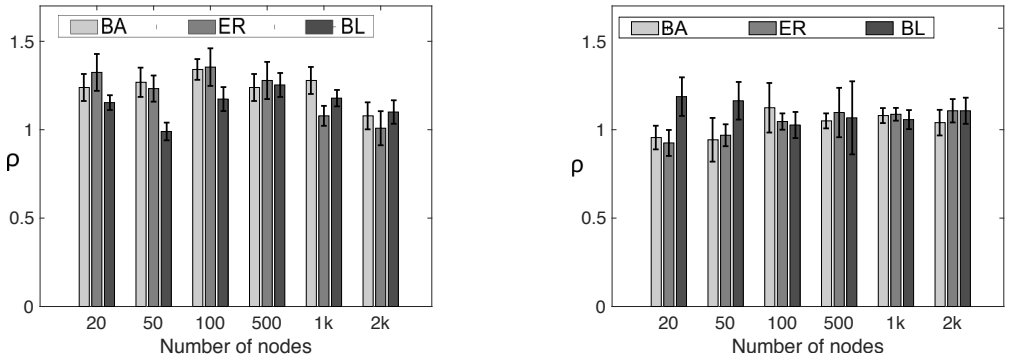
Here we evaluate the quality of NetGist on various networks and distributions.

Synthetic networks

To evaluate the quality of NetGist on distribution D we use the popular Erdos-Renyi (ER), Barabasi-Albert (BA) and Block Models to generate graphs from the same distribution. We trained NetGist on 80 randomly sampled networks from a distribution (BA, ER or Block

[2]http://www.sociopatterns.org/
[3]http://snap.stanford.edu

Model) and test on 20 networks previously unseen networks drawn from the same distribution. The average ρ (Equation 6.2) and standard deviation on the test test with respect to two GOP tasks are shown in Figure 6.5. Clearly, NetGist generalizes well and gets high ρ for both tasks for all the network distributions with various sizes. The ρ is relatively lower for the *Community detection* task as there is no reasonable community structure in random networks.



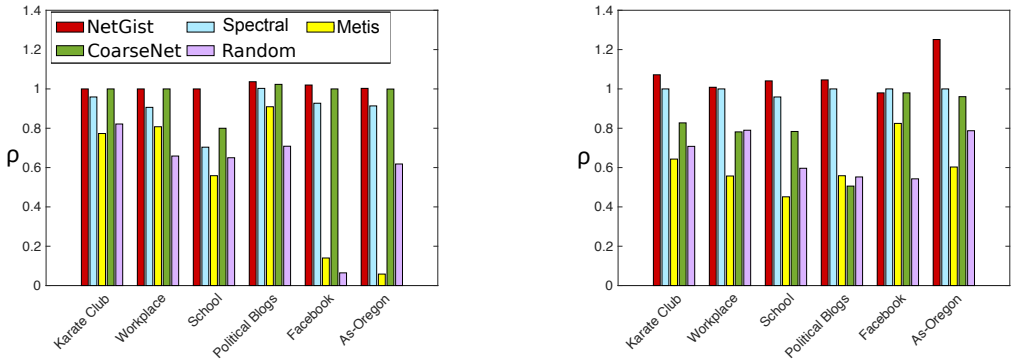
(a) *Influence maximization* (b) *Community detection*

Figure 6.5: ρ on test graphs from D. NetGist obtains high ρ consistently indicating that it learns well how to summarize networks for a given task.

Real-world networks

We ran NetGist and the baselines on six different real datasets to get task-based summaries for two GOP tasks. NetGist runs in less than half a second per iteration. We compute ρ and show them in Figure 6.6.

As shown, NetGist results in high ρ values across various datasets and tasks. For the *Influence maximization* problem, $\rho \geq 1$ across all datasets. Note that our method even slightly outperforms CoarseNET, which is specifically designed for the *Influence maximization* task (although it performs well too). This can be attributed to the fact that CoarseNET selects a random seed from a 'super-node' while we leverage our divide-and-conquer framework to ensure the best node is selected. For the *Community detection* task, NetGist outperforms baselines consistently. Note that we outperform both METIS and Spectral, which are designed for the *Community detection* task. Overall, the results show that NetGist even outperforms baseline methods specifically designed for the task, implying NetGist summaries can help solve GOP tasks.



(a) *Influence maximization* (b) *Community detection*

Figure 6.6: ρ of NetGist summaries. NetGist outperforms all the baselines and performs consistently for all the tasks and datasets. Note, the quality of NetGist solutions is equal or even better than the algorithms that are designed for the given task across all datasets.

Sample Visualization

Here we visualize our learnt summaries from the well-known KARATE CLUB network (Figure 6.7) to demonstrate the difference between various tasks. The *Influence maximization* task-based summary consists of two super-nodes which have considerably higher degree than the rest of the network. These two super-nodes contain the optimal solution, while the rest of the super-nodes consists of homogeneous nodes with low influence. For the *Community detection* task as well, we see that two super-nodes in the summary highlight the homogeneous regions well (which match closely with the ground truth communities in the network).

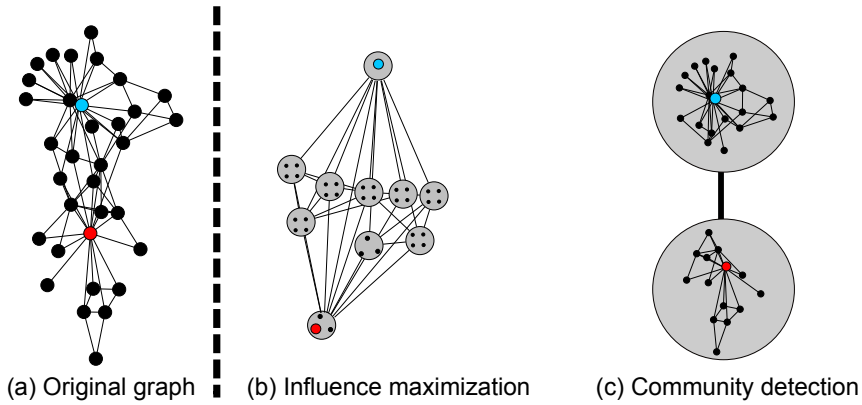


Figure 6.7: KARATE CLUB. (a) Original graph (b and c) Summary graphs of two GOP tasks. The red and blue nodes are the leaders of the two communities in the karate club.

6.5.4 Q2. Detecting anomalies on mobility graphs

Here we leverage NetGist for another application of discovering epidemiologically relevant anomalies in the WORK-PLACE networks, which are based on a mobility log of employees in a company with five departments. The dataset has been studied before for vaccination problems [64]. Due to variation in organizational roles and mobility patterns, some nodes may play anomalous roles in any epidemic outbreak. Such nodes are difficult to mine due to the temporal nature of data and as different nodes could be anomalous at different times.

Our main idea here is to leverage the inherent uncertainty in the data (the fact that any two people interacting at a given time is probabilistic) to learn the summarization process specific to the epidemiologically related influence maximization task. Once trained, we summarize the contact networks generated from the mobility log by applying the learned policy. Finally, we use the summaries to identify the anomalous nodes. Note that independently summarizing each network snapshot will not be useful as it discards the relationship between the snapshots (as we observe in our experiments as well).

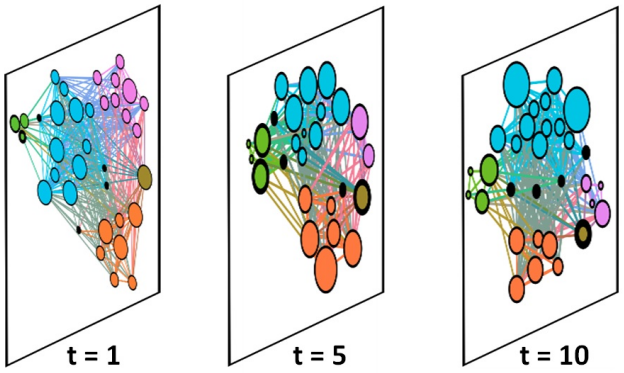


Figure 6.8: Summary graph of three snapshots of WORK-PLACE data. Black nodes are anomalies.

NetGist runs in less than 0.5s per iteration. Left, we show the learnt summary networks for three different snapshots. Color represents different departments and the size of the 'super-nodes' is proportional to the number of nodes inside them. First note that all the departments are clearly visible in the summaries. This highlights that NetGist is able to capture the temporal stability of the departments in the dataset. Secondly, we take the singleton (unmerged) nodes connecting different departments as the anomalous nodes (shown in black). Interestingly the nodes we identify as anomalous (80, 131, 751, 222, 134) have also been labelled as the 'linkers' in [64]. We consistently find some of the linkers in most time-stamps highlighting the stability of linkers over time as discussed in [64]. On the other hand, we also find different nodes as linkers in different timestamps highlighting that different nodes play anomalous roles at different times. Hence, the visualization of the summaries helps in sense-making of the anomalous nodes and the role they play, which is not obvious

from the original networks.

6.5.5 Q3. Robustness of NetGist

Here we evaluate the robustness of NetGist. First, we evaluate the quality of NetGist with changing reduction factors α. We summarized the graph using NetGist for various values of α and computed ρ for both of our GOP tasks. The results are shown in Figure 6.9b. As we can see, the performance of NetGist is robust across various values of α. The high value of ρ indicates that we obtain high-quality summary even when the network size is reduced by up to 90%.

Also, we evaluate the quality of NetGist for various parameter sets Θ. Figure 6.9a shows the results. They show that NetGist method is robust to the change of the parameters Θ of the GOP problem.

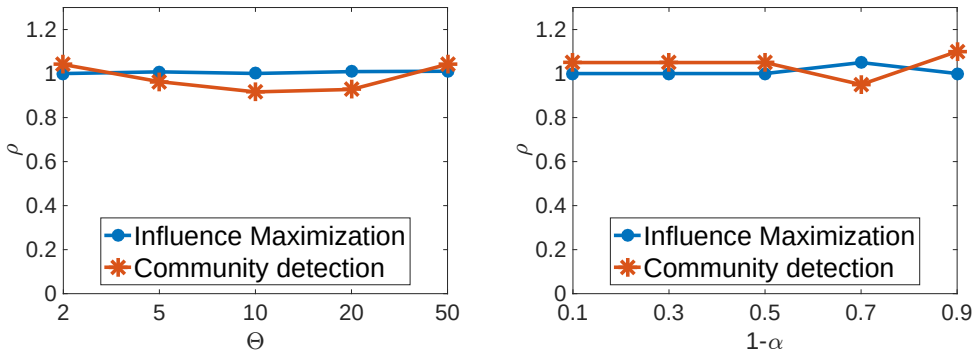


Figure 6.9: Quality of NetGist with respect to (a) the reduction factor $1-\alpha$ and (b) the parameter set Θ in HIGH-SCHOOL data. NetGist is robust across various values of α and Θ for both tasks.

6.5.6 Q4. Guided-NetGist helping StarSPIRE

Here we focus on showing how our principled approach Guided-NetGist learns to generate high-quality layouts for the StarSPIRE task (see Algorithm 14). We show that Guided-NetGist generates high-quality summaries of document networks which can help StarSPIRE to have more effective visualization. The network summary generated by Guided-NetGist gives better visualization for the working space that is closer to the user's expectations by visualizing more relevant documents with high Υ nearer to each other.

Setup: We use three popular document datasets in StarSPIRE database [31]: Crescent, VAST contest in 2007 and 2010. Document graphs are weighted networks. Nodes represent documents. TF-IDF vector of each document is its attribute vector. The edge weights represent the similarity between TF-IDF vectors of corresponding documents of end nodes. The human suggestions were selected from interactions observed during users studies and subset of the ground truth structure. Human suggestions are sparse as it is a common scenario in many real-world applications. Table 6.3 shows the datasets details. We compare the quality of Guided-NetGist with other summarization techniques which are listed below:

1. **NetGist** with no adjustments.
2. **Spectral** [158] partition a network for the community detection task based on weighted document network.
3. **Metric-Learning** [168] assigns different weight to each word of TF-IDF vector of the documents based on human suggestions and partition document network using their weighted vectors.

Table 6.3: Document networks details

Dataset	# Nodes	# Edges	# Node attributes	# Human suggestions
Crescent	41	820	1041	19
IEEE VAST contest in 2007	1473	1M	24K	25
IEEE VAST contest in 2010	103	5K	3K	17

For an approach Alg, we examine the quality κ_{Alg} of its generated summary G'_{Alg} by calculating the ratio of satisfied human suggestions.

Effectiveness: We investigate κ of the Guided-NetGist and baselines. Figure 6.10 shows that in *all* the networks the relative quality κ of Guided-NetGist (red) is significantly (i.e. 73%) higher than the baselines (grey). NetGist and Spectral methods cluster documents without considering human suggestions. Also, due to the sparsity of the human suggestions Metric-Learning does not perform well in learning document similarities and improving κ.

Visualization: We also visualize the document networks using Guided-NetGist. We use weighted force-directed layout [48] to visualize the summary graph (i.e. super-nodes and super-edges) and each subgraph inside the super-nodes. Then we "combine" these layouts in a multilevel fashion to get an associated visualization with the entire network (see Figure 6.11). Visualizing the networks by Guided-NetGist shows some qualitatively non-trivial aspects which helps StarSPIRE to highlight important characteristics of the networks (see Figure 6.1 in introduction, Figure 6.12 and Figure 6.13).

Crescent: Figure 6.1b shows the visualization of the Crescent data generated by Guided-NetGist. Crescent contains synthetic intelligence reports related to terrorist activities. It

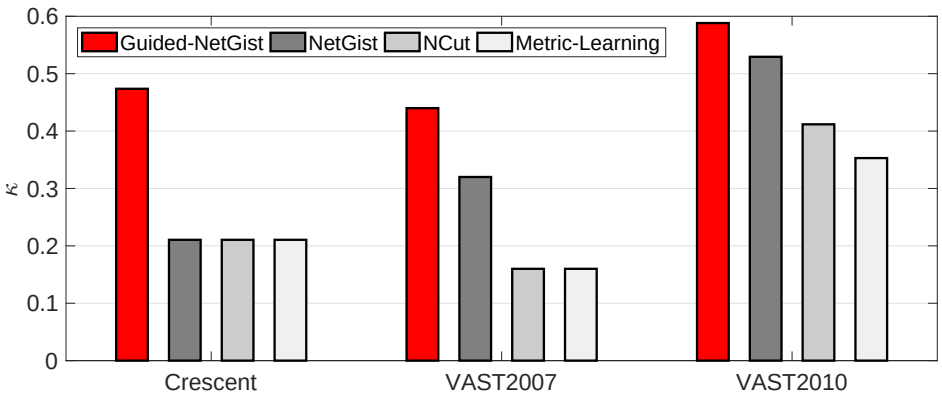


Figure 6.10: The quality κ of Guided-NetGist and baselines: Guided-NetGist (red) improves κ by 73% on average. NetGist and Spectral methods do not take the human suggestions into account. Metric-Learning performs poorly due to the sparsity of the human suggestions.

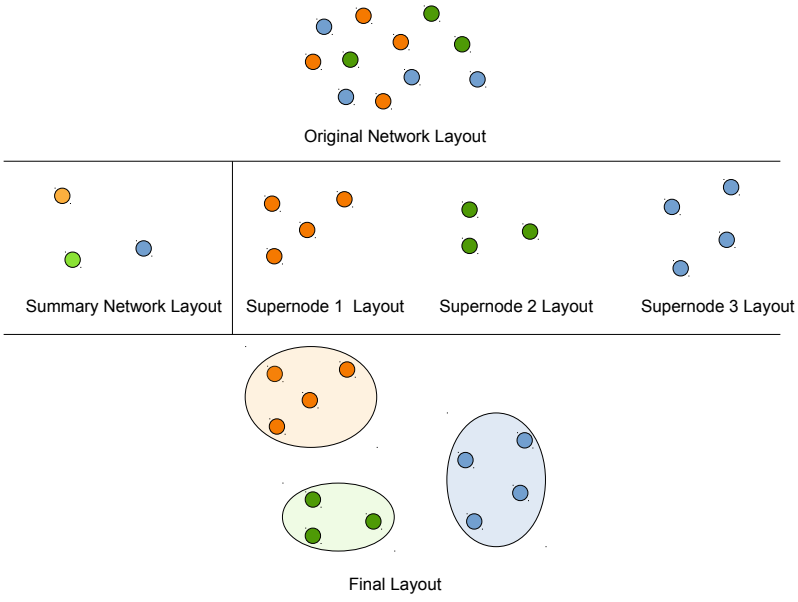


Figure 6.11: Multi-level scheme to generate layouts for summary networks: We generate weighted force-directed layouts for the summary network and each subgraph inside the supernodes. Then we combine them to get the final layout. Note, nodes this the same colors are in the same supernode.

contains information related to three coordinated terrorist actions in Boston, Atlanta, and New York Stock Exchange. Guided-NetGist visualization in Figure 6.1b reflects the user's interests by satisfying more of its suggestions. In Figure 6.1b, it is easy to identify different clusters of relevant documents. According to the ground-truth, the documents in the purple supernode are related to incidents in Boston. Also, documents in the dark green supernode are related to Atlanta, and the green supernode contains documents related to New York Stock Exchange incidents.

VAST contest 2007: Figure 6.12 shows the visualization of the VAST contest 2007 data generate by Guided-NetGist. This dataset consists of news stories and blog entries about exotic animal trafficking. Guided-NetGist can clearly distinguish three main stories among the documents: (1) chinchillas fur trading and demands for chinchillas, (2) animal rights activists reports, and (3) Monkeypox disease. As shown in Figure 6.12 the summary graph makes it possible for the users to limit their searching space by only exploring the documents in related supernode and not expanding the rest of the supernodes.

VAST contest 2010: Similar to the previous networks we can quickly identify main groups of related documents using the Guided-NetGist visualization. (Figure 6.13).

Hence Guided-NetGist automatically learns to generate network summaries which help to generate higher-quality visualization for document networks than current state-of-the-art algorithms: it satisfies not only the human suggestions but also group related nodes together, which are critical to analyzing document networks.

6.5.7 Summary of observations

Our main experimental observations are:

- **Generating effective summaries:** Our framework NetGist effectively learns how to generate summaries that maximizes ρ in Equation 6.2. It consistently performs at least as well as the baselines designed explicitly for a task. Also, it is scalable enough to solve many practical problems. So, our summaries are useful to solve the given tasks.
- **Learning generalizable model:** The learned model in NetGist is generalizable to the unseen instances of D and has a low test error.
- **Incorporating human guidance:** We extended NetGist to incorporate human suggestions and help in generating high-quality visualizations for the StarSPIRE problem, reflecting on average 73% more of user feedback.
- **Detecting anomalies:** NetGist helps in complex sense-making and anomaly-detection in temporal networks by using summaries.
- **Robustness:** NetGist is robust to a wide range of parameters for the TBNS problem.

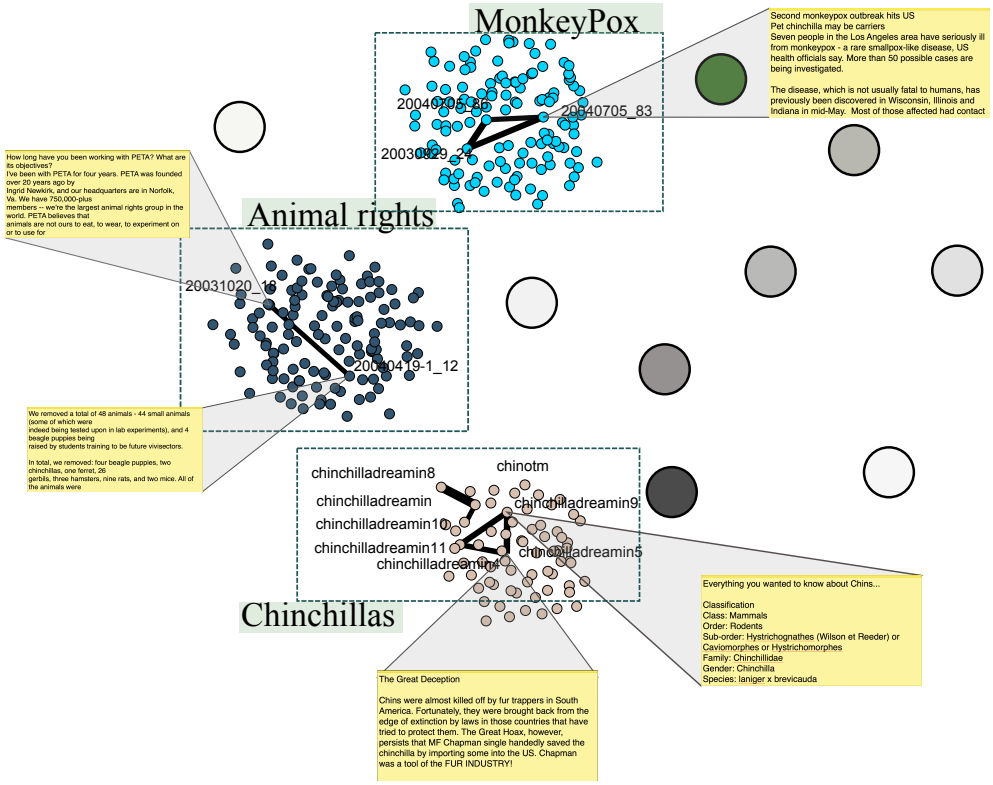


Figure 6.12: VAST contest 2007 visualization. Node colors represent their supernodes, and each rectangle represents a major subplot in the dataset. Note, only edges and node labels that are part of the human suggestions are shown in the visualization. Also, unrelated supernodes are not expanded.

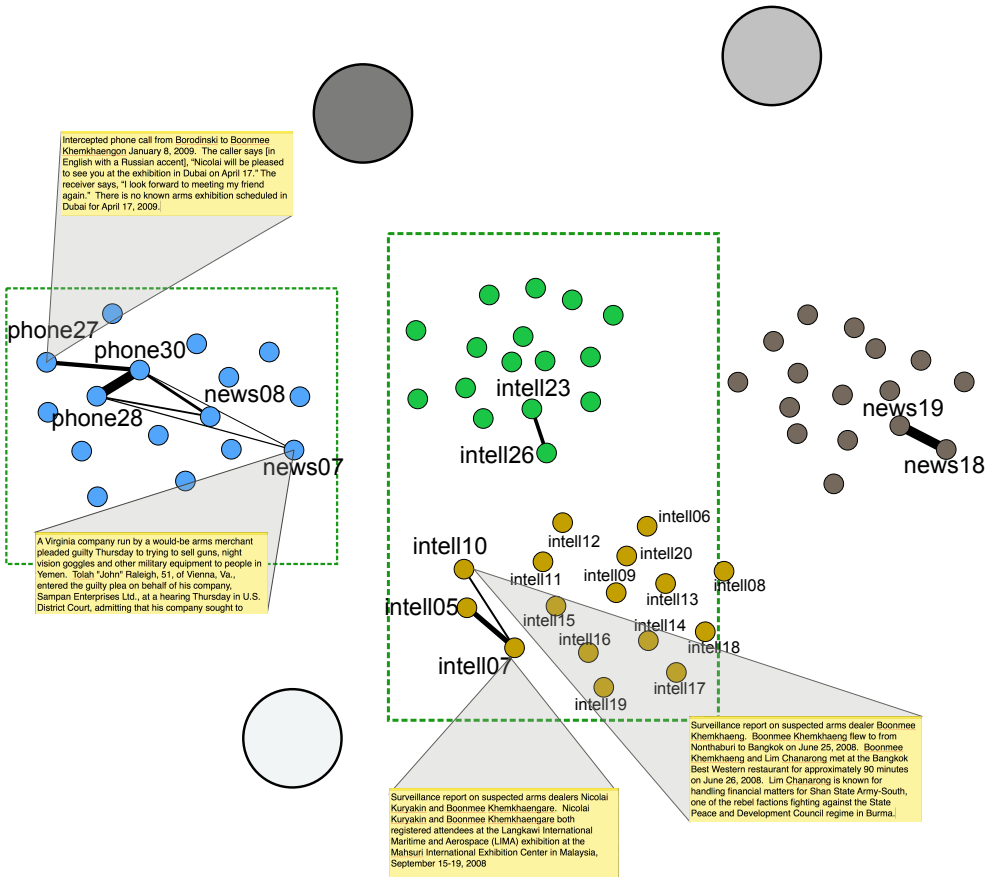


Figure 6.13: VAST contest 2010 visualization. Node colors represent their supernodes, and each rectangle represents a major subplot in the dataset. Note, only edges and node labels that are part of the human suggestions are shown in the visualization. Also, unrelated supernodes are not expanded.

6.6 Conclusions

We generalize over multiple threads of prior work and propose a novel Task-Based Network Summarization (TBNS) problem to automatically learn how to generate task specific network summaries. We proposed an effective method NetGist by leveraging the deep Q-learning framework. As shown by our experiments on both real and synthetic data, NetGist is able to learn meaningful summaries for various tasks like *Influence maximization* and *Community detection* even when we reduce the network size by 90%. Further, we show how we can use our framework to help develop practical new algorithms for challenging open problems (like GUIDED-TBNS) and also sense-making via visualization by proposing Guided-NetGist. Also, since Guided-NetGist is entirely automatic and reusable, it can generate summaries for multiple networks at the same time and use the learned model in other similar graphs. Hence, it has a significant impact in speeding up creating the high-quality visualization for multiple document datasets.

We believe our work opens several interesting avenues for future work. We can explore using the framework for even more tasks already in GOP and consider other more advanced approaches to aggregate human suggestions for the GUIDED-TBNS problem. Additionally, generalizing the GOP set of considered tasks (like to include graph alignment) and speeding up NetGist by leveraging deep network embedding would be fruitful. Finally, extending our framework to more complex notions of summaries which may include real-time interaction would also be interesting.

Chapter 7

Feedback-based summarization

In this chapter, we continue to investigate the learning-based summarization approaches on multivariate networks and focus on the visualization application of such network summarizations. Using human-in-the-loop techniques to generate high-quality visualizations for such datasets is a challenging and important area which has seen growing research work in recent years. For example, for when analyzing large text datasets, a user can express her agreement/disagreement with the visualization via iterative feedback to improve its quality. However, to incorporate such human feedback in generating a high-quality visualization, we should answer several challenging questions: How should we integrate human input with the original objective of the visualization? Sometimes human opinions can be inconsistent with the original objective of the visualization. How should we deal with such situations? Also, human feedback is typically sparse in the real-world, especially as the analysis task is usually exploratory, and the user only has a partial understanding of a final visualization. How should we generalize sparse user feedback?

In this chapter, we design NetReAct, a novel interactive network algorithm for text corpora that enables a more comprehensible visualization by a multilevel approach. NetReAct incorporates human feedback with reinforcement learning to summarize multivariate networks and visualize document corpora. It converts such datasets to a multivariate network structure and summarizes it by grouping relevant documents and laying them out in groups close to each other. We show how NetReAct can help in generating high-quality visualization and revealing hidden patterns and giving an in-depth understanding of documents. We also show it can detect related documents and group them better than the competitors' algorithms.

7.1 Introduction

Understanding large-scale structured and unstructured document corpora has many applications such as social media analysis, security, marketing, and anomaly detection. Visualizing

such datasets in a two-dimensional space is a common way of exploring them and understating their characteristics [116]. However, doing this for a large and complex document corpus is often challenging, and is sometimes an impossible task to do meaningfully on a medium-size screen such as regular desktop monitors. Using human-in-the-loop techniques to develop an interactive visualization can help in understanding and visualizing such datasets more accurately on such screens.

One of the state-of-the-art systems for interactive visual analytics for text data is StarSPIRE [31]. This system builds a multilevel model of user interests based on the user's semantic interactions. Using the inferences made from these interactions, the models identify the importance of each term in the documents and then calculates the weighted similarity between each pair of documents. However, this term vector is high-dimensional, and generating optimal weights requires a significant amount of interactions. Additionally, StarSPIRE builds this user model from scratch in each separate execution. As a result, the user must supply a large number of interactions to solve any task, even a task similar to a previously-solved task.

It is important to consider user interaction in generating visualizations which help in sense-making and guide the user to connect the dots between the documents. However, it is often challenging to do so as user inputs could be orthogonal to what the visualization tool is programmed to do. It is possible for a user to make a mistake or misunderstand the visualization task or the task does not quite match the user's intention. An example could be that the visualization tool may be developed for finding clusters of related documents while the user is trying to identify the important documents in the corpus. To handle this situation and consider both the user's interest and the goal of visualization, we should have a procedure to combine them effectively.

The possible discrepancy between the goal of the visualization tool and user input motivates the need to *learn* how to incorporate and generalize human feedback efficiently with the visualization goal/objective, as well as to generate a high-quality visualization that satisfies user's interests and helps in understanding the data. Learning these user interests makes it possible to *re-apply* the learned model in similar scenarios in order to reduce the amount of feedback required from the analyst.

We tackle the novel problem of learning a multivariate network-based, interactive visualization with the aim of investigating text data and supporting human sense-making process, i.e., help the users 'connect the dots' across different documents and discover the hidden stories behind them. More specifically we try to answer; Given a document corpus and an analysis task, can we *learn* a model that incorporates user feedback alongside the objectives of the analysis task from first principles? and can such a model be re-applied to other document corpora? Towards solving this problem, we face several challenges:

- *Required simplicity of feedback.* Generally, the users providing the feedback are not experts in the visualization models and cannot directly interact with the underlying

model of the system. We should be able to interpret their high-level semantic interaction into low-level instructions for the visualization models.

- *Sparsity and inconsistency of human feedback.* Getting human feedback is a slow and expensive process as the user needs to understand the data and task in hand. As a result, in real-world scenarios such feedback is sparse. Also, users may be inconsistent in their feedback. Thus, instead of merely relying on feedback, we should have a procedure to effectively incorporate other objectives of the given task to be able to visualize the data meaningfully.
- *Connecting dots between documents across the corpus.* Our goal is to help the user quickly read the most related documents to the hidden story behind them and navigate through the corpus. Therefore, we should attempt to define what the document groups should look like and automatically determine the relationships between different groups.

People often think of document collections as graphs [72]. Hence we want to represent the text corpora as a multivariate *network* data structure where each node in the network represents a document, and edge lengths indicate the similarity between their word usage. We then try to simplify the entire network down to a significant story-network and to organize it in a logical way on the screen. One can propose to simplify the network representation of text corpora by grouping the similar/relevant documents together to produce a high-level understanding of the entire corpus and highlight hidden patterns. Hence, we can model this approach as a multivariate network summarization problem, which consists of finding hierarchical "super-nodes" (representing collection of documents) and "super-edges" (representing similarities between groups of documents) and laying out the resulting smaller network based on the relatedness of the super-nodes/edges. Later on we discuss the concepts of super-nodes and super-edges in detail.

Our main idea is to tackle the above challenges using reinforcement learning (RL) approach to summarize multivariate document networks. We believe RL is especially suited for this problem as it makes it possible to re-apply the learned model in similar scenarios to reduce the necessary amount of human feedback to gain useful information. We design NetReAct, a feedback-based RL algorithm which integrates user interests with the objective of the visualization. Thus, this algorithm can be used in visual analytic tools. Some examples for such objectives are understanding the underlying story of the documents, detecting relevant documents to a given story, and highlighting the relationships between documents. NetReAct learns a model to summarize a network of super-nodes and super-edges and gives a high-level understanding of the entire text data. It also provides a multilevel understanding of the data by generating summaries of various sizes and enabling the user to zoom into each document group to see these different summaries. The main contributions of our work are:

- *Incorporating human feedback.* We introduce NetReAct, a novel approach which leverages feedback-based reinforcement learning to principally incorporate human feedback with the objective of the visualization task. NetReAct makes it possible to re-apply the

learned model to similar situations to reduce the amount of required feedback to generate a high-quality visualization. NetReAct is already tested on real-world networks with various sizes to meaningfully visualize them for real-world applications.

- *Meaningful relationships between groups as a summary network.* NetReAct not only groups relevant document into super-nodes, but it also defines the relationships between super-nodes. The length of the edges in the summary network by NetReAct represent the similarity between groups. This helps the user explore the entire corpus effectively by navigating through different super-nodes.
- *Multi-scale visualization.* We leveraged NetReAct to develop a multi-scale, interactive visualization. This visualization groups documents to summarize the corpus hierarchically. It makes it possible for the user to get a multilevel understanding of data by looking at the summaries on different levels. Depending on the task, users can choose how much detail they want to see about the dataset. Also, by only investigating documents in one of the super-nodes, users can locally study the data in more detail.

7.2 Proposed Method

Visualizing a document corpus helps users in sense-making and understanding the documents by providing a two-dimensional illustration of the entire corpus and highlighting the hidden patterns. Learning from the user is essential in designing an intelligent visualization framework which reflects the user's interests. Moreover, leveraging user feedback helps to visualize data more effectively and efficiently than unsupervised approaches. On the other hand, using supervised approaches are not realistic in many real-world applications, particularly when the analyst is not an expert in data visualization. Hence, understanding and incorporating the user's partial semantic feedback within a visualization in a practical way is essential when designing a high-quality visualization framework.

StarSPIRE [31] is a state-of-the-art interactive document visualization system that generates similarity-based layouts of document corpora. It generates a 2-D visualization of documents by combining user interests and the similarity of documents. It learns the user preference from its semantic interactions such as opening, minimizing, removing, annotating and highlighting documents. More specifically, it identifies the importance of each term in the TF-IDF [103] vector of documents to calculate the weighted similarity between them given the user's semantic interactions. For example, a user can express her interest in a document by opening and bookmarking or highlighting parts of it. In this case, StarSPIRE increases the weight of the terms in the highlighted text, updates the similarity measures to visualize new similar documents on screen, and recommends them to read. As a result, the user can find more relevant documents easier. However, the TF-IDF vector is high-dimensional, and generating the optimal weights requires a significant number of user semantic interactions. Also, for each usage, StarSPIRE builds the user model from scratch. As a result, it expects

a similar number of interactions even in similar repeated scenarios. Instead, we propose a new method which can learn from a low amount of human feedback and generalize them to generate high-quality visualizations and re-apply the learned model in similar scenarios.

7.2.1 Visualization with Summarization

To achieve the goal above, we summarize a document corpus by grouping related documents and demonstrating the relationship between groups in order to both generate a high-quality visualization for sense-making tasks and detect the underlying stories hidden within documents. We call such document summaries a summary network. We design NetReAct to make it possible to incorporate human feedback with network summarization, using that to generate a visualization of a document corpus. A "good" document network summary leads to a high-quality visualization, which helps a user to read related documents and make sense of them quickly. More specifically, in a good network summary, each super-node (i.e., group) contains documents that are most relevant to each other according to the users interest. Also, the structure of the network summary indicates the relation between different groups, which guides the user on how to navigate through different groups. If we have such a summary, first we can visualize the summary network. After this, we can expand and visualize the super-nodes that the user is interested in to suggest the most relevant documents. Then, if the user wants to investigate more documents, we can expand the closest/ most similar super-nodes to the current one to suggest the next most relevant group of documents to the user's interests. The overarching idea is to generate such a network summary with these characteristics.

7.2.2 User Feedback

As mentioned earlier, in the StarSPIRE framework, a user can interact with the system through semantic interactions such as minimizing document, closing document, annotation, highlight and overlap document We observe the user while completing the sense-making task using the StarSPIRE framework, and use a subset of these interactions for generating user feedback. We divide such interactions into *positive* and *negative* feedback (See Table 7.1). Positive feedback indicates the user's intention to put two documents close to each other and negative feedback means they should be far from each other. For example, overlapping two documents indicates that the user agrees to display them close to each other. On the other hand, minimizing a document while reading another one is a sign of the disagreement with the visualization. Such feedback is sparse, as the user cannot evaluate all documents and every aspect of the visualization. Hence, it is crucial to incorporate user feedback with the visualization framework effectively.

Table 7.1: NetReAct feedback types and corresponding semantic interactions in StarSPIRE.

Feedback Type	**Semantic Interaction**
Negative feedback	(1) Minimizing document, (2) Closing document
Positive feedback	(1) Annotation, (2) Highlighting document, (3) Overlapping document

7.2.3 Interactive Summarization Model

In this section, we design an interactive network summarization framework to incorporate the user feedback and address its sparsity. Note that, our goal is to learn the steps to be taken for the summarization process so that the same approach can be re-applied on other document corpora with similar characteristics.

Network Construction. We start by converting the given document corpus into a network $G(V, E, W)$ where nodes (i.e., V) represent documents and edges (E) and their weights (W) represent the similarity between documents. We define the weight $w(v_1, v_2)$ between nodes v_1 and v_2 to be the cosine similarity between their corresponding TF-IDF vectors. Note that G is a complete-graph, a clique, of size $|V|$.

Network Summarization. Once the network G is constructed, we begin the summarization process. Our idea is it to generate a smaller network $G^s(V^s, E^s, W^s)$ from the original network $G(V, E, W)$ such that nodes representing similar/relevant documents in G are grouped into a single node (a 'super-node') in G^s. Note that nodes in G^s now represent a group of documents. We call $G^s(V^s, E^s, W^s)$ a summary network, where super-nodes (V^s) are the groups of related documents and super-edges (E^s) and their weights (W^s) represent the average similarity between group of documents represented by the two end-points.

The way we obtain G^s is via a series of 'assign' operation on G. The 'assign' operation assigns nodes to their super-nodes. Note that, this operation partitions the original network G and groups each partition to form a super node in the summary network G^s. Now an obvious question is how to partition the original network G in a meaningful manner? How to decide between two partitions of the G? And how to measure the quality of each assigning operation and network summary?

Reinforcement Learning (RL)—more specifically the Q-learning [150]—is a natural solution to the above questions. In general, RL methods learn to take an '*action*' in an '*environment*' to maximize cumulative '*reward*' based on a '*transition*' function. RL is a natural fit in our case as we can view our problem as taking actions (assigning nodes to a super-node) to maximize the reward (the final quality of grouping of documents). The next step in our summarization process is to formalize the Reinforcement learning framework for our task.

Interactive Reinforcement Learning Formulation

Here we use Q-learning as it is known to be more sample efficient than policy gradient methods [162]. Each RL framework has five main components: states, actions, transition function, reward, and policy. We also, add a new feedback component to it as we design an interactive RL formulation. Brief description of each follow.

1. State: The state s is the sequence of actions which assign nodes to different super-nodes. We use the embedding in Eq. 7.1 to represent the states in a vector of n-dimensional space.

$$s = [l_1, l_2, \ldots, l_n] \qquad \forall_{i \in \{1,2,\ldots,n\}} \quad l_i \in \{1, 2, \ldots, |V_s|\} \tag{7.1}$$

2. Action: An action a at state s_t selects a document i and assigns it to a new super-node v^s and transfer it to the next state s_{t+1}.

3. Transition: We assume the transition function $\mathcal{T}$ is deterministic and corresponds to the action of assigning a document to a new super-node – i.e., $\mathcal{T}(s_t, a) = s_{t+1}$.

4. Rewards: we define the reward to be -1 for a state s, unless it is a terminal state. A terminal state in our case is a state which satisfies all the positive and negative feedback of the user. Intuitively the reward of -1 encourages the learner to satisfy the user feedback faster. Formally, we define our reward function as follows:

$$r(s, a) = \begin{cases} F_{prob}(s_{next}) = \sum_{i=1}^{k} \frac{y_i^T A y_i}{y_i^T D y_i} & \text{if } s_{next} \text{ is a terminal state} \\ -1 & \text{otherwise} \end{cases} \tag{7.2}$$

Where A is the adjacency matrix of the document graph G, D is the diagonal matrix of node degrees and y_i is an indicator vector for super-node $v_i^s \in V^s$, i.e., $y_i(v) = 1$ if a node v belongs to super-node v_i^s, zero otherwise. In Eq. 7.2 we compute $F_{prob}(s_{next})$ measures the quality of the document groups. By maximizing $F_{prob}(s_{next})$ we maximize the quality of document groups [166]. Note that not having a predefined reward function (i.e., -1) would essentially require us to solve the task at hand at each state, which would be a costly process. Also, -1 reward for the non-terminal states encourages the learner to reach the terminal state faster and converges quickly.

5. Feedback We assume a case that the user is interacting with the system until she is happy with the visualization, a processing of incrementally formalizing the space [146]. It means that we must learn the model until all the feedback from the user is satisfied. The feedback is in the form of positive and negative interactions (See Tab. 7.1) The user can indicate if she agrees to group a pair of document (i.e., positive feedback) or disagrees with it (i.e., negative feedback). We represent the feedback with two graphs which we name the feedback graphs. A positive feedback graph G^+ is created from the set of positive feedback, i.e., the edges in G^+ are pair of relevant documents that the user indicated. Similarly, the negative feedback graph G^- is created from the set of negative feedback.

To satisfy all the user feedback, we must group all document pairs in positive feedback in the same super-nodes and all document pairs in negative feedback in different super-nodes. These constraints can now be stated using the positive and negative feedback graphs G^+ and G^-. More formally

$$\sum_{i=1}^{k} y_i^T A_{G^+} y_i - \sum_{i=1}^{k} y_i^T A_{G^-} y_i = \sum A_{G^+} \tag{7.3}$$

where y_i is each of the document super-nodes, k is the number of super-nodes and A_G is the adjacency matrix of a graph G. In the real world, we can not expect the user to provide all possible feedback and basically providing the desired summary. Usually, the provided feedback are sparse especially when the task is more exploratory. To handle such problems we combine the original reward in Eq. 7.2 with feedback (Eq. 7.3). So, our goal is to achieve a summary which satisfies all the feedback and maximizes the cumulative reward.

6. Policy: The policy function $\pi(a^*|s)$ specifies what action to take at a given state. It is defined as $\pi(a^*|s) = \arg\max \text{Q-value}(s, a)$ where Q-value(s, a) is the Q-value of the state s and action a which estimates the expected cumulative reward we achieve after taking action a at state s [150]. Our goal is to learn optimal Q-value function results in the highest cumulative reward. We leverage the Q-learning algorithm [150] which iteratively updates Q-value(s, a) until convergence.

Q-learning

Our pipeline learns the best super-node for each node in the document graph G such that its corresponding summary graph G^s gives a high-quality visualization and generalizable to similar unseen document corpus. We use Q-learning to learn the pipeline. First we define how to estimate the Q-value of a state s and action a, Q-value(s, a). We define Q-value of a state and action as the expected rewards in future as follows,

$$\text{Q-value}(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] \tag{7.4}$$

Our aim is to find the maximum available cumulative reward achievable with state s and action a. That is Q-value$^*(s, a) = \max \text{Q-value}(s, a)$. We estimate Q-value$^*(s, a)$ iteratively using Bellman equation as follows,

$$\text{Q-value}_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} \text{Q-value}_i(s', a') | s, a\right] \tag{7.5}$$

We use a fully connected neural network to embed each state s (Eq. 7.1) and get a compact representation of it. We combine the embedding layer with Q-value$^*(s, a)$ estimator to have an end-to-end framework to summarize the document network.

In our framework the input state s (Eq. 7.1) is fed into Fully Connected (FC) neural network. The outputs of this step is a compact representation of the state, which is then fed into a another FC which decides how to update the super-nodes. Alg. 15 shows an overview of our summarization algorithm.

Algorithm 15 Summarization

Require: G, G^+, G^-, k
1: Randomly Initialize the deep Q-learning parameters
2: // learning how to summarize
3: **for** episode=1 **to** T **do**
4: Initialize s_0 : Initialize $\{y_1, y_2, \ldots, y_k\}$
5: **while** Feedback-value $< k$ **do**
6: // Take a action
7: $a^* = \arg\max$ Q-value$(s_{current}, a)$
8: $s_{current} = \mathcal{T}(s_{current}, a^*)$
9: // Evaluate
10: Evaluate the corresponding partitioning to $s_{current}$ (Eq. 7.2)
11: // Optimize
12: Update the deep Q-learning parameters to yield better summary (see Section 7.2.3)
13: **Return** the trained model and super-nodes $\{y_1, y_2, \ldots, y_k\}$

Hierarchical Approach

Our goal is not only summarizing the document data but giving a multilevel understanding of it as well. Also, in large document corpora, it is challenging to meaningfully and efficiently generate the best summary of it. Hence, we propose a hierarchical approach to summarize the data and generate summaries with different sizes. Alg. 16 shows our hierarchical summarization approach. In each step it tries to summaries the data into two super-nodes and the iteratively summaries each part until reaching a summary with the desired size.

Generate Summaries

After learning the best super-nodes of the document network, we *merge* nodes in the same super-nodes to generate a corresponding super-node of it. Also, we connect each super-node to others by super-edges. The weight of the super-edge from y_1 to y_2 is the average similarity between documents in y_1 to y_2. More formally we define the merge operation as follows,

Definition 23 *(Merge) Merge operation merges nodes v_1, v_2, ..., v_b into a new node y, such that $\forall_{j=1,\ldots b}\ v_j \in V$. We add new edge (y, i) for all the nodes $i \in \bigcup_{j=1\ldots,b} NB(v_j)$ with*

Algorithm 16 Hierarchical-summarization

Require: G, G^+, G^-, k, $current_k$
1: y_1, y_2 = Partitioning(G, G^+, G^-, 2)
2: **if** $2 \times current_k \leq k$ **then**
3: $G_1, G_1^+, G_1^- \longleftarrow$ sub-graph and feedback corresponding to y_1
4: $G_2, G_2^+, G_2^- \longleftarrow$ sub-graph and feedback corresponding to y_2
5: $y_{1,1},\ldots y_{1,\frac{k}{2}}$ = Partitioning(G_1, G_1^+, G_1^-, k, $2 \times current_k$)
6: $y_{2,1},\ldots y_{2,\frac{k}{2}}$ = Partitioning(G_1, G_1^+, G_1^-, k, $2 \times current_k$)
7: $y_1, \ldots y_k \longleftarrow$ combine $y_{1,1},\ldots y_{1,\frac{k}{2}}$ and $y_{2,1},\ldots y_{2,\frac{k}{2}}$
8: **Return** $y_1, \ldots y_k$

weight $\frac{\sum_{j=1}^{b} W(v_j,i)}{b}$.

We merge nodes in the same super-node using def. 23 to get a summary document network $G^s(V^s, E^s, W^s)$ where $|V^s| = k$.

7.2.4 Two-step Visualization

Once the summary is generated, our goal is to visualize the document network. The visualization is challenging as our summary network involves visualizing super-nodes. Hence, we design a multilevel framework for visualizing the summary network. We first leverage the weighted force-directed layout [48] to visualize the summary graph. This gives us 2-d layout of the summary network, which we treat as the 'backbone' of our visualization process.

Now, note that each super-nodes consists of group of nodes, which induce a sub-graph in the original network. We separately, run the weighted force-directed layout on each sub-graph induced by the super-nodes. Finally, we "combine" the layouts within each super-node with the backbone layout (of the entire summary network) in a multilevel fashion to visualize with entire network. lines 3-7 of Alg. 17 shows the pseudo-code of our two-step visualization approach.

7.2.5 Overview of the Algorithm

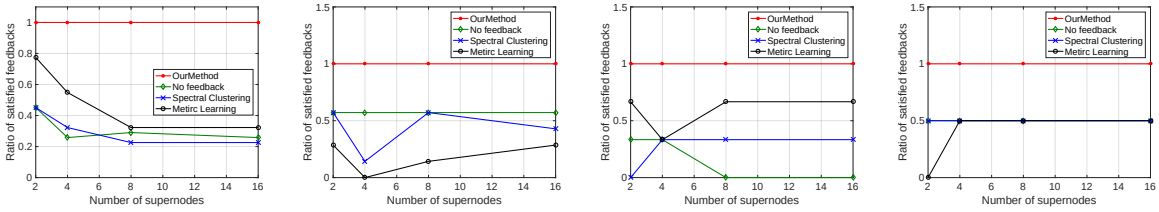
Alg. 17 gives an overview of NetReAct framework. It gets the document networks, user feedback and the number of supernodes k as input. First, it hierarchically partitions the network (line 1). Next, it generates the best summary which satisfies user's feedback (line 2). Finally, it generate the final visualization with the two-step weighted force-directed layout approach (lines 3 - 5).

Algorithm 17 Overview of NetReAct

Require: G, G^+, G^-, k, $current_k$

1: $y_1, \dots y_k$ = Hierarchical-Partitioning$(G, G^+, G^-, k, 1)$
2: $G^s \longleftarrow$ merge nodes in $y_1, \dots y_k$
3: $\forall_{1 \leq i \leq k} loc_{y_i} \longleftarrow Layout(G^s)$
4: **for** super-node $y_i \in \{y_1, \dots, y_k\}$ **do**
5: $\quad G_i \longleftarrow$ Corresponding sub-graph of y_i
6: $\quad \forall_{v \in y_i} loc_v \longleftarrow Layout(G_i)$
7: $\quad \forall_{v \in y_i} loc_v = \frac{loc_v}{k} + loc_{y+1}$
8: **Return** $\forall_{v \in V} loc_v$

7.3 Empirical Study



(a) Crescent (b) Chinchilla subplot (c) Bert subplot (d) Circus subplot

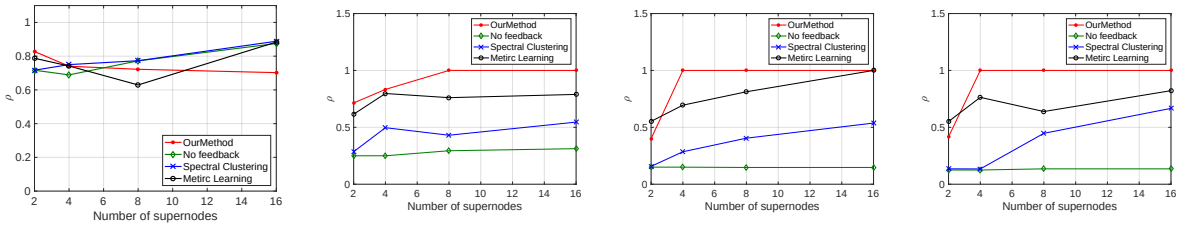
Figure 7.1: Ration of satisfied feedback in (a) Crescent and (b-d) different subplots of VAST 2007 dataset. Note, NetReAct satisfies all the user feedback while other baselines do not.

We design various experiments to evaluate the performance of NetReAct. Specifically, we answer the following questions:

- **Q1.** Does NetReAct learn to generate high-quality summaries?
- **Q2.** What is the effect of feedback on NetReAct?
- **Q3.** Does NetReAct give high-quality document visualization?

We used Python and PyTorch to implement all the steps of NetReAct, and our code is publicly available for academic/research purposes[1]. We implement deep Q-learning on GPUs, which makes it suitable for summarizing and visualizing document networks.

[1]http://github.com/SorourAmiri/NetReAct



(a) CRESCENT (b) Chinchilla subplot (c) Bert subplot (d) Circus subplot

Figure 7.2: Quality of summaries in (a) CRESCENT and (b-d) different subplots of VAST 2007 dataset. Note, NetReAct generates network summaries with the highest ρ.

7.3.1 Datasets

We explore the effectiveness of NetReAct on two well-known document datasets. Brief description of each dataset is as follows:

CRESCENT [81] is a dataset containing synthetic intelligence reports related to terrorist activities. It includes 41 synthetic intelligence reports about three coordinated terrorist plots in Boston, Atlanta, and the New York Stock Exchange. Each plot involves at least four suspects. Twenty-four of these reports are relevant to the scenarios, and the rest are irrelevant.

VAST 2007 Challenge dataset (Blue Iguanodon) [70] contains documents, images, and spreadsheets. We extract only the text, as our method is designed for visualizing document datasets. It includes around 1,500 text files. There are 26 documents relevant to the final solution. These documents belong to four major subplots related to unexpected activities concerning wildlife law enforcement: (1) Chinchilla Bio-terror (2) Bert (3) Circus and (4) Fish. The rest of the documents are irrelevant.

7.3.2 Baselines

We compare the performance of NetReAct with three competitors which are listed below.

FeedbackFree uses NetReAct without any user feedback.

Spectral [158] partitions the document network G while minimizing its normalized cut value. The partitions can be an input to our summarization (i.e., Sec. 7.2.3) and the two-step visualization approach (i.e., Sec. 7.2.4) to visualize the document corpus.

Metric-Learning [168] learns the importance of each word in the TF-IDF vector of the documents based on human feedback and assigns a weight to them. It measures the similarity between documents by calculating the similarity between the weighted TF-IDF vectors which mimics the StarSPIRE approach in updating the similarity between documents. Next, it

updates the edge weight W in the document network G using the updated similarities. Finally, this method partitions the updated document network using Spectral and uses the partitions to visualize the document corpus.

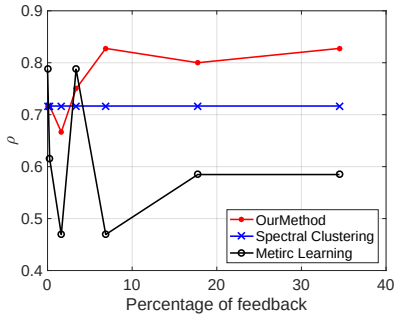


Figure 7.3: Value of ρ with various amount of user feedback in CRESCENT dataset

7.3.3 Q1. Quality of Summaries

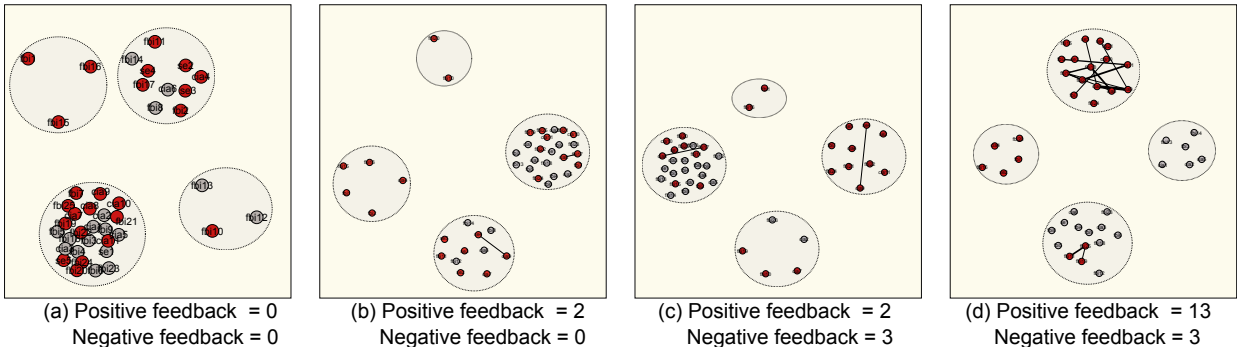
Here we demonstrate that NetReAct can effectively separate the relevant documents and irrelevant ones, thus helping analysts and users identify necessary documents to read.

In order to measure the ease of identifying the relevant documents, we measure the purity of super-nodes that contain relevant documents. In other words, we calculate the average probability of observing a relevant document in a super-node that contains at least one relevant document. Formally,

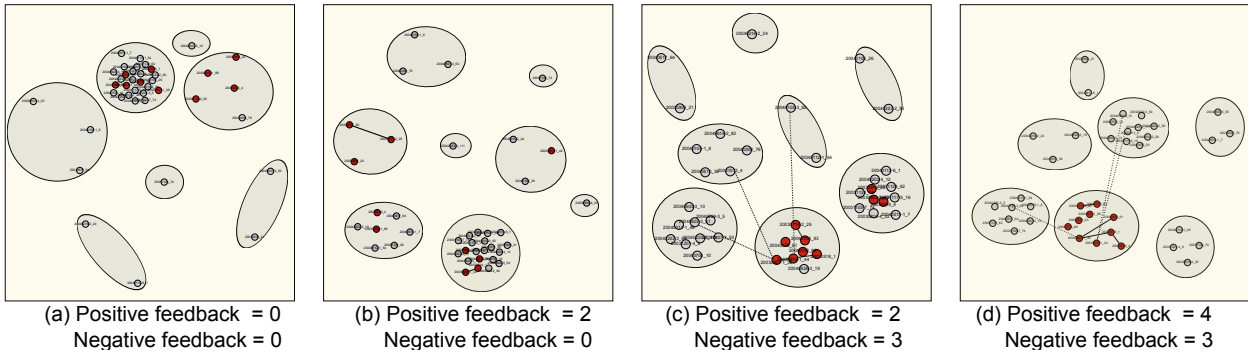
$$\rho = \frac{1}{|V_r^s|} \sum_{v^s \in V_r^s} Pr(doc = relevant|v^s) \tag{7.6}$$

where V_r^s is the set of super-nodes that have at least one relevant document to the scenario, v^s is a super-node in the set V_r^s and $Pr(doc = relevant|v^s)$ is the probability that a document is relevant to the hidden scenario in v^s. Intuitively, if the value of ρ is closer to one, it means the user can easily find relevant documents in a selected super-node.

We investigate the quality of summary networks generated by NetReAct using CRESCENT and three subplots of VAST 2007 datasets with 2, 4, 8, and 16 super-nodes and calculate their ρ values (Eq. 7.6). In addition, we compare the quality of NetReAct with baselines. Figs. 7.1 and 7.2 show the quality of summary networks with various number of super-nodes. For each experiment, we randomly selected positive and negative feedbacks from the ground-truth (See 7.2.2). More specifically, we randomly choose a few pairs of nodes that are relevant to the hidden story as positive feedback and similarly pick pairs that only one of them is relevant as negative feedback. Note that in all the experiments we fixed the amount of positive and negative feedback: 10% of all possible positive feedback and 1% of all possible



(a) Crescent



(b) Chinchilla subplot

Figure 7.4: The visualization evolves with user feedback. (a) Shows the changes in the visualization of the Crescent data and (b) shows the visualization of the Chinchilla subplot of VAST 2007 dataset. Note the black lines represent positive feedback and dashed lines represent negative feedback. Also, red nodes represent relevant documents to the scenario and the gray ones are irrelevant.

negative feedback. In all figures, we depict positive feedbacks as solid black lines between documents and negative feedbacks as dashed lines.

Fig. 7.1 shows the ratio of satisfied feedback. The results indicate that NetReAct generates the highest-quality summary that matches the interests of the user, as it satisfies all of the user's feedback while other baselines can only satisfy part of the feedback. Fig. 7.2 shows the quality of summaries ρ. NetReAct generates high-quality summary (i.e highest ρ) networks for various sizes. This in turn means that users can easily find the relevant documents to scenarios using NetReAct. It is interesting to mention that because feedback is sparse and the TF-IDF vectors are high-dimensional, Metric-Learning approach is not able to learn proper weights and does not perform well in some cases.

7.3.4 Q2. Effect of Feedback

We investigate how NetReAct evolves the visualization of a document corpus while the user gradually gives positive and negative feedback. Also, we objectively measure the change in quality of super-nodes based on the amount of feedback by tracking the changes in ρ. Note, similar to Sec. 7.3.3 feedbacks are randomly generated from the ground-truth.

Usage Scenario

NetReAct initializes the visualization with no user feedback. At first, only the summary network (i.e., super-nodes and super-edges) is displayed on the screen. Also, for each super-node, the word-cloud of its documents is shown as a reference for the user to have a brief overview of the word usage of the documents. Based on the word-cloud and the layout of the summary network, the user selects a super-node to investigate and starts reading the documents in that super-node. While the user is reading documents, she gives feedback if she agrees or disagrees with the visualization. For example, the user provides negative feedback if there is a pair of dissimilar documents in the same super-node and positive feedback if a couple of similar documents are in different super-nodes. We use such inputs to update our feedback-based reinforcement learning model and consequently update the visualization. The visualization is updated until the user is satisfied with the result and does not have any more feedback.

Crescent: Fig. 7.4a shows how visualization evolves with user feedback. The user works on the Crescent data to identify documents that are related to suspicious activities and eventually detects documents related to terrorist attacks. First (Fig. 7.4a(a)), NetReAct gives the initial visualization of the Crescent data. The user starts her work by selecting one of the super-nodes randomly or by looking at the word-cloud. The selected super-node gets expanded, and the user can further investigate the documents inside it. The visual encoding, such as the location of documents inside the same super-node, position of other super-nodes, and their word-cloud, guides the choice of next documents/super-nodes to investigate. For example, in 7.4a(a) the user selects 'cia9' report first in the largest super-node of the summary network. Next, she decides to read 'cia8' the nearest report to it on the screen. Next, she goes further to documents in another super-node on the top right of the screen. First, she selects, 'se4' and its nearest document 'fbi17' and realizes that they are related to the documents in the previous super-node. So, she gives two positive feedback to indicate they should be in the same super-node. After such interactions, NetReAct updates the model and the visualization consequently. Fig. 7.4a(b) shows the updated visualization after user input. The user continues working on the updated visualization. The new visualization has better quality, as it has two super-nodes with only relevant documents. However, the quality of the two other super-nodes should be improved. She selects 'se4' again to investigate its super-node and realizes the three of its nearest documents (i.e. 'fbi8', 'fbi14', and 'cia6') are not related to it. So she provides three negative feedback actions. Each new feedback helps

NetReAct to learn the model that puts relevant nodes in the same super-nodes (See 7.4a(c)). Once the user iteratively specifies positive and negative feedback, NetReAct infers which other unseen documents are similar and updates the summary network and the visualization. In the CRESCENT dataset, NetReAct generates a high-quality summary with high ρ (Eq. 7.6) after the user reads around 20 documents.

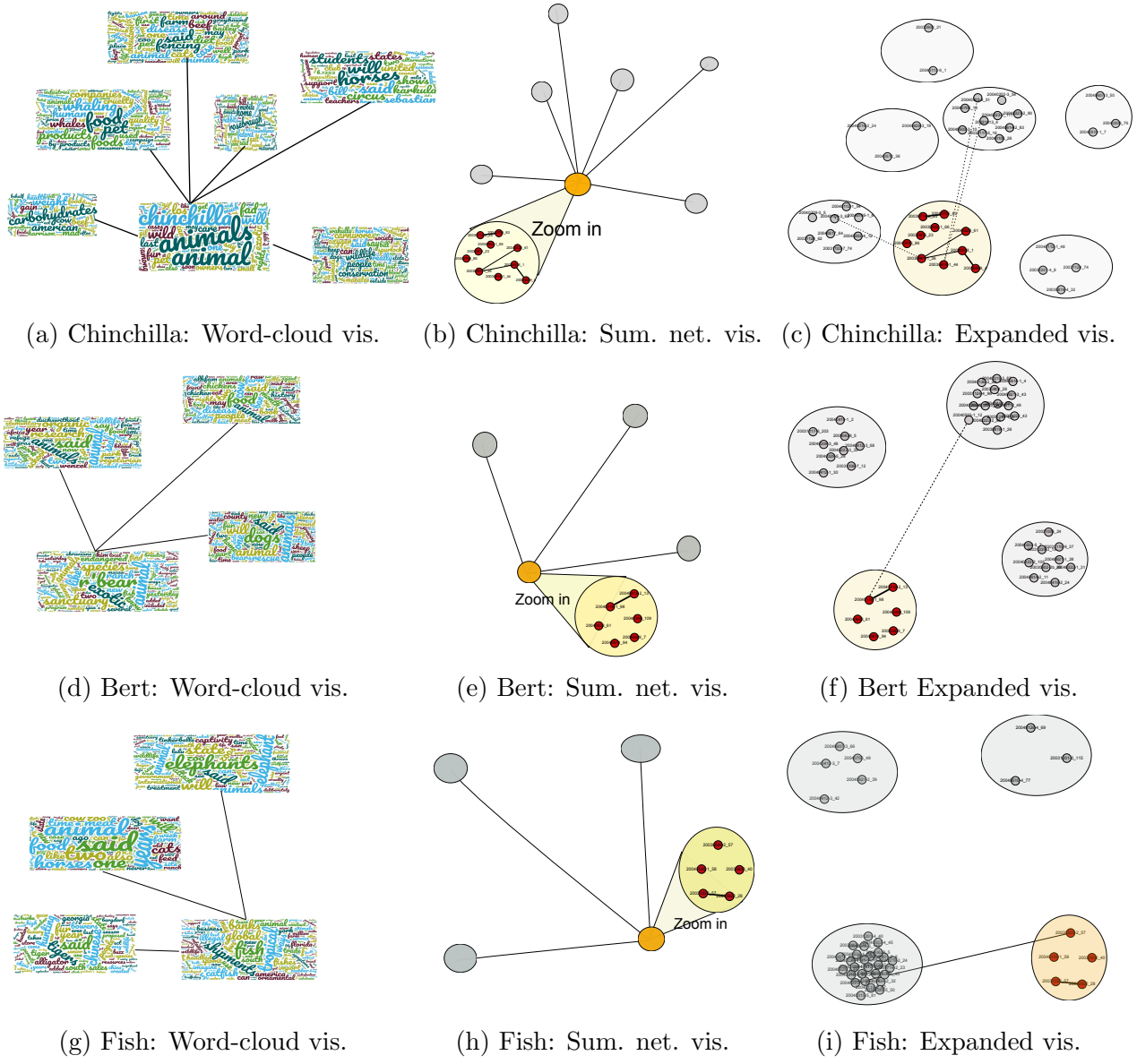
VAST 2007: To showcase the quality of NetReAct on the VAST 2007 dataset, for each subplot we extract a subset of documents relevant to the subplot, as well as a number of other randomly selected documents from the original corpus. Fig. 7.4b shows the visualization of the Chinchilla Bio-terror subplot. Initially our method can only identify four of the related documents with the subplot and puts them in a super-node. However, the rest of the relevant documents are mixed with other irrelevant ones in the largest super-node of the summary network. Next, the user gives feedback regarding the similarity of two pairs of documents. NetReAct updates the visualization (Fig 7.4b(b)). However, this is not enough to improve the quality. When the user adds the negative feedback, NetReAct can distinguish more relevant documents (Fig 7.4b(c)). Finally, by giving two more positive feedback NetReAct can accurately identify the relevant documents with the subplots and puts them in a separate super-node (Fig 7.4b(d)).

We investigate the effect of the amount of given positive and negative feedback on the quality of summaries generated by NetReAct and other baselines. We calculate the ρ values with randomly generated feedback from the ground-truth. Fig. 7.3 show the results for the CRESCENT dataset with NetReAct, Spectral, and Metric-Learning methods. The results shows that NetReAct consistently outperforms the baselines. Further, NetReAct only needs 6% of possible feedback, which is around 20 pairs, to produce a high-quality summary network.

7.3.5 Q3. Quality of Document Visualizations

We further examine the quality of visualizations generated by NetReAct by running multiple case studies on the CRESCENT and VAST 2007 datasets with various settings. We show that visualizations generated by the hierarchical approach of NetReAct reveals interesting patterns in document networks and gives an in-depth understanding of them. As a result, it helps users to analyze document datasets more effectively. Finally, we showcase the result of re-applying the learned model for one of the subplots of the VAST 2007 data on a different subplot to show the reusability applications of NetReAct.

Fig. 7.8, and Fig. 7.5 are the final visualization of the CRESCENT dataset and the Chinchilla Bio-terror, Bert, and Fish subplots of the VAST 2007 dataset. First, a layout of summary graph will be displayed on the screen. The user, sees the word-cloud of the documents in each super-node as a representation of that super-node. A mixture of the layout and word-cloud helps user to quickly choose the best super-node and further investigate it. For example, in Fig. 7.5a the user can easily notice the super-node that is related to chinchilla species (i.e., the middle bottom supper-node). Also, 7.5d highlights the super-node in the left-bottom



(a) Chinchilla: Word-cloud vis. (b) Chinchilla: Sum. net. vis. (c) Chinchilla: Expanded vis.

(d) Bert: Word-cloud vis. (e) Bert: Sum. net. vis. (f) Bert Expanded vis.

(g) Fish: Word-cloud vis. (h) Fish: Sum. net. vis. (i) Fish: Expanded vis.

Figure 7.5: The visualization of the Chinchilla bio-terror, Bert, and Fish sub-plots of VAST 2007 dataset. (a, d, g) show the word-cloud of the documents of each super-node in the summary network. (b, e, h) show the visualization of the summary graph. NetReAct expands the super-node that the user selects and shows its documents for further investigation. (c, f, i) show the expanded visualization of all super-nodes. Note, red nodes represent relevant documents to the scenario and the gray ones are irrelevant.

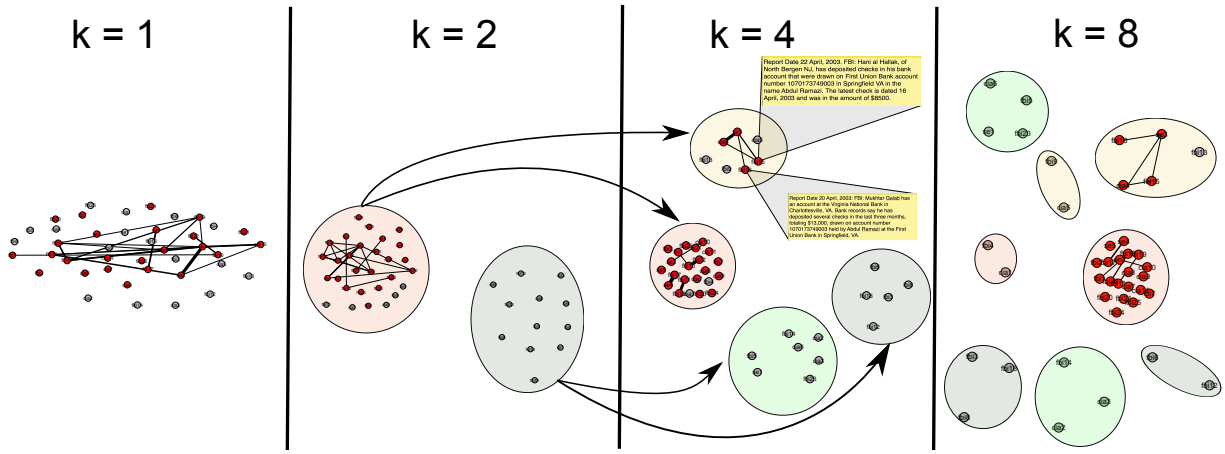


Figure 7.6: Hierarchical visualization of the CRESCENT dataset. NetReAct provides a multilevel visualization of the document data and lets the user investigate it with various granularity.

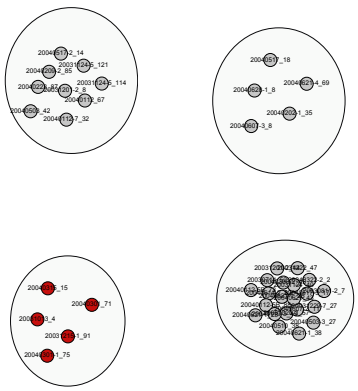
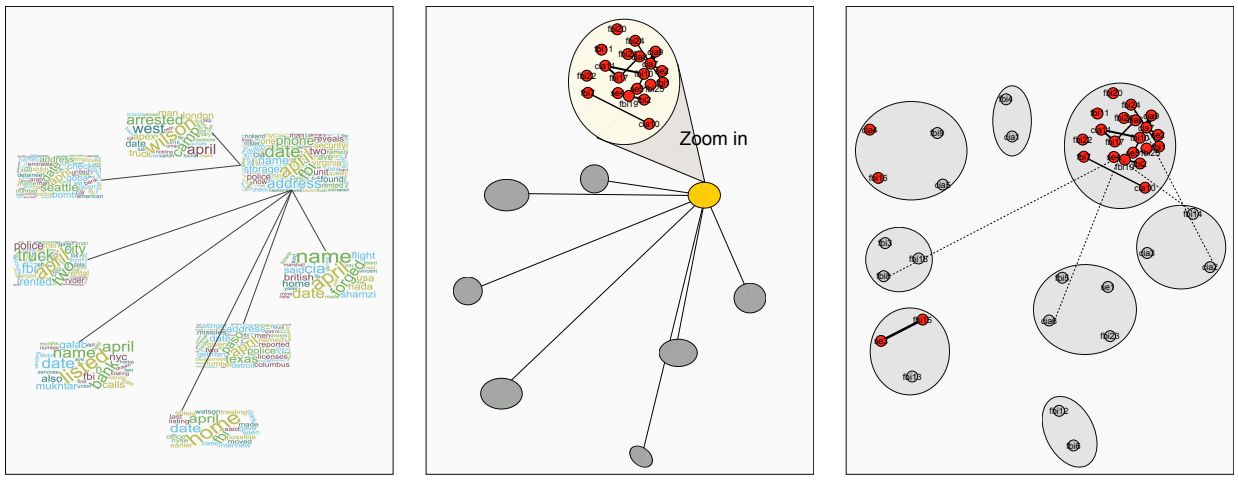


Figure 7.7: Visualization of the 'Circus' subplot with the use of the learned model on the 'Fish' subplot.



(a) Word-cloud visualization (b) Summary network visualization (c) Expanded visualization

Figure 7.8: The visualization of the CRESCENT dataset with the use of NetReAct. (a) Our method visualizes the summary document network and displays the word-cloud of the documents of each super-node as their representation. (b) It expands the super-node that the user selects and shows its documents for further investigation. (c) NetReAct can display all the documents and their hierarchical structure.

that is related to 'bear' and 'exotic' animals. When the user selects the super-node, it will be expanded (see Fig. 7.5b and Fig. 7.5e) to let the user investigate the documents inside it. Therefore, NetReAct guides the user to quickly find the relevant documents to the hidden story and avoid reading unnecessary ones.

Multi-level Visualization

Since NetReAct uses a hierarchical approach to summarize the document network, it has the ability to give a multi-level visualization of the corpus. As a result, the users can decide how much detail they want to identify and review at a variety of different levels. Fig. 7.6 shows the visualization of the CRESCENT data in multiple levels with a different number of super-nodes ($K = 1, 2, 4, 8$). By looking at the super-nodes at different levels, we understand the relevance and similarity between documents. In the first level (i.e., $K = 1$), all documents are in the same super-node and the visualization is merely a weighted, force-directed layout. Moving from the first to the second level tells us that even though the system can differentiate between relevant and irrelevant nodes to the hidden scenarios, there are a few documents that remain falsely detected. We note that among relevant documents, we are still unable to distinguish between different topics. By further dividing the super-nodes, we will be able to further identify the sub-plots in relevant documents. For example, in Fig. 7.6 with $K = 4$, we can highlight the phone calls and money transfers to an agent in Virginia by putting its related documents in a separate super-node. Such insight was not apparent in

the visualization with $K = 2$. Instead, the visualization with $K = 2$ gives a high-level understanding of the related documents separating them from the others.

Re-applying NetReAct

We aim to learn a generalizable model with the use of feedback-based reinforcement learning to be able to re-use the model in similar situations (e.g., similar visualization task and similar document corpora). To test the reusability of NetReAct, we learned a model on the 'Bert' subplot of the VAST 2007 dataset with a small amount of feedback from the user. Next, we reused this learned model on the 'Fish' subplot without any input from the user. Fig. 7.7 shows that the relevant documents are well separated from the others. NetReAct made it possible to re-apply the learned model on the new data and reduce the amount of required feedback significantly.

7.3.6 Summary of observations

To summarize, our main experimental observations are:

- **Generate effective document network summaries:** Our framework, NetReAct, effectively learns how to generate document network summaries with higher ρ (Eq. 7.6) compared to the baselines (See Sec. 7.3.3).
- **Provide multilevel understanding:** NetReAct provides a multi-level understanding of the document data and lets the user investigate the data with various granularity (See Sec. 7.3.5).
- **Learn reusable models:** The learned model in NetReAct is generalizable to unseen similar datasets, and it reduces or eliminates the amount of required feedback from the user. Therefore, it lowers the cost of generating high-quality visualizations for sense-making tasks (See Sec. 7.3.5).

7.4 Conclusions and Discussion

We explored the problem of learning interactive network summarizations to generate multi-level and generalizable visualization models for text analysis. We proposed a novel and effective algorithm NetReAct which leverages a feedback-based reinforcement learning approach to incorporate human input as well as the visualization task to produce a high-quality visualization. Our extensive experiments show that NetReAct is able to summarize and visualize a document network meaningfully to reveal hidden stories in the corpus and connect the dots between different documents.

Our approach here opens several interesting avenues for future work. We can explore using our approach for visualizing other data types such as social networks and images. Also, the flexibility we get from the reinforcement learning approach makes it possible to bring learning into visualization and enable better generalization and personalization. For example, we can build a personalized model for each user to reflect their interests and quickly visualize different datasets without much input from them.

We can also investigate how to formulate more diverse semantic interactions to incorporate into the reinforcement learning approach. For example, we would like to explore how to differentiate between highlighting, overlapping and annotating documents in our framework. Also, leveraging more visual encoding to create a more understandable and user-friendly visualization is a fruitful direction. Finally, even though NetReAct works well on medium sized corpora and solves real tasks in practice we plan to apply our network summarization model to much larger document datasets and temporal data scenarios.

Chapter 8

Conclusion and Future Work

In this thesis, we focused on summarizing large graphs and graph sequences based on a given task. To the best of our knowledge we are the first to systematically bring the necessity of considering the given task to the forefront and emphasize the importance of learning-based approaches in network summarization. We worked on the various types of networks and proposed optimization-based and learning-based approaches to summarize them for a vast number of tasks. We summarize the main idea and conclusions in each chapter below:

- **Chapter 3:** We leverage the fundamental so-called 'system matrix', and matrix perturbation presents an effective model-agnostic, near-linear, and parallelizable algorithm NetCondense to dramatically speed-up influence maximization and event detection algorithms on a variety of large temporal networks (i.e., up to $48X$). We also show the usefulness of such summaries to visualize, explore, and understand networks.
- **Chapter 4:** We presented SnapNETS, and practical 'global' algorithm to segment network sequences with binary labels. Our 'global' approach is not simply a change-point detection. We are not just looking for local changes; rather we track the 'total variation' in structural and rate characteristics using $\mathsf{G_s}$. Hence this allows us to find important cut-points automatically and without any specification. Therefore, our method is useful for detecting anomalies and important events in dynamic graph sequences.
- **Chapter 5:** We proposed ANeTS a new unsupervised, iterative, scalable and parallelizable algorithm to summarize attributed networks based on their influence-based and attribute-based properties for propagation based tasks. Such summarize help in sensemaking of complex network datasets by highlighting their important structural and attribute properties. We show how our summaries can speed-up solving the TIM task tremendously ($\sim 20X$) while maintaining the quality of the results.

- **Chapter 6:** We generalize over multiple threads of prior work and propose an effective method NetGist by leveraging the deep Q-learning framework. It is able to learn meaningful summaries for various tasks graph optimization tasks even with high reduction factors (up to 90%). Our framework helps to develop practical new algorithms for challenging open problems such as GUIDED-TBNS and also sensemaking via visualization. Also, since our approach is entirely automatic and reusable, it can generate summaries for multiple networks at the same time and use the learned model in other similar graphs. Hence, it has a significant impact on speeding up creating high-quality solutions for the given task.
- **Chapter 7:** We explored the problem of learning interactive network summarizations to generate multilevel and generalizable visualization models for text analysis. We proposed a novel and effective algorithm NetReAct which leverages a feedback-based reinforcement learning approach to incorporate human input as well as the visualization task to produce high-quality network summaries. Our method is able to summarize and visualize document networks meaningfully to reveal hidden stories in the corpus and connect the dots between different documents.

In summary, our work has many real world applications such as detecting events, anomalies and malicious activities and visualization and sensemaking.

8.1 Future Work

Our summarization algorithms generate high-quality network summaries and help in solving many real-world applications. However, there are still many interesting directions to explore which we discuss next.

8.1.1 Summarizing streaming and heterogeneous networks

One of the natural expansion of our work is to consider other types of networks such as heterogeneous networks. Many of the real-world networks are heterogeneous. For example, in social networks such as Facebook, there are different types of edges and nodes in data. Nodes can be users, pages, events, images, etc. and edges can be friendship, following, likes and so forth. Each type of node or edge in such a network can have a different set of attributes. We can leverage an ANeTS style approach to maintain the structural, and attribute characteristics of the networks by merging best nodes and define the supernodes and superedges in a way that matches the heterogeneous nature of the network. Also, summarizing streaming network is another exciting and challenging extension of our work. For summarizing such networks, we should be able to answer several questions such as; How

often should we update the summary? How to select similar nodes to merge? How regularly should we update the similarity criteria? And so on.

8.1.2 Summarizing partially observed or noisy networks

Another important direction for future work is to study summarizing partially observed networks. In most of the real-world networks, the entire structure of the network is not available due to technical or privacy reasons. So, in most cases, we use network crawling approaches or infer the underlying networks. It is important to consider the uncertainty factor in summarization algorithms. Also, it is interesting to study the robustness of current approaches to such partially observed and noisy networks and estimate their quality.

8.1.3 Exploring prediction and forecasting tasks

Our learning approach opens several exciting avenues for future work. Exploring summarization methods to solve other kinds of tasks such as prediction and recommendation is a promising future direction. The flexibility we get from the reinforcement learning approach makes it possible to bring learning into recommendation and enable better generalization and personalization. For example, we can build a personalized model for each user in a social network to reflect their interests and recommend friends or pages without much input from them. We can also investigate how to formulate more human feedback to incorporate into the reinforcement learning approach. For example, in the context of recommendation in a social network, we would like to explore how to differentiate between reacting, commenting and reading a post in our framework.

8.1.4 Learning to summarize with the use of human input

In this thesis, we explored how to incorporate human feedback with the learning process to generate high-quality summarization for the task of document visualizations. We believe that learning to summarize is more general and can be used for other tasks in other applications and domains. Exploring such applications and other types of human feedback that is particular for those applications is an interesting future track.

8.1.5 Scaling up learning based approaches

To summarize real-world networks and solve practical problems with a reinforcement learning approach we used deep-RL and leveraged GPU implementation. However, to be able to scale up our learning approaches we can design more effective graph embedding architectures in

combination with RL algorithms to achieve faster convergence. Also, exploring other RL algorithms such as policy gradient and actor-critic in addition to Q-learning is an exciting future direction. Furthermore, incorporating other types of user feedback is an exciting research track to speed up the learning based approaches. For example, a user can provide group feedback such as stars (i.e., a node is central in a group), clique (i.e., nodes are similar in a group), chain (i.e., there is causality relation between nodes), etc.

Bibliography

[1] L. A. Adamic and B. A. Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000.

[2] B. Adhikari, Y. Zhang, S. E. Amiri, A. Bharadwaj, and B. A. Prakash. Propagation-based temporal network summarization. *IEEE Trans. Knowl. Data Eng.*, 30(4):729–742, 2018.

[3] C. C. Aggarwal and N. Li. On node classification in dynamic content-based networks. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pages 355–366. SIAM / Omnipress, 2011.

[4] C. C. Aggarwal, S. Lin, and P. S. Yu. On influential node discovery in dynamic social networks. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 636–647. SIAM / Omnipress, 2012.

[5] C. C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 47(1):10:1–10:36, 2014.

[6] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761, 2010.

[7] L. Akoglu, H. Tong, B. Meeder, and C. Faloutsos. PICS: parameter-free identification of cohesive subgroups in large attributed graphs. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 439–450. SIAM / Omnipress, 2012.

[8] R. Albert and A. Barabási. Statistical mechanics of complex networks. *CoRR*, cond-mat/0106096, 2001.

[9] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378, 2000.

[10] S. E. Amiri, B. Adhikari, A. Bharadwaj, and B. A. Prakash. Netgist: Learning to generate task-based network summaries. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 857–862, 2018.

[11] S. E. Amiri, B. Adhikari, M. Dowling, J. Wenskovitch, C. North, and B. A. Prakash. Learning to generate effective network summaries. In *(Under review)*, 2019.

[12] S. E. Amiri, B. Adhikari, J. Wenskovitch, M. Dowling, C. North, and B. A. Prakash. Netreact: Learning network visualizations for text analytics using interactions. In *(Under review)*. IEEE Computer Society, 2019.

[13] S. E. Amiri, L. Chen, and B. A. Prakash. Segmenting sequences of node-labeled graphs. In *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain.*, pages 61–68, 2016.

[14] S. E. Amiri, L. Chen, and B. A. Prakash. Snapnets: Automatic segmentation of network sequences with node labels. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3–9, 2017.

[15] S. E. Amiri, L. Chen, and B. A. Prakash. Automatic segmentation of dynamic network sequences with node labels. *IEEE Trans. Knowl. Data Eng.*, 30(3):407–420, 2018.

[16] S. E. Amiri, L. Chen, and B. A. Prakash. Efficiently summarizing attributed diffusion networks. *Data Min. Knowl. Discov.*, 32(5):1251–1274, 2018.

[17] G. Amitai, A. Shemesh, E. Sitbon, M. Shklar, D. Netanely, I. Venger, and S. Pietrokovski. Network analysis of protein structures identifies functional residues. *Journal of molecular biology*, 344(4):1135–1146, 2004.

[18] R. M. Anderson and R. M. May. *Infectious diseases of humans: dynamics and control*. Oxford university press, 1992.

[19] C. Andrews, A. Endert, and C. North. Space to think: Large high-resolution displays for sensemaking. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 55–64, 2010.

[20] C. Andrews, A. Endert, and C. North. Space to think: large high-resolution displays for sensemaking. In E. D. Mynatt, D. Schoner, G. Fitzpatrick, S. E. Hudson, W. K. Edwards, and T. Rodden, editors, *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010*, pages 55–64. ACM, 2010.

[21] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra. Com2: Fast automatic discovery of temporal ('comet') communities. In V. S. Tseng, T. B. Ho, Z. Zhou, A. L. P. Chen, and H. Kao, editors,

Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part II, volume 8444 of *Lecture Notes in Computer Science*, pages 271–283. Springer, 2014.

[22] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[23] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, and X. Wu, editors, *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 81–90. IEEE Computer Society, 2012.

[24] I. Batal, D. Fradkin, J. H. H. Jr., F. Moerchen, and M. Hauskrecht. Mining recent temporal patterns for event detection in multivariate time series data. In Q. Yang, D. Agarwal, and J. Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 280–288. ACM, 2012.

[25] A. Bay and B. Sengupta. Approximating meta-heuristics with homotopic recurrent neural networks. *CoRR*, abs/1709.02194, 2017.

[26] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

[27] K. R. Bisset, A. M. Aji, M. V. Marathe, and W.-c. Feng. High-performance biocomputing for simulating the spread of contagion over large contact networks. *BMC genomics*, 13(2):S3, 2012.

[28] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[29] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenková. Clustering attributed graphs: Models, measures and methods. *Network Science*, 3(3):408–444, 2015.

[30] L. Bradel, C. North, L. House, and S. Leman. Multi-model semantic interaction for text analytics. In *2014 IEEE Conference on Visual Analytics Science and Technology, VAST 2014, Paris, France, October 25-31, 2014*, pages 163–172, 2014.

[31] L. Bradel, C. North, L. House, and S. Leman. Multi-model semantic interaction for text analytics. In M. Chen, D. S. Ebert, and C. North, editors, *2014 IEEE Conference on Visual Analytics Science and Technology, VAST 2014, Paris, France, October 25-31, 2014*, pages 163–172. IEEE Computer Society, 2014.

[32] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the International Conference on Web Search and Web Data Mining, WSDM 2008, Palo Alto, California, USA, February 11-12, 2008*, pages 95–106, 2008.

[33] N. Cao, Y. Lin, L. Li, and H. Tong. g-miner: Interactive visual group mining on multivariate graphs. In B. Begole, J. Kim, K. Inkpen, and W. Woo, editors, *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 279–288. ACM, 2015.

[34] C. Chen, H. Tong, B. A. Prakash, C. E. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau. Node immunization on large graphs: Theory and algorithms. *IEEE Trans. Knowl. Data Eng.*, 28(1):113–126, 2016.

[35] S. Chen, J. Fan, G. Li, J. Feng, K. Tan, and J. Tang. Online topic-aware influence maximization. *PVLDB*, 8(6):666–677, 2015.

[36] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 199–208. ACM, 2009.

[37] X. C. Chen, K. Steinhaeuser, S. Boriah, S. Chatterjee, and V. Kumar. Contextual time series change detection. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA.*, pages 503–511. SIAM, 2013.

[38] S. Chiappa. A bayesian approach to switching linear gaussian state-space models for unsupervised time-series segmentation. In M. A. Wani, X. Chen, D. Casasent, L. A. Kurgan, T. Hu, and K. Hafeez, editors, *Seventh International Conference on Machine Learning and Applications, ICMLA 2008, San Diego, California, USA, 11-13 December 2008*, pages 3–9. IEEE Computer Society, 2008.

[39] 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.

[40] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Commun. ACM*, 47(3):45–47, 2004.

[41] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[42] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY,*

USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2702–2711. JMLR.org, 2016.

[43] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Trans. Graph.*, 15(4):301–331, 1996.

[44] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific reports*, 3:2980, 2013.

[45] E. Desmier, M. Plantevit, C. Robardet, and J. Boulicaut. Trend mining in dynamic attributed graphs. In H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*, volume 8188 of *Lecture Notes in Computer Science*, pages 654–669. Springer, 2013.

[46] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, 2007.

[47] P. Doreian, V. Batagelj, and A. Ferligoj. Generalized blockmodeling of two-mode network data. *Social Networks*, 26(1):29–53, 2004.

[48] T. Dwyer and Y. Koren. DIG-COLA: directed graph layout through constrained energy minimization. In J. T. Stasko and M. O. Ward, editors, *IEEE Symposium on Information Visualization (InfoVis 2005), 23-25 October 2005, Minneapolis, MN, USA*, pages 65–72. IEEE Computer Society, 2005.

[49] M. D. Dyer, C. Neff, M. Dufford, C. G. Rivera, D. Shattuck, J. Bassaganya-Riera, T. Murali, and B. W. Sobral. The human-bacterial pathogen protein interaction networks of bacillus anthracis, francisella tularensis, and yersinia pestis. *PloS one*, 5(8):e12089, 2010.

[50] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[51] A. Endert, R. Burtner, N. Cramer, R. Perko, S. Hampton, and K. Cook. Typograph: Multiscale spatial exploration of text documents. In *2013 IEEE International Conference on Big Data*, pages 17–24, Oct 2013.

[52] A. Endert, P. Fiaux, and C. North. Semantic interaction for sensemaking: Inferring analytical reasoning for model steering. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2879–2888, 2012.

[53] S. Eubank, H. Guclu, V. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180, 2004.

[54] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[55] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 157–168. ACM, 2012.

[56] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, and M. Grobelnik. Monitoring network evolution using MDL. In G. Alonso, J. A. Blakeley, and A. L. P. Chen, editors, *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 1328–1330. IEEE Computer Society, 2008.

[57] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[58] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[59] J. Fournet and A. Barrat. Contact patterns among high school students. *CoRR*, abs/1409.5318, 2014.

[60] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi. A general framework for graph sparsification. In L. Fortnow and S. P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC, 2011, San Jose, CA, USA, 6-8 June 2011*, pages 71–80. ACM, 2011.

[61] A. Ganesh, L. Massoulié, and D. Towsley. The effect of network topology on the spread of epidemics. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 1455–1466. IEEE, 2005.

[62] F. R. Gantmacher and J. L. Brenner. *Applications of the Theory of Matrices*. Courier Corporation, 2005.

[63] N. T. H. Gayraud, E. Pitoura, and P. Tsaparas. Diffusion maximization in evolving social networks. In A. Sharma, R. Agrawal, and M. Grossglauser, editors, *Proceedings of the 2015 ACM on Conference on Online Social Networks, COSN 2015, Palo Alto, California, USA, November 2-3, 2015*, pages 125–135. ACM, 2015.

[64] M. Génois, C. L. Vestergaard, J. Fournet, and A. Panisson. Data on face-to-face contacts in an office building suggest a low-cost vaccination strategy based on community linkers. *Network Science*, 3(3):326–347, 2015.

[65] A. Gensler, T. Gruber, and B. Sick. Blazing fast time series segmentation based on update techniques for polynomial approximations. In W. Ding, T. Washio, H. Xiong,

G. Karypis, B. M. Thuraisingham, D. J. Cook, and X. Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM, Workshops, TX, USA, December 7-10, 2013*, pages 1002–1011. IEEE Computer Society, 2013.

[66] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[67] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang, editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 1019–1028. ACM, 2010.

[68] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[69] C. Görg, Z. Liu, J. Kihm, J. Choo, H. Park, and J. T. Stasko. Combining computational analyses and interactive visualization for document exploration and sensemaking in jigsaw. *IEEE Trans. Vis. Comput. Graph.*, 19(10):1646–1663, 2013.

[70] G. G. Grinstein, C. Plaisant, S. J. Laskowski, T. A. O'Connell, J. Scholtz, and M. A. Whiting. VAST 2007 contest - blue iguanodon. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology, IEEE VAST 2007, Sacramento, California, USA, October 30-November 1, 2007*, pages 231–232. IEEE Computer Society, 2007.

[71] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016.

[72] N. Guan, X. Huang, L. Lan, Z. Luo, and X. Zhang. Graph based semi-supervised non-negative matrix factorization for document clustering. In *11th International Conference on Machine Learning and Applications, ICMLA, Boca Raton, FL, USA, December 12-15, 2012. Volume 1*, pages 404–408. IEEE, 2012.

[73] I. Günes, Z. Çataltepe, and S. G. Ögüdücü. Ga-tvrc-het: genetic algorithm enhanced time varying relational classifier for evolving heterogeneous networks. *Data Min. Knowl. Discov.*, 28(3):670–701, 2014.

[74] S. Günnemann, B. Boden, and T. Seidl. DB-CSC: A density-based approach for subspace clustering in graphs with feature vectors. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*, volume 6911 of *Lecture Notes in Computer Science*, pages 565–580. Springer, 2011.

[75] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun. Evolutionary clustering and analysis of bibliographic networks. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2011, Kaohsiung, Taiwan, 25-27 July 2011*, pages 63–70. IEEE Computer Society, 2011.

[76] M. GNOIS, C. L. VESTERGAARD, J. FOURNET, A. PANISSON, I. BONMARIN, and A. BARRAT. Data on face-to-face contacts in an office building suggest a low-cost vaccination strategy based on community linkers. *Network Science*, 3:326–347, 9 2015.

[77] L. W. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[78] K. Henderson, T. Eliassi-Rad, C. Faloutsos, L. Akoglu, L. Li, K. Maruhashi, B. A. Prakash, and H. Tong. Metric forensics: a multi-level approach for mining volatile graphs. In B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang, editors, *KDD*, pages 163–172. ACM, 2010.

[79] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

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

[81] F. Hughes and D. Schum. Discovery-proof-choice, the art and science of the process of intelligence analysis-preparing for the future of intelligence analysis. *Joint Military Intelligence College, Washington, DC.*, 2003.

[82] A. K. Jain. Data clustering: 50 years beyond k-means. In W. Daelemans, B. Goethals, and K. Morik, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, volume 5211 of *Lecture Notes in Computer Science*, pages 3–4. Springer, 2008.

[83] U. Kang, J. Y. Lee, D. Koutra, and C. Faloutsos. Net-ray: Visualizing and mining billion-scale graphs. In V. S. Tseng, T. B. Ho, Z. Zhou, A. L. P. Chen, and H. Kao, editors, *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, volume 8443 of *Lecture Notes in Computer Science*, pages 348–361. Springer, 2014.

[84] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, 2002.

[85] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[86] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

[87] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1998, November 7-13, 1998, Orlando, FL, USA*, page 28. IEEE Computer Society, 1998.

[88] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In L. Getoor, T. E. Senator, P. M. Domingos, and C. Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24- 27, 2003*, pages 137–146. ACM, 2003.

[89] Y. Keneshloo, J. Cadena, G. Korkmaz, and N. Ramakrishnan. Detecting and forecasting domestic political crises: a graph-based approach. In *ACM Web Science Conference, WebSci '14, Bloomington, IN, USA, June 23-26, 2014*, pages 192–196, 2014.

[90] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6351–6361, 2017.

[91] A. Khan, S. S. Bhowmick, and F. Bonchi. Summarizing static and dynamic big graphs. *PVLDB*, 10(12):1981–1984, 2017.

[92] M. Kim and J. Han. A particle-and-density based evolutionary clustering method for dynamic networks. *PVLDB*, 2(1):622–633, 2009.

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

[94] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 195–202, 2009.

[95] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. VOG: summarizing and understanding large graphs. In M. J. Zaki, Z. Obradovic, P. Tan, A. Banerjee, C. Kamath, and S. Parthasarathy, editors, *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 91–99. SIAM, 2014.

[96] H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 591–600. ACM, 2010.

[97] O. Kwon, C. Muelder, K. Lee, and K. Ma. Spherical layout and rendering methods for immersive graph visualization. In S. Liu, G. Scheuermann, and S. Takahashi, editors, *2015 IEEE Pacific Visualization Symposium, PacificVis 2015, Hangzhou, China, April 14-17, 2015*, pages 63–67. IEEE Computer Society, 2015.

[98] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding effectors in social networks. In B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang, editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 1059–1068. ACM, 2010.

[99] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[100] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1):5, 2007.

[101] J. Leskovec, L. Backstrom, and J. M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 497–506. ACM, 2009.

[102] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 631–640. ACM, 2010.

[103] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[104] C. Li, J. Ma, X. Guo, and Q. Mei. Deepcas: An end-to-end predictor of information cascades. In R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 577–586. ACM, 2017.

[105] G. Li, M. Semerci, B. Yener, and M. J. Zaki. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining*, 5(4):265–283, 2012.

[106] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos. Dynammo: mining and summarization of coevolving sequences with missing values. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 507–516. ACM, 2009.

[107] A. Likas, N. A. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

[108] W. Liu, A. Kan, J. Chan, J. Bailey, C. Leckie, J. Pei, and K. Ramamohanarao. On compressing weighted time-evolving graphs. In X. Chen, G. Lebanon, H. Wang, and M. J. Zaki, editors, *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 2319–2322. ACM, 2012.

[109] Y. Liu, A. Dighe, T. Safavi, and D. Koutra. A graph summarization: A survey. *CoRR*, abs/1612.04883, 2016.

[110] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In C. Apté, J. Ghosh, and P. Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 529–537. ACM, 2011.

[111] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Autoplait: automatic mining of coevolving time sequences. In C. E. Dyreson, F. Li, and M. T. Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 193–204. ACM, 2014.

[112] M. Mendoza, B. Poblete, F. Bravo-Marquez, and D. Gayo-Avello. Long-memory time series ensembles for concept shift detection. In W. Fan, A. Bifet, Q. Yang, and P. S. Yu, editors, *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine 2013, Chicago, IL, USA, August 11, 2013*, pages 23–30. ACM, 2013.

[113] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[114] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Finding strongly knit clusters in social networks. *Internet Mathematics*, 5(1):155–174, 2008.

[115] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[116] T. Munzner. *Visualization Analysis and Design*. A.K. Peters visualization series. A K Peters, 2014.

[117] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In J. T. Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 419–432. ACM, 2008.

[118] M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[119] M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[120] M. E. Newman. Communities, modules and large-scale structure in networks. *Nature physics*, 8(1):25, 2012.

[121] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2014–2023. JMLR.org, 2016.

[122] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna. A note on learning algorithms for quadratic assignment with graph neural networks. *CoRR*, abs/1706.07450, 2017.

[123] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814, 2005.

[124] B. Perozzi, L. Akoglu, P. I. Sánchez, and E. Müller. Focused clustering and outlier detection in large attributed graphs. In S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 1346–1355. ACM, 2014.

[125] R. Pienta, J. Abello, M. Kahng, and D. H. Chau. Scalable graph exploration and visualization: Sensemaking challenges and opportunities. In *2015 International Conference on Big Data and Smart Computing, BIGCOMP, 2015, Jeju, South Korea, February 9-11, 2015*, pages 271–278. IEEE Computer Society, 2015.

[126] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. A. Roundy, C. S. Gates, S. B. Navathe, and D. H. Chau. VIGOR: interactive visual exploration of graph query results. *IEEE Trans. Vis. Comput. Graph.*, 24(1):215–225, 2018.

[127] R. Pienta, M. Kahng, Z. Lin, J. Vreeken, P. P. Talukdar, J. Abello, G. Parameswaran, and D. H. Chau. FACETS: adaptive local exploration of large graphs. In N. Chawla and W. Wang, editors, *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017.*, pages 597–605. SIAM, 2017.

[128] B. A. Prakash, D. Chakrabarti, M. Faloutsos, N. Valler, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In D. J. Cook, J. Pei, W. Wang, O. R. Zaïane, and X. Wu, editors, *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 537–546. IEEE Computer Society, 2011.

[129] B. A. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos. Virus propagation on time-varying networks: Theory and immunization algorithms. In J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III*, volume 6323 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2010.

[130] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. S. Subrahmanian. Fast influence-based coarsening for large networks. In S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 1296–1305. ACM, 2014.

[131] Q. Qu, S. Liu, C. S. Jensen, F. Zhu, and C. Faloutsos. Interestingness-driven diffusion process summarization in dynamic networks. In T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*, volume 8725 of *Lecture Notes in Computer Science*, pages 597–613. Springer, 2014.

[132] S. Rayana and L. Akoglu. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In S. Venkatasubramanian and J. Ye, editors, *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, pages 622–630. SIAM, 2015.

[133] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 4292–4293. AAAI Press, 2015.

[134] Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, and S. B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1089–1098. International World Wide Web Conferences Steering Committee / ACM, 2013.

[135] T. Ruotsalo, J. Peltonen, M. Eugster, D. Głowacka, K. Konyushkova, K. Athukorala, I. Kosunen, A. Reijonen, P. Myllymäki, G. Jacucci, and S. Kaski. Directing exploratory search with interactive intent modeling. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 1759–1764, New York, NY, USA, 2013. ACM.

[136] D. Salvi, J. Zhou, J. W. Waggoner, and S. Wang. Handwritten text segmentation using average longest path algorithm. In *2013 IEEE Workshop on Applications of Computer Vision, WACV 2013, Clearwater Beach, FL, USA, January 15-17, 2013*, pages 505–512. IEEE Computer Society, 2013.

[137] A. Samé and G. Govaert. Online time series segmentation using temporal mixture models and bayesian model selection. In *11th International Conference on Machine Learning and Applications, ICMLA, Boca Raton, FL, USA, December 12-15, 2012. Volume 1*, pages 602–605. IEEE, 2012.

[138] P. Sarkar, D. Chakrabarti, and M. I. Jordan. Nonparametric link prediction in dynamic networks. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[139] J. Scott. Social network analysis. *Sociology*, 22(1):109–127, 1988.

[140] B. Seah, S. S. Bhowmick, C. F. D. Jr., and H. Yu. FUSE: a profit maximization approach for functional summarization of biological networks. *BMC Bioinformatics*, 13(S-3):S10, 2012.

[141] J. Z. Self, M. Dowling, J. E. Wenskovitch, I. Crandell, M. Wang, L. House, S. Leman, and C. North. Observation-level and parametric interaction for high-dimensional data analysis. *TiiS*, 8(2):15:1–15:36, 2018.

[142] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[143] S. Senecal and J. Nantel. The influence of online product recommendations on consumers online choices. *Journal of retailing*, 80(2):159–169, 2004.

[144] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In L. Cao, C. Zhang, T. Joachims, G. I. Webb,

D. D. Margineantu, and G. Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1055–1064. ACM, 2015.

[145] L. Shi, H. Tong, J. Tang, and C. Lin. VEGAS: visual influence graph summarization on citation networks. *IEEE Trans. Knowl. Data Eng.*, 27(12):3417–3431, 2015.

[146] F. M. Shipman and C. C. Marshall. Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems. *Computer Supported Cooperative Work (CSCW)*, 8(4):333–352, 1999.

[147] G. W. Stewart. Matrix perturbation theory. *Citeseer*, 1990.

[148] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In P. Berkhin, R. Caruana, and X. Wu, editors, *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 687–696. ACM, 2007.

[149] S. Sundereisan, A. Bhadriraju, M. S. Khan, N. Ramakrishnan, and B. A. Prakash. Sanstext: Classifying temporal topic dynamics of twitter cascades without tweet text. In X. Wu, M. Ester, and G. Xu, editors, *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2014, Beijing, China, August 17-20, 2014*, pages 649–656. IEEE Computer Society, 2014.

[150] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[151] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 449–456, New York, NY, USA, 2005. ACM.

[152] J. Thompson, A. Srinivasan, and J. T. Stasko. Tangraphe: interactive exploration of network visualizations using single hand, multi-touch gestures. In T. Catarci, K. L. Norman, and M. Mecella, editors, *Proceedings of the 2018 International Conference on Advanced Visual Interfaces, AVI 2018, Castiglione della Pescaia, Italy, May 29 - June 01, 2018*, pages 43:1–43:5. ACM, 2018.

[153] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In J. T. Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 567–580. ACM, 2008.

[154] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In C. Apté, J. Ghosh, and P. Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 965–973. ACM, 2011.

[155] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In X. Chen, G. Lebanon, H. Wang, and M. J. Zaki, editors, *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 245–254. ACM, 2012.

[156] V. S. Tseng, C. Chen, P. Huang, and T. Hong. A cluster-based genetic approach for segmentation of time series and pattern discovery. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC, 2008, June 1-6, 2008, Hong Kong, China*, pages 1949–1953. IEEE, 2008.

[157] N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 458–465. IEEE Computer Society, 2002.

[158] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[159] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1225–1234. ACM, 2016.

[160] J. Wang, S. Rao, J. Chu, X. Shen, D. N. Levasseur, T. W. Theunissen, and S. H. Orkin. A protein interaction network for pluripotency of embryonic stem cells. *Nature*, 444(7117):364, 2006.

[161] P. Wang, H. Wang, and W. Wang. Finding semantics in time series. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 385–396. ACM, 2011.

[162] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[163] D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440, 1998.

[164] J. E. Wenskovitch, I. Crandell, N. Ramakrishnan, L. House, S. Leman, and C. North. Towards a systematic combination of dimension reduction and clustering in visual analytics. *IEEE Trans. Vis. Comput. Graph.*, 24(1):131–141, 2018.

[165] J. E. Wenskovitch and C. North. Observation-level interaction with clustering and dimension reduction algorithms. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2017, Chicago, IL, USA, May 14, 2017*, pages 14:1–14:6, 2017.

[166] J. J. Whang, I. S. Dhillon, and D. F. Gleich. Non-exhaustive, overlapping k-means. In S. Venkatasubramanian and J. Ye, editors, *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, pages 936–944. SIAM, 2015.

[167] Y. Wu, Z. Zhong, W. Xiong, and N. Jing. Graph summarization for attributed graphs. In *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, volume 1, pages 503–507. IEEE, 2014.

[168] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell. Distance metric learning with application to clustering with side-information. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 505–512. MIT Press, 2002.

[169] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 505–516. ACM, 2012.

[170] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In L. Getoor, T. E. Senator, P. M. Domingos, and C. Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 286–295. ACM, 2003.

[171] P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1365–1374. ACM, 2015.

[172] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, and X. Wu, editors, *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 745–754. IEEE Computer Society, 2012.

[173] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In S. Leonardi, A. Panconesi, P. Ferragina, and A. Gionis, editors, *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, pages 587–596. ACM, 2013.

[174] J. Yang, J. J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In H. Xiong, G. Karypis, B. M. Thuraisingham, D. J. Cook, and X. Wu, editors, *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, pages 1151–1156. IEEE Computer Society, 2013.

[175] F. Zhang. *Matrix theory: basic results and techniques*. Springer Science & Business Media, 2011.

[176] H. Zhang, M. Sun, D. D. Yao, and C. North. Visualizing traffic causality for analyzing network anomalies. In S. Huang and R. M. Verma, editors, *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics, IWSPA@CODASPY 2015, San Antonio, TX, USA, March 4, 2015*, pages 37–42. ACM, 2015.

[177] H. Zhang, D. D. Yao, and N. Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, pages 39–50, 2014.

[178] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 880–891, 2010.

[179] Y. Zhang and B. A. Prakash. DAVA: distributing vaccines over networks under prior information. In M. J. Zaki, Z. Obradovic, P. Tan, A. Banerjee, C. Kamath, and S. Parthasarathy, editors, *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 46–54. SIAM, 2014.

[180] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[181] Y. Zhou, H. Cheng, and J. X. Yu. Clustering large attributed graphs: An efficient incremental approach. In G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 689–698. IEEE Computer Society, 2010.