# Immunization and Summarization of Large Graphs

**Muhammad Ahmad**

2017-03-0056

**Advisor: Dr. Imdadullah Khan**

5

Department of Computer Science,

School of Science and Engineering,

Lahore University of Management Sciences (LUMS), Pakistan

This dissertation is submitted for the degree of *Doctor of Philosophy*

*Dedicated to my family*

**Lahore University of Management Sciences**

**School of Science and Engineering**

**CERTIFICATE**

We hereby recommend that the thesis prepared under our supervision by ***Muhammad Ahmad*** titled ***Immunization and Summarization of Large Graphs*** be accepted in partial fulfillment of the requirements for the degree of PhD.

Dr. Imdadullah Khan (Advisor)
Dr. Asim Karim (Co-Advisor)

## Acknowledgements

I am very thankful to my advisor Imdadullah Khan for his support and guidance in my academic journey. His sincere efforts from the initial to the final stage have made this work possible. I am also very much indebted to everyone who has helped me in my research work. In the end, I am thankful to the Higher Education Commission (HEC), Pakistan and Lahore University of Management Sciences (LUMS), Pakistan for funding this research.

# List of Publications

## Publications

### Journal

1. **Muhammad Ahmad**, Sarwan Ali, Juvaria Tariq, Imdadullah Khan, Mudassir Shabbir, Arif Zaman, Combinatorial trace method for network immunization in Information Sciences, 2020.

2. **Muhammad Ahmad**, Juvaria Tariq, Mudassir Shabbir, Imdadullah Khan, Spectral Methods for Immunization of Large Networks in Australasian Journal of Information Systems, 2017

### Conference

1. Maham Anwar Beg, **Muhammad Ahmad**, Arif Zaman, Imdadullah Khan, Scalable Approximation Algorithm for Graph Summarization in Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD, 2018

2. Juvaria Tariq, **Muhammad Ahmad**, Imdadullah Khan, Mudassir Shabbir Scalable Approximation Algorithm for Graph Immunization in Pacific Asia Conference on Information Systems, PACIS, 2017

3. **Muhammad Ahmad**, Juvaria Tariq, Muhammad Farhan, Mudassir Shabbir, Imdadullah Khan, *Who Should Receive the Vaccine?* in The Australasian Data Mining Conference (AusDM), 2016

# Abstract

In this thesis, we present results for two problems related to graphs.

**Network Immunization** Immunizing a subset of nodes in a network - enabling them to identify and withstand the spread of harmful content - is one of the most effective ways to counter the spread of malicious content. It has applications in network security, public health policy, and social media surveillance. Finding a subset of nodes whose immunization results in the least vulnerability of the network is a computationally challenging task. In this work, we establish a relationship between a widely used network vulnerability measure and the combinatorial properties of networks. Using this relationship and graph summarization techniques, we propose an efficient approximation algorithm to find a set of nodes to immunize. We provide theoretical justifications for the proposed solution and analytical bounds on the runtime of our algorithm. We empirically demonstrate on various real-world networks that the performance of our algorithm is an order of magnitude better than the state-of-the-art solution. We also show that in practice the runtime of our algorithm is significantly lower than that of the best-known solution.

**Graph Summarization** Massive sizes of real-world graphs, such as social networks and web graphs, impose serious challenges to process and perform analytics on them. These issues can be resolved by working on a small summary of the graph instead. A summary is a compressed version of the graph that removes several details, yet preserves its essential structure. Generally, some predefined quality measure of the summary is optimized to bound the approximation error incurred by working on the summary instead of the whole graph. All known summarization algorithms are computationally prohibitive and do not scale to large graphs. In this paper, we present an efficient randomized algorithm to compute graph summaries with the goal to minimize *reconstruction error*. We propose a novel weighted sampling scheme to sample vertices for merging that will result in the least reconstruction error. We provide analytical bounds on the running time of the algorithm

and prove an approximation guarantee for our score computation. The efficiency of our algorithm makes it scalable to very large graphs on which known algorithms cannot be applied. We test our algorithm on several real-world graphs to empirically demonstrate the quality of summaries produced and compare to the state-of-the-art algorithms. We use the summaries to answer several structural queries about the original graph and report their accuracies.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In recent years, a massive amount of data is being generated in various fields such as online social media, e-commerce, the internet, embedded systems and geo-positioning. The well-known sources of data include social media platforms of Facebook, Twitter and Instagram, online retail stores such as Amazon and e-Bay, web search engines like Google and Bing, the Internet of Things (IoT) and sensor devices installed in complicated machines, and Global Positioning System devices which record location co-ordinates. The generation rate of data from the above-mentioned sources is great and it is estimated that $90\%$ of the total generated data is produced in almost the last couple of years[1]. This huge amount of data is processed and analyzed to get meaningful insights and make data-driven decisions that are profitable and efficient. These decisions are mostly related to risk assessment, cost-cutting and demand prediction.

A lot of data generated from the above-mentioned sources consists of interactions among participating entities. The common examples of pairwise interactions among entities include friendships and connections among users in online social media platforms, pairs of items co-purchased in retail stores, cross-links among different web pages on the internet, communication and information sharing among sensor devices and control machines and the movement of a device from one location to another. The interactions among entities are normally represented as graphs [1]. A graph consists of nodes and edges in which the objects are represented as nodes and an edge be-

[1] IBM report of 2012 on big data, Forbes 2019

tween two nodes shows an interaction between the corresponding nodes. Representation of data as graphs has been successfully used in diverse domains like data mining, medical sciences, operating systems and road mapping.

In pairwise interactions, nodes normally exchange information with each other. The nature of communication and information shared varies depending upon the type of interactions and nodes. Common examples are sharing of opinions with friends in a social network, the exchange of greeting messages in a human interaction, the delegation of tasks from a server machine to a client machine in a data center, sharing of news content in an online social network and the transfer of sensor values form sensor devices to control units.

However, in information sharing, there is a chance of the spread of malicious information among the participating nodes. A node can deliberately or unintentionally pass irrelevant or harmful information to others. For example in a human network, a person can be a source of the spread of a disease, in a data center a compute machine can respond by sending a malfunctioning code that can affect the performance of the whole data center, in an online social media network, a user can start sharing wrong information in the form of fake news and in an Internet of Things ecosystem some sensor devices can report inappropriate or invalid sensor values.

The spread of irrelevant and malicious content can degrade the usability capacity and performance of a given network. In order to protect the network, there is a need of countering the possible spread of malicious content in the network. An effective way to safeguard a network against the spread of malicious content is to *empower* the nodes. Empowering nodes accounts to giving extra capabilities to a node such that the node acts as a shield against malicious content. The strengthening process may amount to vaccinating people, deploying surveillance systems at junctures and installing anti-virus software on computers depending on the underlying network. The nodes with these added capabilities will be referred to as the immunized nodes and the malicious content, as the virus. Effectively, when a node is immunized, it will neither get contaminated nor will it pass the contaminant to other nodes.

In addition to the threat of the spread of malicious content in a graph, the large sizes of massive graphs pose another challenge in processing and performing analytics on them. For example, popular social media networks consist of billions of nodes and billions of edges [1]. Facebook graph consists of around $3$ billion nodes and has $2$ billion active users, Instagram has nearly $1$ billion active users, the webgraph consists of links between billions of web pages of the world wide web and on Twitter, around $500$ million tweets are shared daily by its active users. These huge graphs have a huge storage cost and the conventional methods fail to scale to such large graphs. In order to perform analytics on large graphs, one of the practical alternatives is to compress such large graphs into a summary graph [2, 3]. A summary graph is a condensed/compressed version of the original graph that retains the essential structural properties of the original graph. The summary graph is small enough to fit in the main memory and the data analytics can be easily performed on the summary graph. Based on the analytics applied to the summary graph, the structural properties of the original graph can be estimated efficiently. However, in constructing a summary graph, it is essential to build such a summary that is a compressed replica of the original graph and the original graph can be closely reconstructed from the summary. In addition to this, the queries regarding the structure of the original graph should be efficiently answered using the summary only.

The rest of the chapter is organized as follows. In the coming two sections, we briefly define network immunization and graph summarization problems, respectively. Then, we enlist the research objectives of our work and the challenges related to the two problems. In section 1.5, we highlight our main contributions and section 1.6 presents the achievements of our work.

## 1.1 Network Immunization Problem

The problem of identifying and immunizing a subset of nodes of a given size in a network, which results in the minimal spread of malicious content in the network is known as Network Immunization Problem [4]. In the literature, it is well established that the absolute value of the largest eigenvalue of the adjacency matrix of a graph, $\lambda_{max}$, quantifies the vulnerability (the susceptibility to an external attack) of the graph [5, 6]. The goal of the network immunization problem is to select that subset of nodes for immunization which renders the graph least susceptible to virus attack. Network immunization problem is an NP-complete problem [5] as network immunization problem reduces from minimum vertex cover problem. The network immunization problem can be formally stated as follows:

**Problem 1.** *Given an undirected graph $G = (V, E)$, where $V$ and $E$ represent the set of nodes and edges in $G$ respectively, and an integer $k < |V|$, find a subset $S$ of $k$ nodes such that* immunizing *nodes in $S$, renders $G$ the least* 'vulnerable' *to a virus attack over all choices of $S$.*

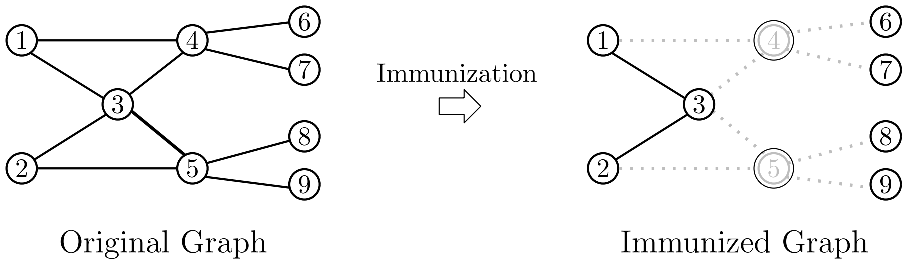


Figure 1.1: A toy example of immunization on a graph with $9$ nodes and its immunized version is shown. In this particular case, we have a budget to immunize two nodes and a node set $\{4, 5\}$ is selected for immunization. The immunized nodes will neither receive nor pass on the malicious content to other nodes and the immunization of a subset of nodes is analogous to the removal of those nodes from the graph.

## 1.2 Graph Summarization Problem

The problem of compressing a given graph into a summary graph having fewer nodes such that the original graph can be closely *reconstructed* from the summary graph is known as Graph Summarization Problem [7]. In literature, the difference between the element-wise entries of the adjacency matrices of the original graph and the graph reconstructed from the summary graph is known as *reconstruction error* and is used as a goodness measure of the summary [7]. In addition to this, accuracy in query answers related to the original graph structure based on the summary graph only is also used as the quality measure of a particular summary. The goal of the graph summarization problem is to make a summary that retains structural properties and is a close replica of the original graph. The graph summarization problem can be formally stated as follows:

**Problem 2.** *Given an undirected graph $G = (V, E)$, where $V$ and $E$ represent the set of nodes and edges in $G$ and an integer $k < |V|$, find a summary $S$ for $G$ with $k$ super nodes such that the reconstruction error of the graph reconstructed using $S$ is minimized.*

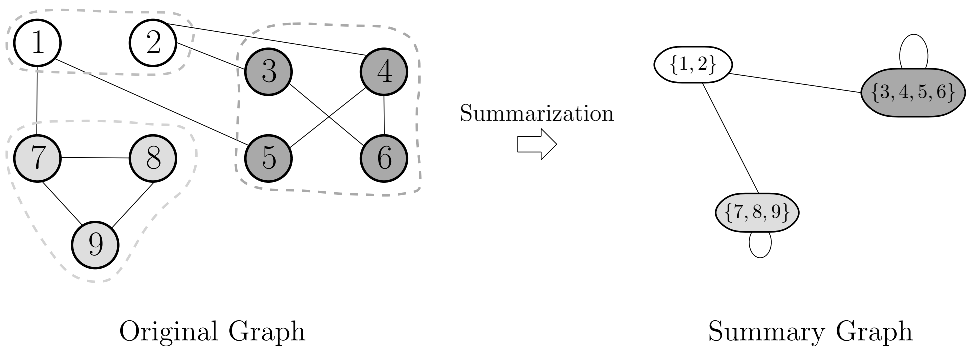


Figure 1.2: (Left) A graph having $9$ vertices and its one of the possible summaries (right) on three nodes is shown.

## 1.3 Research Objectives

In this thesis, we have the following research objectives:

1. For the network immunization problem, we aim to develop a technique to select that subset of nodes for immunization whose removal results in the maximum reduction in the vulnerability, $\lambda_{max}$, of the graph. The maximum reduction in $\lambda_{max}$ will result in the minimum spread of malicious content in the graph
2. The technique for network immunization should be scalable and efficient that can be applied to large graphs having millions of nodes
3. For the graph summarization problem, we aim to build a summary graph of a large graph such that the original graph can be closely reconstructed from the summary graph. The summary graph should also be able to accurately answer the queries related to the structure of the original graph
4. The last research objective is that the summarization algorithm is scalable and efficient so that can be applied to large graphs on which the summarization is mostly desired

## 1.4 Challenges

As already mentioned, graphs with millions of nodes and billions of edges are common in the domain of big data and the large sizes of these graphs pose computational as well as storage challenges to apply analytics on them. The main challenges of our research problems are listed below:

1. For a graph having $n$ nodes in which we want to immunize $k$ nodes, the brute force solution will explore $\binom{n}{k}$ subsets to find the *best* subset of $k$ nodes which results in maximum reduction in $\lambda_{max}$. For example, for a graph having $1000$ nodes and $10,000$ edges, and if it takes $0.00001$ second to compute the reduction in $\lambda_{max}$ by immunizing a subset of $10$ nodes, it will take ~3 billion years to select the *best* subset of $10$ nodes for immunization. This shows that finding a subset of $k$ nodes for immunization is a challenging task

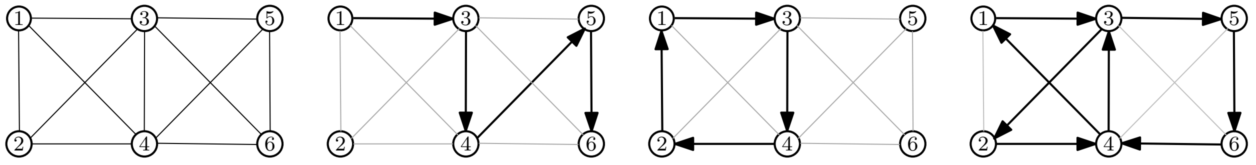
2. A common approach of graph summarization is to iteratively merge a pair of nodes in the original graph to make a summary graph [7, 8]. Let at an iteration $t$, for a graph having $n(t)$ nodes, there are $\binom{n(t)}{2}$ possible choices of pair for merging. Selecting the *best* pair of nodes in each iteration that results in a minimum increase in reconstruction error is therefore a challenging task. Moreover, the computation of error incurred after merging a pair is also computationally expensive as it amounts to the comparison of each entry of the adjacency matrix of the original and reconstructed graph which amounts to $O(n^2)$ work

## 1.5 Our Contribution

Our main contribution to network immunization and graph summarization problems are briefly described below:

### 1.5.1 Network Immunization Problem

Our solution to the network immunization problem is based upon the number of *closed walks* passing through each node in the graph. A *walk* is a sequence of edges in the graph and the number of edges in the walk is called the length of the walk. A closed walk is a walk that starts and ends at the same node. Figure 1.3 shows a walk of length $4$ and closed walks of length $4$ and $8$ in a graph having $8$ nodes. Using the facts from linear algebra, we establish that the number of closed walks passing through a node correlates with the node's contribution to the vulnerability ($\lambda_{max}$) of the graph. We define a score function based on the number of closed walks passing through a node and in our series of work, we consider closed walks of length $4, 6$ and $8$ for the selection of nodes for immunization. Let $A$ be the adjacency matrix of the graph and $A^p$ represents the $p^{th}$ power of $A$, where $p$ is a positive integer, the entry $(i.j)$ of $A^p$ shows the count of walks of length $p$ from node $i$ to node $j$. Similarly, the diagonal entries of $A^p$ represent the count of closed walks of length $p$ for respective nodes. Note that the sum of diagonal entries of a matrix, called the trace of the matrix, is equal to the sum of eigenvalues of the matrix. We use the relation that for higher values of $p$, the trace of $A^p$ becomes approximately equal to $\lambda_{max}$ of $A^p$ [9]. This implies that minimizing the trace of $A^p$ will result in the reduction of $\lambda_{max}$ of the graph. We minimize the trace of $A^p$ by deleting those nodes from the graph through which the maximum number of closed walks pass and this results in the reduction of $\lambda_{max}$ of the graph. We devise three techniques named Walk-$p, p \in \{4, 6, 8\}$ for the selection and immunization of nodes.



(a) Graph (b) Walk of length 4 {1,3,4,5,6} (c) Closed walk of length 4 $\{1, 3, 4, 2, 1\}$ (d) Closed walk of length 8 $\{1, 3, 5, 6, 4, 3, 2, 4, 1\}$

Figure 1.3: (a) A graph having $6$ nodes is shown. A walk of length $4$ on the graph is represented through bold edges in (b). Closed walks of length $4$ and $8$, where starting and ending nodes of the walks are the same are shown in (c) and (d), respectively.

### Walk-4

In our first approach of Walk-$4$, we select nodes for immunization based on the number of closed walks of length $4$ passing through them. We derive an expression that estimates the number of walks of length $4$ passing through each node and use this count as a score of nodes. We devise an approximate algorithm that iteratively selects a set $S$ of $k$ nodes for immunization. We compare our technique with the approach in which the maximum degree node is immunized in each iteration and the state-of-the-art approach, NETSHIELD [5]. Experimental results show that Walk-$4$ achieves significant quality gain (up to $20\%$ more reduction in the vulnerability of the graph) over the competitor approaches.

**Walk-6**

We devise a score function that selects nodes for immunization based on the closed walks of length $6$. The score function quantifies the contribution of nodes in $\lambda_{max}$ of the graph. Moreover, we derive a closed-form formula to compute the number of walks of length 6 passing through a node. Note that for large graphs, computing the number of walks of length $6$ passing through each node is also a computationally expensive task. To make our approach scalable and efficient, we give an approximate method that closely estimates the number of walks of length 6 passing through a node. To evaluate the goodness of our technique, we evaluate our solution on several real-world graphs. Results show that our approach maximally reduces the virus spread and $\lambda_{max}$ of the immunized graph. Moreover, our algorithm is scalable on large graphs and has a lower runtime based on the used approximation parameters.

**Walk-8**

We extend the score function based on the number of closed walks of length 8 for sets of nodes that quantify the importance of sets to reduce the graph vulnerability. This score function is monotonically non-decreasing and sub-modular which enables to greedily construct a set with improved approximation quality. We derive a closed-form formula to compute the number of walks of length 8 passing through a node that may be of independent interest. We also give an approximate method that closely estimates the number of walks of length 8 passing through a node. We evaluate the quality of our solution on several real-world graphs. We show that our approximate method is a close estimate of the exact solution. Results show that our approach maximally reduces the virus spread (the fraction of infected nodes) and the vulnerability (the largest eigenvalue) of the immunized graph. Moreover, our algorithm is scalable on large graphs and has a lower runtime based on the approximation parameters used. Comparisons also demonstrate that our approach outperforms the state-of-the-art methods both in terms of quality and runtime.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & a_{ii} & a_{ij} & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} \qquad A^p = \begin{bmatrix} z_{11} & z_{12} & \cdots & \cdots & z_{1n} \\ z_{21} & z_{22} & \cdots & \cdots & z_{2n} \\ \vdots & \vdots & \boxed{z_{ii}} & z_{ij} & \vdots \\ z_{n1} & z_{n2} & \cdots & \cdots & z_{nn} \end{bmatrix}$$

number of closed walks of length $p$ starting and ending at node $i$

Figure 1.4: (left) Adjacency matrix $A$ is shown in which an entry $a_{ij} = 1$ if there is an edge between node $i$ and $j$ else $a_{ij} = 0$. (Right) An entry $z_{ij}$ in $A^p$ represents the number of walks of length $p$ from node $i$ to $j$. Similarly, $z_{ii}$ in $A^p$ is the count of closed walks of length $p$ starting and ending at node $i$.

### 1.5.2 Graph Summarization

In graph summarization, the goal is to construct a summary graph such that the original graph can be closely reconstructed using the summary graph. We aim to minimize the reconstruction error (the element-wise difference between the adjacency matrices of the original and reconstructed graph). We devise a solution that iteratively builds the summary graph in an agglomerative fashion. In each iteration, we merge that pair of nodes from a logarithmic-sized sample of nodes such that there is a minimum increase in the reconstruction error. For each pair of nodes in the sample, we define a score that quantifies the error introduced by merging that pair. The score is efficiently approximated using a closed-form expression to compute the error introduced after merging the pair and by storing extra information at nodes. Moreover, each node is assigned a weight that estimates the contribution of the node in the score of pairs of nodes containing it. The logarithmic-sized sample of nodes for merging is selected based on node weights and is of better quality (based on reconstruction error and accuracy in query answers) as compared to a random sample of linear size. Based on the above-mentioned steps, our algorithm builds a summary efficiently that is of better quality as compared to the competitor techniques, GRASS [7] and S2L [10]. Using our solution, the original graph can be reconstructed from the summary graph with minimum error and the queries related to graph structure can be accurately answered using the summary graph only.

## 1.6 Our Achievements

The main achievements of our work are briefly listed as follows:

1. Our proposed solution of network immunization is of better quality as compared to the state-of-the-art solution, NETSHIELD [5]. We evaluate the quality of our method based on two measures. First, we report the eigen drop percentage which shows the reduction in the vulnerability $\lambda_{max}$ of the immunized graph. We achieve upto $30\%$ higher eigen drop percentage as compared to the competitor method. Secondly, we report the rate of infection spread in a graph immunized by our approach and the state-of-the-art method. Results show that the graph immunized by our method has up to $20\%$ fewer infected nodes as compared to that of the competitor method. A higher eigendrop percentage justifies the minimum spread of the virus in the immunized graph

2. Our method of network immunization is more efficient than the competitor method, NETSHIELD. We compare the runtime (in seconds) of both the methods which shows that our approach is fast as compared to the competitor method and achieves high eigendrop in relatively less time

3. Our approach to make summary graph of a given graph is of better quality as compared to the competitor methods, GRASS [7] and S2L [10]. We measure the quality of the summary graph based on reconstruction error and error in the accuracy of query answers related to graph structure based on the summary only

4. Our summarization algorithm is more scalable and efficient as compared to GRASS, which only works for moderate graphs with a few hundred nodes and can not be applied to large

graphs on which our method takes a few hundred seconds to compute the summary

## 1.7 Thesis Organization

The thesis is organized as follows. In chapter 2, we give background and literature of two problems: network immunization and graph summarization. In chapter $3$, we describe our initial work on the network immunization problem in which we solved the problem by incorporating the closed walks of length $4$. In chapter $4$, the problem of network immunization is discussed with reference to walks of length $6$ and the approximate method to count those walks. We give our final contribution to the immunization problem by extending our approach to walks of length $8$ in chapter $5$. After that, in chapter $6$ we describe our contribution to the graph summarization problem and we give the conclusion and our future work in chapter $7$.

# Chapter 2

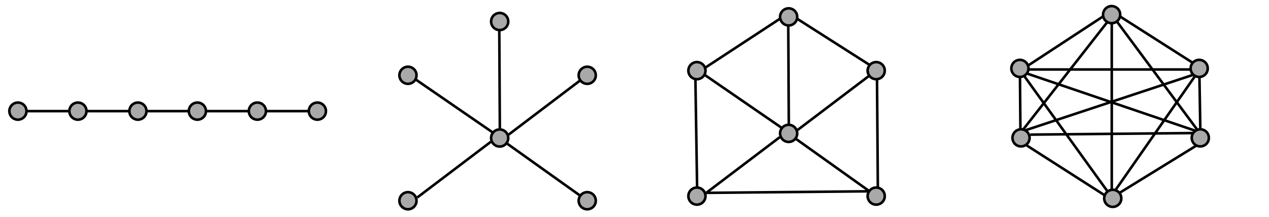
# Background and Literature Review

In this chapter, we describe the related work of the two problems mentioned in Chapter 1. First, we discuss the background of network immunization problem, then we discuss the related work of graph summarization problem.

## 2.1 Network Immunization

Information spread in networks is widely studied in epidemiology, sociology and information sciences. In the literature, there are two main models of information spread [11] across the graphs which are $i)$ independent cascade and $ii)$ linear threshold model. In the independent cascade model, each node $u$ gets exactly one chance to propagate the information to its neighbors. The node $u$ can pass the information to its neighbor with a given probability $p$. In the linear threshold model, each node has some activation probability and a node will pass information to its neighbors if its activation probability is greater than a given threshold value. Researchers are usually interested in estimating the extent to which a contagion will affect the population, predicting the timeline of infection and methods for containing or limiting the effect. The spreading process is studied on a network: agents are represented by nodes and the potential spread of information between a pair of agents is modeled by the presence of an edge between the corresponding pair of nodes. Information spread has also been discussed in the context of structural virality [12], which is the average distance between all pairs of nodes in a diffusion tree T. They have also discussed two modes of information spread. In the broadcast spread, one node spreads information to a large number of the population while in viral propagation, a hierarchy of nodes progressively shares information with a small subset of nodes. In the literature, there are three common virus spread models [13]: $i)$ Susceptible-Infected (SI) model in which a susceptible node can be infected from a virus with some defined probability. Once a node becomes infected, it can not recover and will remain infected afterwards. $ii)$ Susceptible-Infected-Susceptible (SIS) model assumes that a healthy node can receive the infection. However, after some time the infected node can cure from the infection with some probability and will again become susceptible. $iii)$ Susceptible-Infected-Recovered (SIR) model in which a susceptible node can be infected by virus. However, after some time the infected node can be recovered from the infection. A node, once recovered, will neither get infection again nor transmit the infection.

Popular models assume the knowledge of an infection rate $\beta$ (the rate at which an individual/agent accepts content from its neighbors) and a rate of recovery $\delta$ (the rate at which an individual/agent loses content) [6]. A relation between the spread rate of the virus and the largest eigenvalue of adjacency matrix $A$ of the graph, $\lambda_{max}(A)$, was established in [6, 14]. The higher the largest eigenvalue, the more vulnerable the graph is. In fact, the largest value shows the connectivity among the graph. A higher eigenvalue shows that a graph is more connected as shown in Figure 2.1.



Path graph, $P_6$, $\lambda_{max} = 1.8$ Star graph, $S_6$, $\lambda_{max} = 2.23$ Wheel graph, $W_6$, $\lambda_{max} = 3.34$ Complete graph, $K_6$, $\lambda_{max} = 5$

Figure 2.1: Various types of graphs with an equal number of nodes are shown in which the line graph with the minimum number of edges (connectivity) has the minimum largest eigenvalue ($\lambda_{max} = 1.8$) and the complete graph has the maximum largest eigenvalue ($\lambda_{max} = 5$). This shows that the vulnerability of a graph increases with the number of edges (connectivity) in the graph.

It has been established that $\lambda_{max}$ has been bounded by the average degree of nodes, $deg_{avg}$, in the graph and the maximum degree of a node, $\Delta$, in the graph.

$$deg_{avg} \leq \lambda_{max} \leq \Delta$$

However, deleting the maximum degree nodes in a graph may not result in the maximum reduction in $\lambda_{max}$ of the adjacency matrix of the graph. For example in the graph shown in Figure 2.2, the maximum reduction in $\lambda_{max}$ is achieved by removing node $f$ and not by node $n$ having the maximum degree.

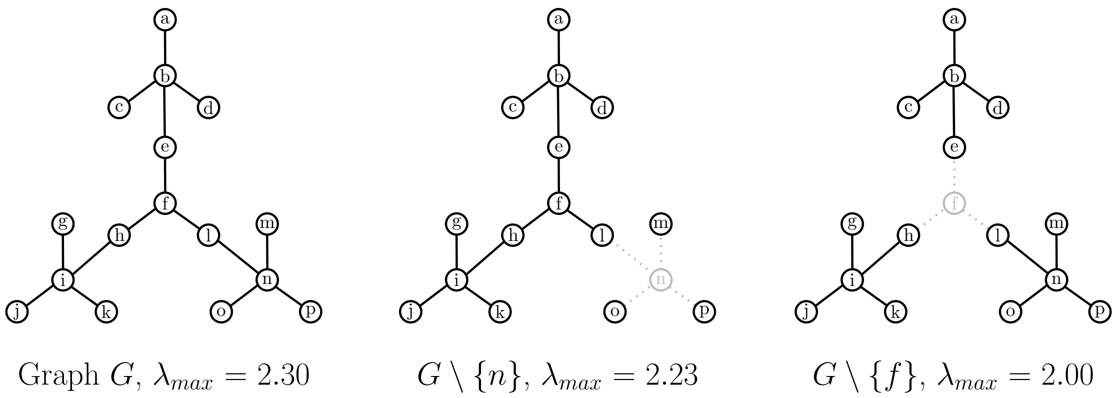

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
Graph $G$, $\lambda_{max} = 2.30$
$G \setminus \{n\}$, $\lambda_{max} = 2.23$
$G \setminus \{f\}$, $\lambda_{max} = 2.00$

Figure 2.2: A graph $G$ has $\lambda_{max} = 2.30$. $G \setminus \{n\}$, the resulting graph after the removal of maximum degree node $n$ from $G$, has $\lambda_{max} = 2.23$ while the removal of node $f$ results in $\lambda_{max} = 2.00$. This shows that the removal of the maximum degree node from a graph does not always result in the maximum reduction in the vulnerability ($\lambda_{max}$) of the graph.

Moreover, there are also other centrality-based measures that are utilized in the literature to quantify the vulnerability of the graph or the role of a particular node in infection spread. Degree centrality, closeness centrality and betweenness centrality are some of these measures but these centrality-based measures have not been proven to be very accurate [5].

In particular, [6] assumes discrete time and at each time stamp, a node $u$ can receive infection from its infected neighbor $i$ with probability $\beta$. Moreover, at each time stamp, $i$ can recover with probability $\delta$. The model shows that an epidemic dies out in sub-linear time with respect to the size of the population following a stochastic model if $\beta/\delta < 1/\lambda_{max}(A)$. The model also shows that the epidemic threshold $\tau$, state beyond which infection becomes endemic, is $1/\lambda_{max}(A)$. The model works for various types of graphs including Erdos-Reyni, homogeneous and power-law graphs. The relation of graph structure and the spread of disease is studied in [14] which considers various graph topologies including complete, star and Erdos-Reyni graphs. Similarly, an exponential lower bound on expected die-out time or time for full network recovery (i.e. $\geq e^{cN}$ where $c$ is a constant dependent on the infection rate and $N$ is the size of population) is also known when $\beta > \delta/\lambda_{max}(A)$ [15, 16]. Recent works of [17–19] established a similar relation of infection and recovery rates with $\lambda_{max}$ for infection spread or die-out while approximating the stochastic model by a deterministic one.

Various studies have proposed preemptive methods to control virus spread and avoid a potential outbreak of contagion. These methods remove a subset of nodes or edges from the graph, so the remaining graph has the lowest $\lambda_{max}$. This problem has been shown to be NP-COMPLETE in [5,20]. NETSHIELD is a greedy approach to iteratively select a subset of nodes for immunization [5]. NETSHIELD addresses three problems related to network immunization, The paper considers $\lambda_{max}$ as the vulnerability of the graph. Secondly, the paper defines a *shield value* for a subset of nodes that quantifies the benefit achieved in terms of reduction in $\lambda_{max}$ after immunizing the subset of nodes. Thirdly, the greedy algorithm selects nodes for immunization based in the values eigenvector corresponding to $\lambda_{max}$ of the graph. The greedy method also caters that the immunized nodes are well-spread in the graph. The paper also proposes an improved version, NETSHIELD+, that selects nodes for immunization in batches.

A combinatorial trace method is adopted in [21–23] to select a subset of nodes whose removal will result in the maximum reduction in the $\lambda_{max}$. The trace of a large power of an adjacency matrix, $A^p$, is closely related to the $\lambda_{max}(A)$ (also known as the spectral radius of the graph) [24]. Trace of $A^p$, on the other hand, is just the count of the number of closed walks of length $p$ in the graph. Approximation algorithms are given in [21, 23] to select nodes for removal to eliminate the most number of closed walks of length $4$ from the graph. Approximation of the number of closed walks of length $6$ containing a node using a randomly constructed summary of a graph is given in [22]. In this paper, we extend this work by considering walks of length $8$ that leads to improved quality. We note that more sophisticated techniques for graph summarization [7, 10, 25, 26] and membrane computation [27–29] could be utilized to improve this work.

Edge removal techniques are also devised to minimize graph vulnerability. Methods for selecting edges whose removal will reduce $\lambda_{max}$ the most are devised in [30, 31]. In [30] virus spread is modeled by the dynamical system and the transition function which defines the interaction of a node with its neighbors and the state of each node (healthy or infected) in order to reduce $\lambda_{max}$. Two edge manipulation techniques, *NetMelt* and *NetGel* have been devised in [31]. *NetMelt* aims to contain the information spread by selecting edges whose removal results in the maximum reduction in $\lambda_{max}$. The edges for deletion are selected based in the score computed from the left and right eigenvectors corresponding to the $\lambda_{max}$ of the original graph. On the other hand, *NetGel* recommends the addition of new edges in the graph which will result in maximum gain in information dissemination across the graph.

In another line of work, non-preemptive techniques are devised in [4, 32, 33] to immunize select nodes after the virus spread has started and the healthy and infected nodes are known. In this setting, methods are evaluated by *save ratio (SR)*: the ratio of the number of affected nodes in a graph when $k$ nodes are immunized to the number of infected nodes in case of no immunization. A reverse engineering technique is used to identify the nodes in a graph where the virus spread is initiated [34]. A related problem is to decontaminate the graph by deploying *cleaning agents* at certain nodes that travel along edges. Monotonicity is assumed in [35–38] that a node cleaned by the agent will not get affected again. Non-monotonic strategies are given in [39].

Some other problems related to graph immunization include the influence maximization [40], the filter placement [41] and the critical node detection problem (CNDP) [42–44]. In the influence maximization problem, the goal is to find a subset of nodes whose *activation* will lead to the maximal spread of information across the graph. The filter placement problem deals with minimizing the multiplicity of information flowing across the network. In CNDP, the goal is to identify nodes whose removal results in maximum graph fragmentation.

## 2.2 Graph Summarization

Graph summarization and compression have been studied for a wide array of problems and have applications in diverse domains [2, 45, 46]. It is widely used in clustering, classification, community detection, outlier identification, network immunization, etc. In literature, there are two main types of graph summarization methods: *lossless* and *lossy*. In lossless graph summarization, the original graph is exactly reconstructed form the summary graph and there is no loss of information in the graph reconstructed from the summary. The exact reconstruction is done by storing extra information, known as *edge corrections*, along with the summary graph [47]. Edge corrections are used to insert missing edges or delete the extra edges in the reconstruction of the graph from the summary graph. The edges that need to be added after reconstructing the graph are termed as *positive edge corrections* and the edges that need to be deleted after reconstructing the graph from the summary are called *negative edge corrections*.A scalable summarization approach summarizes sets of similar nodes that are found using locality sensitive hashing [8]. In lossless graph summarization, the goal is to create such a summary such that minimum information should be stored as edge corrections.

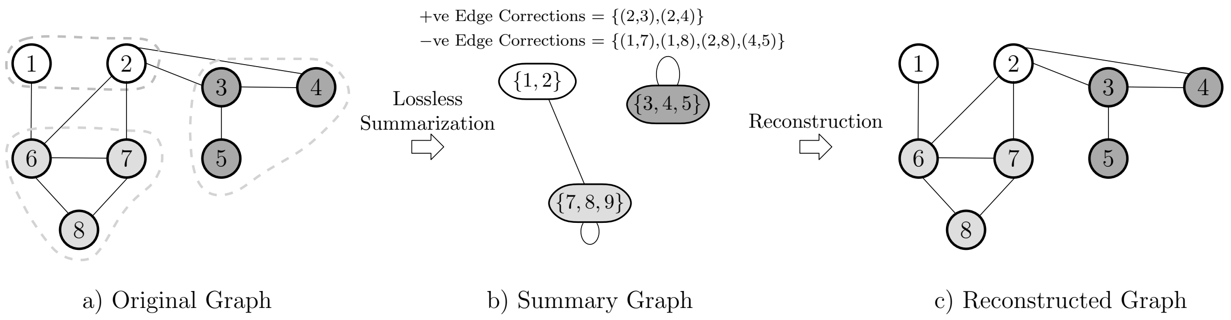


Figure 2.3: a) A graph on $9$ vertices and b) its lossless summary are shown. In lossless summarization, edge corrections are stored along with the summary graph such that the original can be exactly reconstructed form the summary graph. c) The reconstructed graph which is same as the original graph is also shown.

In lossy graph summarization, there is no guarantee of exact reconstruction of the original graph from the summary graph [8, 47]. In lossy summarization, some details of the original graph are compromised and resultantly, some error is introduced in the reconstructed version of the graph.

Reconstruction error [7], cut-norm error [10] and error in query answering are some of the widely used quality measures of a lossy summary. Reconstruction error is the norm of the error matrix (difference between the adjacency matrices of the original and the reconstructed version of the graph) [7]. Cut-norm error is defined as the maximum absolute difference between the sum of the weights of the edges between any two sets of vertices [10]. Similarly, accuracy in answers to various types of graph queries indicates the quality of the summary. For partitioning nodes into supernodes, an agglomerative approach is used in [7] that greedily merges pairs of nodes to minimize the $l_1$-reconstruction error. This approach is very simple and achieves great summarization quality, but it does not scale to large graphs. Even after subsampling [7], the approach scales only to graphs of order of a few thousand. An efficient algorithm that uses a weighted sampling scheme was proposed in [48] that can be applied to large-scale graphs.

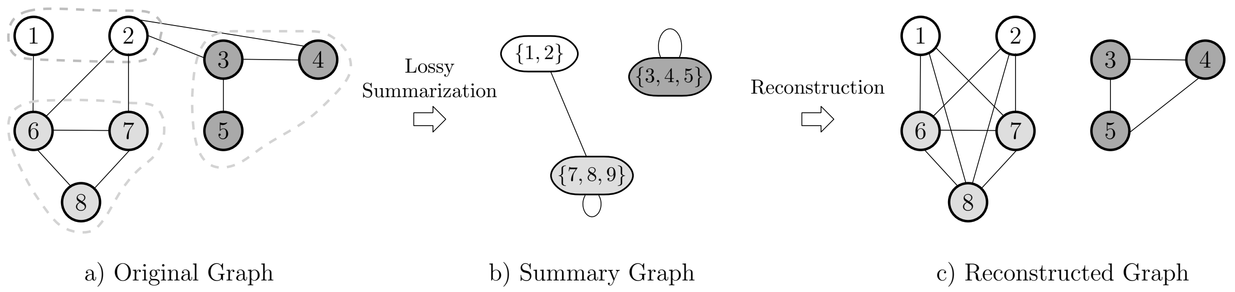


Figure 2.4: a) A graph having $9$ vertices and b) its lossy summary are shown. In lossy summarization, some details of the original graph are compromised and some error is introduced in the reconstructed version of the graph as shown in c).

Note that various types of graphs exist in different domains. Summarization on different types of graphs is applied to get useful results. In attributed graphs, nodes have certain associated attributes (properties) [49]. The lossless summarization of attributed graphs is described in [50], which identifies the sets of nodes having *similar* neighborhood and attribute values for merging. Locality sensitive hashing is used to select nodes having similar neighborhood and attribute values. Moreover, to construct a summary with approximate homogeneous neighborhood information and attribute values using an entropy model is described in [51]. In addition to this, the summarization of attributed graphs based on a selected set of attributes is described in [52]. The work also presents an operation to allow users to *drill down* to a larger summary for more details and *roll up* to a concise summary with fewer details. Another line of work for attributed graphs finds a compact subgraph of the desired number of nodes having query attribute values [53]. A survey [46] discusses various summarization techniques for attributed graphs.

Weighted graphs have weights on nodes or edges. Compression of edge weighted graphs using locality sensitive hashing while preserving the edge weights is described [54]. Furthermore, compression of node and edge weighted graphs is described such that the weights on the path between two nodes in the summary graph should be similar to that in the original graph [55]. The paper also aims to preserve more information related to nodes with high weights. Another closely related area is of influence graph in which nodes have influence over other nodes. The influence graph summarization takes into account the influence of nodes on other nodes in the summary graph [56].

Dynamic graphs, where nodes or edges evolve over time, are also prevalent these days. An approach discusses the summarization of a dynamic graph based on connectivity and communication among nodes [57]. The work creates summaries of the dynamic graph over a sliding window of fixed size. MoSSo, a lossless approach, incrementally updates the summary in response to the deletions or additions of edges [58]. A summarization framework that captures the dynamic nature of dynamic graphs is described in [59].

Compression of web graphs is used to improve the performance of search engines [2,45]. The web graphs are compressed efficiently by exploiting the link structure of the web [3]. Permuting the nodes in a web graph such that *similar* nodes are placed together produces improved compression results [60]. Parallel methods are also devised to summarize massive web graphs spread across multiple machines [61].

Summarization of graph streams such that to approximately answer the queries on the graph stream is discussed in [62]. The real-time summarization of massive graph streams is done using the count-min sketch approach to preserve the structural information of the graph [63].

Several methods have been proposed to summarize graphs, where some features of graphs are used as building blocks (vocabulary), and the graph is then represented using the vocabulary. *VoG* (Vocabulary-based summarization of graphs) summarizes graphs based on the substructures like cliques, chains, stars, and bipartite cores [64]. Graph compression based on communities identified on the basis of central nodes and hubs is studied in [65]. A comprehensive survey compares different graph summarization techniques [66].

# Chapter 3

# Foundations for network immunization problem

## 3.1 Introduction

Graphs or networks are used to model many practical scenarios involving pairwise interactions between entities. The entities could be humans, computers, mobile devices, power components, etc. while interactions can be face-to-face meetings, email and SMS communication and various kind of flows e.g. electric current in a power infrastructure network or fluid in pipelines. Many of the practical networks are very large with millions of nodes and edges.

Every interaction in such large networks can not be monitored and there is a possibility of undesired and potentially harmful communication taking place among entities in networks. Such undesired spread could be intentional or un-intentional entailing various degrees of harm. The unintentional spread of flu-virus, for instance, may be life-threatening and may cause an epidemic. A rumor, on the other hand, may well be originated intentionally and its effect might be limited to a particular segment of a network. An effective way to safeguard a network against the spread of malicious content is to *empower* the nodes. The strengthening process may amount to vaccinating people, deploying surveillance systems at junctures and installing anti-virus software on computers depending on the underlying network. The nodes with these added capabilities will be referred to as the *immunized* nodes and the malicious content, as the *virus*. Effectively, when a node is immunized, it will neither get contaminated nor will it pass the contaminant to other nodes.

There is a cost associated with immunization, hence it is not feasible to immunize all nodes in large networks. The problem to select a subset of nodes (not exceeding a given budget) for immunization that will maximally hinder the virus spread is called the *Network Immunization Problem* and is abstractly formulated in [5] as follows:

**Problem 3.** *Given an undirected graph $G = (V, E)$, $|V| = n$ and an integer $k < n$, find a subset $S$ of $k$ nodes such that* immunizing *nodes in $S$, renders $G$ the least* 'vulnerable' *to a virus attack over all choices of $S$.*

This requires a quantitative measure for the vulnerability of the graph. As in the literature [5, 21, 22, 67], we use the largest eigenvalue of the adjacency matrix of the graph to quantify the graph vulnerability. The objective in Problem 6, therefore becomes that of immunizing a fixed-sized subset so as the remaining graph has the minimum largest eigenvalue. More precisely,

**Problem 4.** *Given an undirected graph $G = (V, E)$, $|V| = n$ and an integer $k < n$, find a subset $S$ of $k$ nodes such that the largest eigenvalue of the adjacency matrix of $G - S$ (the matrix after removing the rows and columns corresponding to $S$) is minimum over all choices of $S$.*

A score function was proposed in [21, 22], for a subset of nodes based on the number of small length closed walks a node is contained in. The number of fixed length closed walks containing a node co-relates with the node's contribution towards the largest eigenvalue of the adjacency matrix of the graph. However, while the longer walks provide a better approximation, only shorter walks were considered due to time complexity. In our series of work, we propose randomized approximation approaches based on the number of closed walk passing through a node to immunize graphs. The contribution of our work can be summarized as follows:

- We give the score function based on the number of closed walks of length $p \in \{4, 6, 8\}$ for sets of nodes that quantify the importance of sets to reduce the graph vulnerability. This score function is monotonically non-decreasing and sub-modular that enables employing greedily constructing a set with improved approximation quality

- We derive closed-form formulae to compute the number of walks of length $p \in \{4, 6, 8\}$ passing through a node which may be of independent interest. Note that computation of count of closed walks is a computationally expensive task, so, we also give an approximate method that closely estimates the number of walks of length $6$ and $8$ passing through a node

- We evaluate the quality of our solution on several real-world graphs. We show that our approximate method is a close estimate of the exact solution. Results show that our approaches maximally reduces the virus spread and the vulnerability (the largest eigenvalue) of the immunized graph. Moreover, our algorithm is scalable to large graphs and has a lower runtime based on the approximation parameters used. Comparisons also demonstrate that our approach outperforms the state-of-the-art methods both in terms of quality and runtime

## 3.2 Preliminaries

In this section, we formulate the immunization problem. Given a simple graph $G = (V, E)$, the goal is to select a subset $S$ of $k$ nodes such that removing $S$ from the graph maximally reduces the largest eigenvalue of the remaining graph denoted by $\lambda_{max}(A|_{-S})$. Since $\lambda_{max}$ can be computed in $O(|E|)$, the optimal subset of nodes can be found by iterating through each of the $\binom{n}{k}$ subsets. The overall runtime of this brute force algorithm is $O(\binom{n}{k} \cdot |E|)$ rendering it computationally infeasible even for moderately large graphs.

Indeed, it turns out that Problem 4 is NP-HARD. A reduction from *Minimum Vertex Cover Problem* is as follows: if there exists a set $S$ with $|S| = k$ such that $\lambda_{max}(A|_{-S}) = 0$, then $S$ is a vertex cover of the graph. It follows from the below implication of famous *Perron-Frobenius theorem*:

**Fact 1.** *Deleting an edge from a simple connected graph G strictly decreases the largest eigenvalue of the corresponding adjacency matrix [68].*

Also, if there is a vertex cover $S$ of the graph such that $|S| = k$, then deleting $S$ will result in an empty graph which has eigenvalue zero.

Although Problem 4 is NP-HARD, its objective function is monotone and sub-modular. The greedy algorithm (GREEDY-1) guarantees $(1 + 1/e)$-approximation ($e$ is the base of the natural logarithm) to Problem 4 by Theorem 1.

**Theorem 1.** *[69] Let $f$ be a non-negative, monotone and submodular function, $f : 2^{\Omega} \to \mathbb{R}$. Suppose $\mathcal{A}$ is an algorithm, that chooses a $k$ elements set $S$ by adding an element $u$ at each step such that $u = \arg\max_{x \in \Omega \setminus S} f(S \cup \{x\})$. Then $\mathcal{A}$ is $(1 + 1/e)$-approximate algorithm.*

**Algorithm 1** : GREEDY-1 ($G$,$k$)

```
$S \leftarrow \emptyset$
while $|S| < k$ do
    $v \leftarrow \arg\min_{x \in V \setminus S} (\lambda_1(A_{-\{S \cup \{x\}\}}))$
    $S \leftarrow S \cup \{v\}$
return $S$
```

We refer to the achieved benefit after immunizing subset $S$ as *eigendrop* and is defined as $\lambda_{max}(A) - \lambda_{max}(A|_{-S})$. A score, termed as shield-value, is assigned to each subset $S \subset V$, which quantifies the approximated eigendrop achieved after removing $S$. Frequently used symbols in the paper are listed in Table 3.1.

Table 3.1: List of Symbols

| **Symbol** | **Definition & Description** |
|---|---|
| $A$ | Adjacency matrix of the graph $G$ |
| $G\|_{-S}$ | Subgraph after removing node set S from the graph $G$ |
| $A\|_{-S}$ | Adjacency matrix of the graph $G\|_{-S}$ |
| $\lambda_i(A)$ | $i^{th}$ largest eigen value of matrix $A$ on the basis of magnitude |
| $\lambda_{max}(A)$ | The largest eigen value of matrix $A$ i.e. $\lambda_{max}(A) = \lambda_1(A)$ |
| $\Delta\lambda(S)$ | $\lambda_{max}(A) - \lambda_{max}(A\|_{-S})$; eigendrop achieved by immunizing node set $S$ |
| $A^p$ | $p^{th}$ power of (adjacency) matrix A |
| $\mathcal{CW}_p(v, G)$ | The set of $p$-length closed walks in $G$ containing $v$ |
| $\mathcal{CW}_p(S, G)$ | The set of $p$-length closed walks in $G$ containing at least one vertex from $S$ |
| $\mathcal{W}_p(v, G)$ | Number of $p$-length closed walks in $G$ containing $v$ |
| $\mathcal{W}_p(S, G)$ | Number of $p$-length closed walks in $G$ containing at least one vertex from $S$ |
| $d_G(v)$ | Degree of node $v$ in graph $G$ |

## 3.3 Proposed shield value

In this section, we quantify the importance of a subset of nodes for immunization. We first derive a score for each set of size $k$ that closely measures the value of the objective function of Problem 4. We prove that this score function is monotonically increasing and submodular. Using Theorem 1, we can greedily build up the set $S$ by iteratively selecting nodes that are contained in the maximum number of closed walks of length $p$.

Let $A$ be an $n \times n$ matrix; the following two fundamental results from algebraic graph theory [9, 70, 71] relate the eigen spectrum and the trace of $A$.

**Fact 2.**

$$trace(A) = \sum_{i=1}^{n} A(i,i) = \sum_{i=1}^{n} \lambda_i(A)$$

**Fact 3.**

$$trace(A^p) = \sum_{i=1}^{n} \lambda_i(A^p) = \sum_{i=1}^{n} (\lambda_i(A))^p$$

From the theory of vector norms [9] and Fact 3 we know that

$$\begin{aligned}\lim_{\substack{p\to\infty \\ p \text{ even}}} (trace(A^p))^{1/p} &= \lim_{\substack{p\to\infty \\ p \text{ even}}} \left(\sum_{i=1}^{n} \lambda_i(A)^p\right)^{1/p} \\ &= \lim_{p\to\infty} \left(\sum_{i=1}^{n} |\lambda_i(A)|^p\right)^{1/p} = \max_i\{\lambda_i(A)\} = \lambda_{max}(A)\end{aligned}$$

Using the above relation we establish that for the immunization problem, we want to find a subset $S$ of nodes in graph $G$ which, when removed, minimizes $trace((A|_{-S})^p)$. Next, we derive a combinatorial form of this objective function.

As described in Table 3.1, for a vertex $v \in V(G)$, $\mathcal{CW}_p(v, G)$ is the set of all closed walks of length $p$ in the graph $G$ containing $v$ at least once and $\mathcal{W}_p(v, G) = |\mathcal{CW}_p(v, G)|$. Similarly, $\mathcal{CW}_p(S, G)$ denotes the set of closed walks of length $p$ in $G$ containing at least one vertex from $S$ and correspondingly $\mathcal{W}_p(S, G) = |\mathcal{CW}_p(S, G)|$. We use the following combinatorial definition of $trace$.

**Fact 4.** *[70] Given a graph $G = (V, E)$ with adjacency matrix $A$,*

$$\mathcal{W}_p(V, G) = trace(A^p)$$

From Fact 4 and definition of trace (Fact 2), we get that

$$\mathcal{W}_p(V, G) = \mathcal{W}_p(V \setminus S, G|_{-S}) + \mathcal{W}_p(S, G) \tag{3.1}$$

This is true because any walk in $G$ either contains some vertex in $S$ or it does not contain any vertex in $S$. The former type of walks are counted exactly once in the term $\mathcal{W}_p(S, G)$, while the first term counts closed walks of the latter type. Equation (3.1) can be equivalently rewritten as

$$\begin{aligned} trace(A^p) =& trace((A|_{-S})^p) + \mathcal{W}_p(S, G) \\ \implies& trace((A|_{-S})^p) = trace(A^p) - \mathcal{W}_p(S, G) \end{aligned}$$

Thus for a fixed graph $G$ (since $trace(A^p)$ is constant) minimizing $trace((A|_{-S})^p)$ is equivalent to maximizing $\mathcal{W}_p(S, G)$. This implies that the set $S$ with the largest value of $\mathcal{W}_p(S, G)$ will yield the maximum eigendrop. Intuitively, we need to identify nodes contained in many closed walks of length $p$ (nodes with high $\mathcal{W}_p(v, G)$). We define the following shield value of a set $S$, that in addition to maximizing $\mathcal{W}_p(S, G)$, attempts to select those nodes which are far from each other i.e. having $A(u, v) = 0$ in order to maximize the number of distinct closed walks going through nodes in a set $S$.

$$score_p(S) = \gamma \sum_{v \in S} \mathcal{W}_p(v, G)^2 - \sum_{u,v \in S} \mathcal{W}_p(v, G) A(u, v) \mathcal{W}_p(u, G), \tag{3.2}$$

where $\gamma$ is a positive constant. Hence Problem 4 can be rephrased as follows.

**Problem 5.** *Let $G = (V, E)$ be an undirected graph on $n$ nodes and let $k$ be an integer $k < n$, find a subset of nodes $S \subset V$, with $|S| = k$ such that $score_p(S)$ is the maximum over all $k$-subsets of $V$.*

For fixed $p$, given $\mathcal{W}_p(v, G), \forall v \in V$, $score_p(S)$ can be evaluated in time $O(k^2)$ . Selecting a set with maximum $score_p(S)$ takes $O(\binom{n}{k} k^2)$ time which clearly is computationally prohibitive.

Furthermore, note that for this we need to have the values of $\mathcal{W}_p(v, G)$ pre-computed, which is not straight-forward.

We show that the objective function of Problem 5 is monotone and sub-modular. Given $\mathcal{W}_p(v, G)$, by Theorem 1, the greedy strategy for building up the set will yield $(1-1/e)$-approximation
760 of the optimal subset.

**Theorem 2.** *For* $p \geq 1$, $score_p(S)$ *is monotonically non-decreasing.*

*Proof.* We prove that for any $X \subset Y \subseteq V$, $score_p(X) \leq score_p(Y)$. Let $E, F \subset V(G)$ and $x \in V(G)$ such that $F = E \cup \{x\}$. Consider

$$\begin{aligned} & score_p(F) - score_p(E) \\ =&\gamma \sum_{v \in F} \mathcal{W}_p(v)^2 - \sum_{u,v \in F} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) - \gamma \sum_{v \in E} \mathcal{W}_p(v)^2 \\ & + \sum_{u,v \in E} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) \\ =&\gamma \mathcal{W}_p(x)^2 - \sum_{v \in E} \mathcal{W}_p(v) A(x,v) \mathcal{W}_p(x) = \mathcal{W}_p(x)[\gamma \mathcal{W}_p(x) - \sum_{v \in E} \mathcal{W}_p(v) A(u,v)] \geq 0 \end{aligned}$$

Since $\gamma > 0$, for $\gamma \geq k \max_{v \in V(G)}\{\mathcal{W}_p(v)\}$, the last inequality is satisfied. Hence, $score_p(S)$ function is monotonically non-decreasing. □

**Theorem 3.** *For* $p \geq 1$, $score_p(S)$ *is submodular.*

*Proof.* For any subsets $X, Y$, with $X \subset Y \subseteq V$ and a subset $Z \subset V$ such that $Z \cap Y = \emptyset$, we have $score_p(X \cup Z) - score_p(X)$ is at least as large as $score_p(Y \cup Z) - score_p(Y)$. Let $I, J, K \subset V(G)$

with $I \subset J$. We have

$$\begin{aligned}
&score_p(I \cup K) - score_p(I) - score_p(J \cup K) + score_p(J) \\
=&\Big(\gamma \sum_{v \in I \cup K} \mathcal{W}_p(v)^2 - \sum_{u,v \in I \cup K} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) - \gamma \sum_{v \in I} \mathcal{W}_p(v)^2 \\
&+ \sum_{u,v \in I} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u)\Big) - \Big(\gamma \sum_{v \in J \cup K} \mathcal{W}_p(v)^2 - \sum_{u,v \in J \cup K} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) \\
&- \gamma \sum_{v \in J} \mathcal{W}_p(v)^2 + \sum_{u,v \in J} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u)\Big) \\
=&\Big(\gamma \sum_{v \in K} \mathcal{W}_p(v)^2 - \sum_{u,v \in K} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) - 2 \sum_{u \in K, v \in I} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u)\Big) \\
&- \Big(\gamma \sum_{v \in K} \mathcal{W}_p(v)^2 - \sum_{u,v \in K} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) - 2 \sum_{u \in K, v \in J} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u)\Big) \\
=&2 \sum_{u \in K, v \in J} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) - 2 \sum_{u \in K, v \in I} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) \\
=&2 \sum_{u \in K, v \in J \setminus I} \mathcal{W}_p(v) A(u,v) \mathcal{W}_p(u) \geq 0
\end{aligned}$$

□

In the coming chapters, we compute $score_p(S)$ based on the closed walks of length $p \in \{4, 6, 8\}$. We note that computing count of closed walks for larger values of $p \in \{6, 8\}$ is a computationally expensive. To address this challenge, we also give an approximate method based on hypergraph to estimate the number of closed walks of length $6$ and $8$. We also give experimental results on real world graphs for each of the proposed approach.

# Chapter 4

# Spectral Methods for Immunization of Large Networks

In this chapter, we give solution to network immunization problem (Problem 4 mentioned in Chapter 3) based on the number of closed walks of length 4, $\mathcal{W}_4(v, G)$ passing through node $v$ in graph $G$. First, in Section 4.1, we give a closed form expression to compute $\mathcal{W}_4(v, G)$. Then, we devise an algorithm to select nodes for immunization in Section 4.2. Experimental evaluation of the goodness of our solution is presented in Section 4.3. In the end, we give a brief summary of our approach and findings.

## 4.1 Computation of $\mathcal{W}_4(v, G)$

For a graph $G = (V, E)$, a walk $W_l$ of length $l$ in $G$ is a sequence of nodes $v_0, v_1, \ldots, v_l$ such that for $0 \leq i \leq l - 1$ $(v_i, v_{i+1}) \in E$. We say that $W_l$ is a walk from $v_0$ to $v_l$. If $v_0 = v_l$, then $cW_l$ is called a closed walk. We compute score of a node $v$ based on closed walks of length $4$ represented as $\mathcal{W}_4(v, G)$.

First, we give a closed form expression for $\mathcal{W}_4(v, G)$ in terms of degrees and codegrees. For a given graph $G$, $N_G(x) = \{y \in V : A(x, y) = 1\}$ and $d_G(x) = |N_G(x)|$. Define $N_G(x, y) = \{z \in V : A(x, z) = 1 \wedge A(y, z) = 1\}$ to be the common neighborhood of $x$ and $y$ in $G$. Let

$d_G(x,y) = |N_G(x,y)|$, note that $N_G(x,x) = N_G(x)$ and $d_G(x,x) = d_G(x)$. When $G$ is clear in the context, we refer to $N_G(x,y)$ as $N(x,y)$ and similarly to $d_G(x,y)$ as $d(x,y)$.

**Lemma 4.** *For any node $v \in V$,*

$$\mathcal{W}_4(v,G) = 2d(v)^2 + 4 \sum_{u \in V, u \neq v} d(u,v)^2.$$

*Proof.* A closed walk $W : (v,x,u,y,v)$ of length $4$ can be interpreted as the concatenation of two walks of length $2$ with the same endpoints $u$ and $v$. The number of walks of length $2$ with the endpoints $u$, $v$ is $A^2(u,v)$. We want to count the closed walks of length $4$ that contain a fixed node $v$ at least once. Note that $v$ can occur at most twice in a closed walk of length $4$. In a closed walk of length $4$, there are $4$ positions for vertices (since the first and last node is the same, we consider it one position).

First, we count the closed walks of length $4$ that contain $v$ exactly in one position. Call the set of closed walks of length $4$ with $v$ at $i_{th}$ position as $C_4(v,i)$. This gives the total number of closed walks of length $4$ with the appearance of $v$ exactly once as $\sum_{i=1}^{4} |C_4(v,i)|$. Any closed walk in $C_4(v,1)$ is of the form $(v,a,b,c,v)$, where $a,b,c$ are nodes in $V$. Number of these walks is $\sum_{b \neq v} A^2(v,b)^2$. Clearly $(c,v,a,b,c)$ represents any closed walk in $C_4(v,2)$. Note that for a fixed $a,b$ and $c$, $(c,v,a,b,c)$ is one position clockwise rotation of closed walk $(v,a,b,c,v)$. This implies that corresponding to every closed walk in $C_4(v,1)$ there is a closed walk in $C_4(v,2)$ and vice versa. Hence the number of walks in $C_4(v,2)$ is also $\sum_{b \neq v} A^2(v,b)^2$. Similarly, we get that the number of walks in $C_4(v,3)$ and $C_4(v,4)$ is also the same.

So, we have

$$\sum_{i=1}^{4} |C_4(v,i)| = 4 \sum_{u \neq v} A^2(u,v)^2$$

Second, we consider closed walks in which $v$ appears twice. Now its possible in two ways: 1) $v$ appears in $1^{st}$ and $3^{rd}$ position as $(v,a,v,c,v)$ or 2) $v$ takes $2^{nd}$ and $4^{th}$ position as $(a,v,c,v,a)$. Clearly, there are $2A^2(v,v)^2$ such walks.

This gives the total number of closed walks of length $4$ containing a node $v$ as

$$\mathcal{W}_4(v, G) = 2A^2(v, v)^2 + 4\sum_{u \neq v} A^2(u, v)^2$$

which is the same as required. $\square$

We incorporate the above formula in the following algorithm. For a given node $v$,

$$score_G(v) = 4 \sum_{u \in V, u \neq v} d(u, v)^2 + 2d(v)^2$$

then $\sum_{u \neq v} d(u, v)^2$ can be computed by taking the characteristic vector $\chi_v$ of $N(v)$ (a bit vector of length $n$ where the $\chi_v[i] = 1 \leftrightarrow (v, v_i) \in E$). Then for each node $u \in V \setminus \{v\}$, we go through each neighbor $x$ of $u$ in its adjacency list and check if $\chi_v[x] = 1$ to increment $d(v, u)$.

It takes $O(m)$ time to compute the $score_G(x)$ for a node $x$, to get the node with maximum score it takes $O(nm)$ time. Note that after removing $k$ nodes (for constant $k$) the graph still has $O(m)$ edges. Now we give an efficient approximation to $score_G(x)$ that not only can be computed in linear time but also can be updated after removing a node $y$ in time proportional to $d(x)$.

We have

$$\left( \frac{\sum\limits_{u \neq v} d(u, v)}{n} \right)^2 \leq \left( \frac{\sum\limits_{u \neq v} d(u, v)^2}{n} \right) \leq \left( \frac{(\sum\limits_{u \neq v} d(u, v))^2}{n} \right). \tag{4.1}$$

The first inequality is the Cauchy-Schwarz inequality, [c.f [72]], while the second follows from the fact that $d(u, v)$ is non-negative for each $u, v$.

In view of the above inequality, we approximate the $score_G(v)$ by $score'_G(v)$ given as

$$score'_G(v) = 2d_G^2(v) + 4 \left( \sum_{u \neq v} d_G(u, v) \right)^2. \tag{4.2}$$

Our motivation to use $score'_G(x)$ is that not only it is easy to compute but also after a node is deleted it is easy to update the scores of all vertices in the remaining subgraph.

## 4.2 Algorithm

In this section, we give algorithm to select nodes for immunization based on $\mathcal{W}_4(v, G)$. First, we show procedure to find $score'_G(v)$ for graph $G$ in Algorithm COMPUTE-SCORE($G$) , then in Algorithm UPDATE-SCORE($G$, $v_i$), we give algorithm to update the scores of nodes when a node $v_i$ is deleted/immunized from the graph. Finally, we discuss the greedy approach to approximate that which nodes should be immunized in order to immunize the graph in Algorithm GREEDY-3 ($G$,$k$). We also discuss the time and space complexity of our solution in this section.

**Algorithm 2** : COMPUTE-SCORE($G$)

```
 1: $deg \leftarrow$ ZEROS$(n)$                              ▷ Initialize the degree array to $n$ zeros
 2: $codegSum \leftarrow$ ZEROS$(n)$                 ▷ Initialize the codegree sum array to $n$ zeros
 3: $score'_G \leftarrow$ ZEROS$(n)$                                    ▷ Initialize all scores to zeros
 4: for each node $v_i$ do
 5:     for each neighbor $v_j$ of $v_i$ do
 6:         $deg_G[v_i] \leftarrow deg_G[v_i] + 1$
 7: for each node $v_i$ do
 8:     for each neighbor $v_j$ of $v_i$ do
 9:         $codegSum[v_j] \leftarrow codegSum[v_j] + deg_G[v_i] - 1$
10: for each node $v_i$ do
11:     $score'_G[v_i] \leftarrow 2 * deg_G[v_i]^2 + 4 * codegSum[v_i]^2$
```

**Lemma 5.** *Runtime of Algorithm COMPUTE-SCORE(G) is $O(m)$.*

*Proof.* It is clear that line 6 of Algorithm COMPUTE-SCORE($G$) takes $O(1)$ time and it is executed $O(m)$ times, where $m = |E|$. Since loop at line 5 is iterated over all neighbors of a fixed node, $v_i$. Hence for $v_i$, line 6 is executed $d_G(v_i)$ times. Thus for all $v_i \in G$, line 6 runs for $\sum_{v_i \in V(G)} d_G(v_i) = 2m$ [73]. Same is true for line 9.

Line 11 has constant time computation while it is computed for every node, thus it takes $O(n)$ time. So total time taken to compute score of every node is $O(m)$. □

For a given node $v$ of $G$, we update the score of vertices after removing $v$ in the following way;

**Lemma 6.** *Algorithm UPDATE-SCORE(G,$v_i$) takes $O(m)$ time to update score with respect to parameter $v_i$.*

**Algorithm 3** : UPDATE-SCORE($G$,$v_i$)

```
1: for each neighbor $v_j$ of $v_i$ do
2:     $deg_G[v_j] \leftarrow deg_G[v_j] - 1$
3:     $codegSum[v_j] \leftarrow codegSum[v_j] - (deg_G[v_i] - 1)$
4:     for each neighbor $v_k$ of $v_j$ do
5:         $codegSum[v_k] \leftarrow codegSum[v_k] - 1$
6: $deg_G[v_i] \leftarrow 0$
7: $codegSum[v_i] \leftarrow 0$
8: $score'_G[v_i] \leftarrow 0$
9: for each neighbor $v_j of v_i$ do
10:     $score'_G[v_j] \leftarrow 2 * deg_G[v_j]^2 + 4 * codegSum[v_j]^2$
11:     for each neighbor $v_k$ of $v_j$ do
12:         $score'_G[v_k] \leftarrow 2 * deg_G[v_k]^2 + 4 * codegSum[v_k]^2$
```

*Proof.* In algorithm, line 2, 3 and 8 takes constant time steps and both are computed $d_G(v_i)$ times. However line 5 and 12 are computed $\sum_{v_j \in N_G(v_i)} d_G(v_j)$ times which is upper bounded by $m$. Hence for a fixed node $v_i$ runtime of algorithm is $O(m)$ □

We here give the proof of correctness of the Algorithms COMPUTE-SCORE($G$) and UPDATE-SCORE($G$,$v_i$).

**Lemma 7.** *1. For each node $v \in V$, Algorithm COMPUTE-SCORE($G$) computes the $score'_G(v)$ as defined in (4.2).*

*2. For all vertices $u, v \in V$ Algorithm UPDATE-SCORE(G,$v_i$) computes the $score'_{G-u}(v)$ as defined in (4.2).*

*Proof.* 1. It is clear that the Algorithm COMPUTE-SCORE($G$) computes the first term correctly, as each neighbor $v_i$ of $v$ contributes 1 to the degree of $v$. To see why the second term is computed correctly, consider the following fact:

$$\text{For } v \in V, \sum_{u \neq v} d_G(u, v) = \sum_{w \in N_G(v)} d_G(w) - 1$$

The left hand side is counting all occurrences of all vertices $w$ such that $w$ is a common neighbor of $u$ and some node $v$. Essentially counting all paths of length 2 from $u$ to $v$ where

$w$ is the center node.

We count these structures by counting the number of times each center node thus appear. Since $v$ is fixed, the number of times a node $w$ appears in such a structure is exactly the number of neighbors of $w$, i.e. $d_G(w)$ times. Now since $v$ is fixed and it is also a neighbor of $w$, we subtract one from it. Hence the expression on the right hand side follows.

2. To see this, consider a node $v$ which is neighbor of $u$. We note that first contribution of $u$ in score of $v$ is in the first term i.e. degree of $v$, so we decrease the degree of $v$ by one. Clearly from the figure below, $u$ adds $d_G(u) - 1$ in codegree sum of $v$, which is the second term of the $score'_G(v)$, given as $\sum_{u \neq v} d_G(u, v)$.

   Now consider the vertices $v_i$, which are neighbors of neighbors of $u$. Score of these vertices is also effected by removing $u$ since $u$ contribute one to codegree sum of $v_i$.

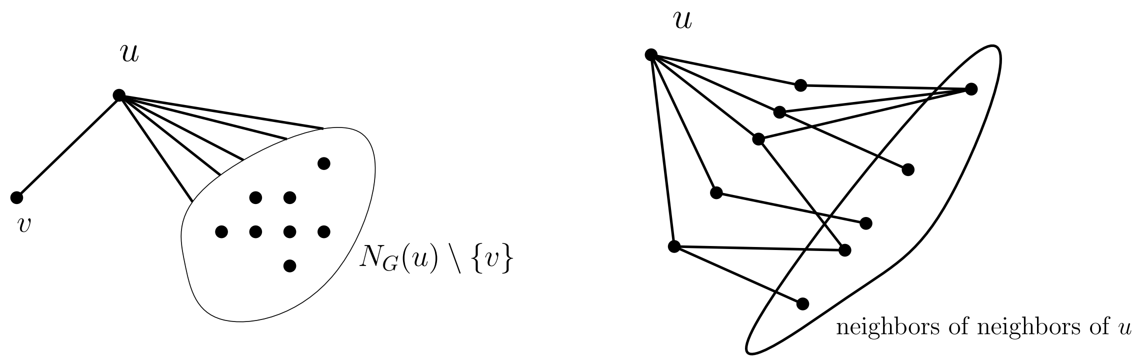


Figure 4.1: (Left) Node $u$ is a neighbor of node $v$ and the neighbors of $u$ except $v$ are shown. (Right) For a node $u$, its neighbors of neighbors are shown.

□

Here is the algorithm to select $k$ vertices in the given graph $G$ such that approximated eigendrop is maximized after deleting those vertices.

**Algorithm 4** : GREEDY-3($G$,$k$)

```
1: $S \leftarrow \emptyset$
2: COMPUTE-SCORE($G$)
3: while $|S| < k$ do
4:     $v \leftarrow \arg\max_{u \in V \setminus S} score'_G[u]$
5:     $S \leftarrow S \cup \{v\}$
6:     UPDATE-SCORE($G$,$v$)
7: return $S$
```

**Theorem 8.** *The computational complexity of the Algorithm GREEDY-3(G,k) is $O(n + km + k \log n)$, while the space complexity is $O(m + n + k)$.*

*Proof.* By Lemma 5 line 2 takes $O(m)$ time. The scores for all nodes can be stored in a MAX-HEAP, which can be built in $O(n)$ time while each UPDATE-KEY (updation of score of a node in MAX-HEAP) and EXTRACT-MAX (retrieval of maximum score entry from MAX-HEAP) takes $O(\log n)$ time [74]. We extract max from the the heap $k$ times, hence its total runtime is $O(\log n)$. As argued by Lemma 6, time consumed in each call to UPDATE-SCORE takes $O(m)$ time, we get that total time taken by the algorithm is $O(n + km + k \log n)$.

For the space complexity, in addition to storing the graph that takes $O(n + m)$ space, we need $O(n)$ space to store the three additional arrays. □

## 4.3 Experimental Evaluation

In this section, we present the experimental evaluation of our immunization solution. We evaluate the goodness of our algorithm on real world graphs to quantify the scalability and effectiveness on large graphs. Throughout this paper, we have used $SIR$ $(susceptible - infected - recovered)$ model for virus propagation in graphs. For benchmark comparison, we also implemented MAX-DEGREE and UPDATED-MAX-DEGREE approaches in which, MAX-DEGREE picks the top $k$ maximum degree vertices for immunization while UPDATED-MAX-DEGREE selects the node with the maximum degree and deletes that node and repeats $k$ times. We use the NETSHIELD implementation which is available online[1]. We implemented our proposed algorithm in Matlab and our implementation along with source code and documentation is available online [2].

| Network | Number of Nodes | Number of Edges |
|---|---|---|
| Karate | 34 | 78 |
| Oregon | 10,670 | 22,002 |
| AA | 418,236 | 2,753,798 |

Table 4.1: Summary of Datasets

The data sets used in experimentation are described in Table 4.1. All the real graphs used for experimentation obey power law distribution of degrees. The first data set is of a local karate club and is named as Karate graph[3]. Nodes of the graph represent members of the club and an edge between nodes show that corresponding members are friends with each other. The graph is undirected and unweighted and consists of $34$ nodes and $78$ edges.

The second data set is from Oregon AS (Autonomous System)[4] router graphs, which are AS level connectivity networks inferred from Oregon route views. There are a number of Oregon AS graphs available and each node represents a router and an edge between two routers represents a direct peering relationship between two routers. We have selected one dataset from Oregon router graphs having $10,670$ nodes and $22,002$ edges. The graph is undirected and unweighted.

The third data set (AA) is from DBLP[5] dataset. In graph a node represents an author and presence of an edge between two nodes shows that two authors have a co-authorship. In DBLP there is total node count of 418,236 and the number of edges among nodes is 2,753,798. We extracted smaller graphs by selecting co-authorship graph of only one journal (e.g Displays, International Journal of Computational Intelligence and Applications, International Journal of Internet and Enterprise Management, etc.). We ran our experiments on 20 different smaller co-authorship networks based on co-authorship graphs of 20 different journals. For the smaller sub graphs that we have extracted from DBLP dataset, node count goes up to few thousands and edge count goes up to

[1]https: //www.dropbox.com/s/aaq5ly4mcxhijmg/Netshieldplus.tar.
[2]`https://www.dropbox.com/sh/n7hwjc4imh62pe6/AADCyHG7uMGX6o9xtr1pdH6Qa?dl=0`
[3]http://konect.uni-koblenz.de/networks/ucidata-zachary
[4]http://snap.stanford.edu/data/oregon1.html
[5]http://dblp.uni-trier.de/xml/

few ten thousands. Co-authorship graph of ActaInf contains 1,791 nodes and 1,659 edges, graph of AI Communication journal has node count of 1,203 and edge count of 2,204 , sub graph of Asia-Pacific Journal of Operational Research (APJOR) has 1,132 nodes and 1,145 edges, Computer in Industry journal contains 2,844 nodes and 4,466 edges among nodes, journal of IEEE Wireless has total of 7,882 authors and 16,557 co-authorship links among authors. Detail of sub graphs of DBLP data set is also given in Table 4.2. These subgraphs are also undirected and unweighted.

| **Network** | **Number of Nodes** | **Number of Edges** |
|---|---|---|
| ActaInf | 1,791 | 1,659 |
| AI Communication | 1,203 | 2,204 |
| APJOR | 1,132 | 1,145 |
| Computer In Industry | 2,844 | 4,466 |
| Computing And Informatics | 1,598 | 2,324 |
| Ecological Informatics | 1,990 | 4,913 |
| IEEE Wireless | 7,882 | 16,577 |
| IJCIA | 848 | 975 |
| IJIEM | 373 | 357 |

Table 4.2: Summary of DBLP subgraphs

We did extensive experimentation with varying count of nodes to be immunized. In the results shown, x-axis shows the value of $k$ which is count of immunized nodes and y-axis shows the percentage of eigendrop which is the achieved benefit after deleting $k$ nodes from the graph. Results are evaluated on the basis of percentage of eigendrop. Eigendrop is difference of largest eigenvalues of original graph and perturbed version of graph after immunization of $k$ nodes.

$$\Delta\lambda = \lambda_{max}(A) - \lambda_{max}(A|_{-S}) \tag{4.3}$$

where $S$ is the set containing nodes to be immunized having cardinality $k$. It is clear from the results that our greedy algorithm outperforms NETSHIELD approach and other approaches like top $k$ degree and updated maximum degree approach. Our greedy algorithm is scalable for larger values of $k$ as well as for larger graphs as our algorithm has less running time complexity than NETSHIELD, top $k$ degree and updated maximum degree.

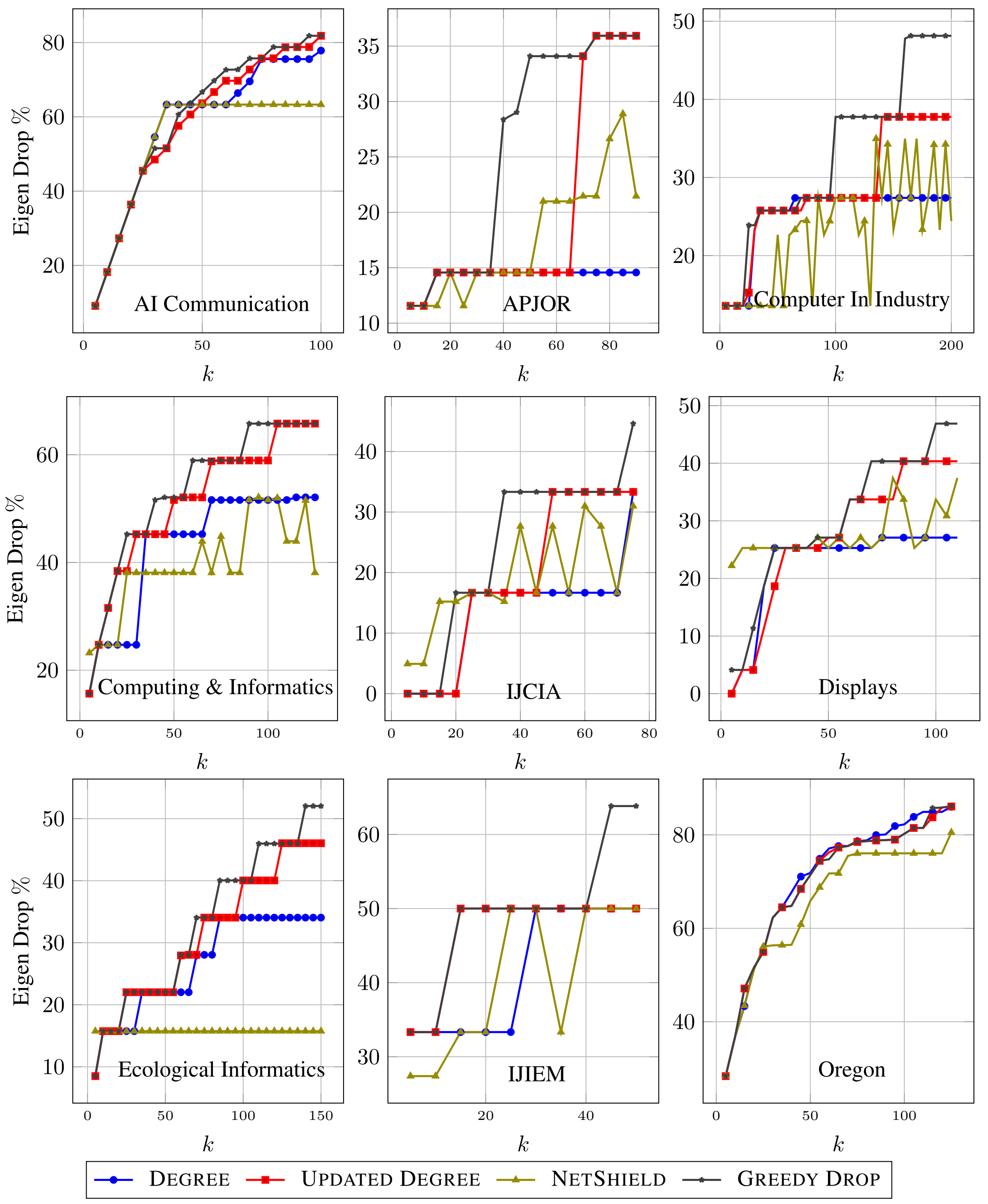


Figure 4.2: Comparison of eigen drop $\%$ (shown on $y$-axis, higher is better), calculated as $\frac{\Delta\lambda \times 100}{\lambda_{max}(A)}$, achieved by immunizing $k$ nodes ($x$-axis) in graphs using various techniques is shown. The comparison of our approach, GREEDY DROP, with NETSHIELD shows that our approach achieves higher eigen drop $\%$, resultantly, more reduction in the vulnerability of the graphs. Eigen drop achieved by immunizing nodes based on the maximum degree and the updated maximum degree is also shown.

## 4.4 Summary

In this work, we explored some links between established graph vulnerability measure and other spectral properties of even powers of adjacency matrix of the graph. We define shield value in terms of trace of the adjacency matrix of the graph. Based on these insights we present a greedy algorithm that iteratively selects $k$ nodes such that the impact of each node is maximum in the graph, in the respective iteration, and thus we maximally reduce the spread of a potential infection in the graph by removing those vertices. Our algorithm is scalable to large graphs since it has linear running time in the size of the graph.

We have conducted experiments on different real world communication graphs to confirm the accuracy and efficiency of our algorithm. Our algorithm outperforms the state-of-the-art algorithm in performance as well as in quality.

In the next chapter, we give method to compute closed walks of length $6$ and evaluate its performance.

# Chapter 5

# Scalable Approximation Algorithm for Network Immunization

In this chapter, we give solution to network immunization problem based on the number of closed walks of length 6, $\mathcal{W}_6(v, G)$ passing through node $v$ in graph $G$. First, in Section 5.1, we give a closed form expression to compute $\mathcal{W}_6(v, G)$. We note that computing $\mathcal{W}_6(v, G)$ for large graph is a computationally expensive task. To scale our solution to large graphs, we also give a method to approximate $\mathcal{W}_6(v, G)$. Then, we devise an algorithm to select nodes for immunization in Section 5.2. Experimental evaluation of the goodness of our solution is presented in Section 5.3.

## 5.1 Computation and Approximation of $\mathcal{W}_6(v, G)$

In this section, first we give a closed form expression to compute $\mathcal{W}_6(v, G)$ and then we present an approximate method to efficiently compute $\mathcal{W}_6(v, G)$. Computing $\mathcal{W}_p(S, G)$ is expensive for large value of $p$, but in practice we observe that $p = 6$ is sufficiently large.

**Theorem 9.** *Given a graph $G$ with adjacency matrix $A$*

$$\begin{aligned}\mathcal{W}_6(v,G) = \; & 6\sum_{i=1}^{n}\sum_{j=1}^{n} A^2(v,v_i)A^2(v,v_j)A^2(v_i,v_j) - 3\sum_{i=1}^{n}\sum_{j=1}^{n} A^2(v,v_i)A^2(v,v_j)A(v,v_i)A(v,v_j) \\ & - 6\sum_{i=1}^{n} A^2(v,v_i)^2 A^2(v,v) + 2A^2(v,v)^3\end{aligned}$$

*Proof.* A typical closed walk $W$ of length 6 can be represented as $(a,b,c,d,e,f,a)$. Note that $v$ can appear in a closed walk of length 6 at most thrice.

First we count the walks that contain $v$ exactly once. Lets assume that $v$ appears at the first position, i.e. $W = (v,a,b,c,d,e,v)$. Now since $v$ can not appear at any other position, we get that $b,c,d \neq v$. Also $(v,a,b)$, $(b,c,d)$ and $(d,e,v)$ are paths of length 2 and number of such walks can be $d(v,b)$, $d(b,d)$ and $d(d,v)$ respectively. But $d(a,d)$ may include $c = v$ case. So to exclude this we subtract $A(b,v)A(d,v)$ from $d(b,d)$ (this will be 1 only if $v$ is neighbor of both $b$ and $d$). We get that number of closed walks of length 6 containing $v$ only at first position is $\sum_{b\neq v}\sum_{d\neq v} d(v,b)d(v,d)[d(b,d) - A(v,b)A(v,d)]$. Each one position rotation of this walk results in distinct walk of the kind, so we get that walks containing $v$ exactly once are $6\sum_{b\neq v}\sum_{d\neq v} d(v,b)d(v,d)[d(b,d) - A(v,b)A(v,d)]$.

Now we count the walks containing $v$ twice. One way to get such walk is $v$ is in first and third positions i.e. $W = (v,a,v,b,c,d,v)$. Number of such walks is $\sum_{c\neq v} d(v,c)^2 d(v)$. Again each rotation gives unique walk, so we have $6\sum_{c\neq v} d(v,c)^2 d(v)$ such walks. Another way to have a walk with $v$ appearing twice is $W = (v,a,b,v,c,d,v)$. There are $\sum_{b\in N(v)}\sum_{d\in N(v)} d(v,b)d(v,d)$ walks with $v$ at first and fourth position which is same as
$\sum_{b\in V}\sum_{d\in V} d(v,b)A(v,b)d(v,d)A(v,d)$. Note that only two clockwise rotations result in new walks. This gives that total number of closed walks of length 6 containing $v$ twice is

$$6\sum_{c\neq v} d(v,c)^2 d(v) + 3\sum_{b\in N(v)}\sum_{d\in N(v)} d(v,b)d(v,d)$$

If we consider walks which contain $v$ thrice, then there are two possibilities for such walks, one which start at $v$ *i)* $(v,a,v,b,v,c,v)$ and *ii)* $(a,v,b,v,c,v,a)$. Count for either of them is $d(v)^3$.

This gives the total count of these walks as $2d(v)^3$.

So number of closed walks of length 6 containing a vertex $v$ in graph $G$ is

$$\mathcal{W}_6(v,G) = 6\sum_{b\neq v}\sum_{d\neq v} d(v,b)d(v,d)[d(b,d) - A(v,b)A(v,d)] + 6\sum_{c\neq v} d(v,c)^2 d(v) + 3\sum_{b\in N(v)}\sum_{d\in N(v)} d(v,b)d(v,d) + 2d(v)^3$$

$\square$

Clearly computing this number for any vertex $v$ takes $O(n^2 + c(n))$ time where $c(n)$ is the time taken for computing $A^2$. So instead we approximate the number of closed walks of length 6 containing $v$.

## 5.1.1 Approximating number of walks

An equivalent expression for $\mathcal{W}_6(v,G)$ is

$$6A^6(v,v) - 6A^4(v,v)A^2(v,v) - 3(A^3(v,v))^2 + 2(A^2(v,v))^3.$$

The formula for $\mathcal{W}_6(v,G)$ suggests that we need to approximate the powers of adjacency matrix $A$ of $G$. For the purpose, we consider a summary graph $H$ of $G$ which is weighted undirected graph with adjacency matrix $A(H) = C$.

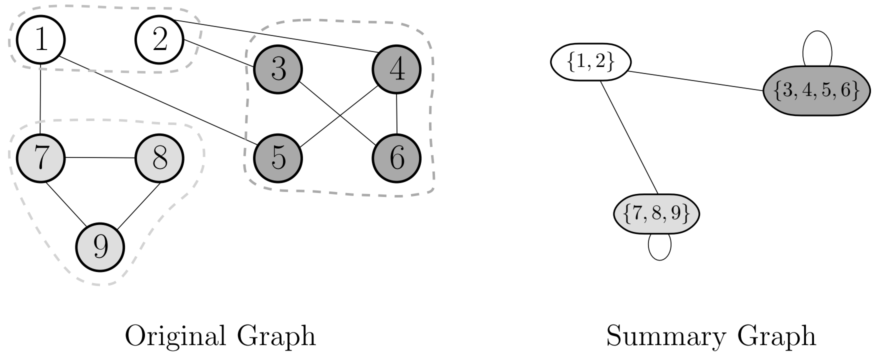


Figure 5.1: (Left) A graph and (right) its random summary are shown.

We generate matrix $C$ (graph $H$) in the following way. First, we partition the node set $V(G)$ into random subsets using a random *hash function* $h$. Let the partition of nodes under the hash function $h$ be $\mathcal{P}(h)$ with $|\mathcal{P}(h)| = \alpha$. Second we construct matrix $C$ as

**Algorithm 5** : SummaryGraph($A(G)$,$\alpha$,$h$)

```
$C \leftarrow$ ZEROS$(\alpha \times \alpha)$
for $i = 1$ to $n$ do
    for $j = i$ to $n$ do
        if $A(i,j) = 1$ then
            $C[h(i)][h(j)] \leftarrow C[h(i)][h(j)] + 1$
            $C[h(j)][h(i)] \leftarrow C[h(i)][h(j)]$
return $C$
```

This matrix $C$ corresponds to summary graph $H$ in which every node (super-node) represents
965 a set of nodes in $\mathcal{P}(h)$ and $C(i,j)$ entry denotes the number of edges from super-node $i$ to super-node $j$ (number of edges from nodes in super-node $i$ to nodes in super-node $j$). Lets denote $i$th super-node of $H$ by $\mathcal{X}_i$

In order to approximate $\mathcal{W}_6(v,G)$ of a node $v \in V(G)$, we use powers of matrix $C$ instead those of $A(G)$. We keep $C^2$ and $C^3$ matrices. For each $\mathcal{X}_i \in V(H)$, we compute terms $C^6(i,i)$
970 using formula $\sum_{j=1}^{\alpha}(C^3(i,j))^2$ and $C^4(i,i)$ by $\sum_{j=i}^{\alpha}(C^2(i,j))^2$.

Note that $C^p(i,j)$ represents the total number of walks of length $p$ from nodes in $\mathcal{X}_i$ to nodes in $\mathcal{X}_j$. So, we can find the number of closed walks of length $p$ containing a specific node $v \in V(G)$, by estimating the contribution of this node in the total number of walks in $\mathcal{X}_{h(v)}$. For this purpose, we define the contribution factor of $v$ with $h(v) = i$ in $C^p(i,i)$ as

$$\frac{d_{\mathcal{X}_i}(v)^p}{\sum_{u \in \mathcal{X}_i} d_{\mathcal{X}_i}(u)^p}$$

This can be seen clear if we expand the terms $C^p(i,i)$. For instance if we expand $C^3(i,i)$, we get one term as $(C(i,i))^3$ which is same as $\left(\sum_{v \in \mathcal{X}_i} d_{\mathcal{X}_i}(v)\right)^3$. We define $\sum_{u \in \mathcal{X}_i} d_{\mathcal{X}_i}(u)^p$ to be $D_p(i)$. So we get that the estimated value of $cw_6(v,G)$ is following when $h(v) = i$

$$\mathcal{W}'(v) = 6C^6(i,i)\frac{d_G(v)^6}{D_6(i)} - 6d_G(v)C^4(i,i)\frac{d_G(v)^4}{D_4(i)} - 3\left(C^3(i,i)\frac{d_G(v)^3}{D_3(i)}\right)^2 + 2\left(d_G(v)\right)^3$$

## 5.2 Algorithm

In this section, we give the approximate algorithm to compute $\mathcal{W}(v, G)$. Since we partitioned $V(G)$ using random hash functions, we use multiple hash functions to normalize the effect of randomness as follows.

**Algorithm 6** : EstimateWalks($A(G)$,$\alpha$,$\beta$)

```
for $i = 1$ to $\beta$ do
    $\mathcal{W}'_i \leftarrow \textsc{zeros}(n)$
    $h_i(i) = (a * i + b)\%\alpha$                    ▷ $a$ and $b$ are two random numbers
    $C_i \leftarrow \textsc{SummaryGraph}(A(G), \alpha, h_i)$
    for $j = 1$ to $n$ do
        Compute $\mathcal{W}'_i[v_j]$
$cwMin \leftarrow \textsc{zeros}(n)$
for $j = 1$ to $n$ do
    $cwMin[v] \leftarrow \min_i \mathcal{W}'_i[v_j]$
return $cwMin$
```

Once we have estimated the walks for each node $v$ of $V(G)$ using multiple hash functions, call it $W(v)(v)$, we can select set $S$ for immunization that contain nodes with most number of walks. But for efficient results we would prefer to choose $S$ that have nodes which are well spread apart and we do not want to select a lot of those nodes which are connected to each other. In order to deal with this, we define the score of each candidate subset $S$, on basis of which we select $S$ for immunization. For $v \in V(G)$, and $S \subset V(G)$,

$$score(S) = \gamma \sum_{v \in S} W(v)^2 - \sum_{u,v \in S} W(v)A(u,v)W(u) \tag{5.1}$$

where $\gamma$ is a positive integer. We want to find set $S$ such that

$$S = \arg\max score(S), |S| = k.$$

But this optimization problem is clearly computationally intractable since it requires computing score for each of $\binom{n}{k}$ sets. So we show that the function $score(S)$ is monotonically non-decreasing and sub-modular, allowing us to devise a greedy strategy to construct set $S$ with guaranteed good approximation of our results.

Proving that our optimization function is sub-modular, and we can use Theorem 1, which guarantees that the greedy strategy will be $(1-1/e)$-approximate algorithm. We give the following greedy algorithm to construct the required set $S$.

**Algorithm 7** : GreedyNodeImmunization($A(G)$,$k$,$\alpha$,$\beta$)

```
1: $S \leftarrow \emptyset$
2: $W_2, Score \leftarrow \textsc{zeros}(n)$
3: $W \leftarrow \textsc{EstimateWalks}(A(G), \alpha, \beta)$
4: $\gamma \leftarrow \max_i W[i]$
5: for $i = 1$ to $n$ do
6:     $W_2[i] \leftarrow \gamma W[i]^2$
7: for $i = 1$ to $k$ do
8:     $a_S \leftarrow A[:, S] * W[S]$
9:     for $j = 1$ to $n$ do
10:        if $j \notin S$ then
11:            $Score[j] \leftarrow W_2[j] - 2a_S[j]W[j]$
12:        else
13:            $Score[j] \leftarrow -1$
14:    $maxNode \leftarrow \arg\max_j Score[j]$
15:    $S \leftarrow S \cup \{maxNode\}$
16: return $S$
```

**Analysis of Algorithm**

Now we analyze our proposed algorithm and give its runtime complexity. First we discuss complexity of EstimateWalks function. This function needs to compute the following $\beta$ times (count of hash functions): $C$, $C^2, C^3, C^4, C^6$, $D_6(i)$ for all sets in partition formed by hash function, $\mathcal{W}'(v)$ for $n$ vertices.

Note that $C$ matrix for all hash functions can be computed with one scan of the whole graph, which takes $n^2$ time. Computing all the above, except $C$ takes at most $O(\alpha^3)$ time. For every hash function, it takes maximum $O(n + \alpha^3)$ time and finding min $\mathcal{W}'(i)$ for each vertex takes $\beta n$ time. This implies $EstimateWalks$ function takes $O(n^2 + \beta(n + \alpha^3))$ time.

Line $4$ and first for loop takes $O(n)$ steps. The $j$th iteration if loop in lines $9$ to $13$ takes $O(n + nj)$ and line $14$ is $O(n)$ work. This shows that second loop from line $7$ to $15$ takes $\sum_{j=1}^{k} O(n+nj)$

time which is $O(nk^2)$ in total.

So GreedyNodeImmunization($A(G)$, $k$, $\alpha$,$\beta$) algorithm takes total $O(n^2 + \beta(n + \alpha^3) + nk^2)$ time.

## 5.3 Experiments

We present results of our suggested algorithm in detail in this section. We have compared results of our algorithm with those of NETSHIELD[1], Brute Force Method and Walk $4$ [21] to evaluate the quality and efficiency. NETSHIELD selects the vertices based on the eigen vector corresponding to largest eigenvalue of graph, Brute Force algorithm picks vertices which have maximum number of closed walks of length six passing across them and Walk $4$ chooses nodes based on approximation of walks of length $4$ for immunization purpose. We have implemented the algorithm in Matlab and we have made our code available at the given link[2].

| **Network** | **Number of Nodes** | **Number of Edges** |
|---|---|---|
| Karate | 34 | 78 |
| Oregon | 10,670 | 22,002 |
| AA | 418,236 | 2,753,798 |

Table 5.1: Summary of Datasets

We have used real world graphs for experimentation and all our graphs are undirected and unweighted. The first data set called Karate graph[3] is a small graph of local karate club in which nodes represent members of the club and an edge between two nodes shows friendship among corresponding members. Karate graph consists of 34 nodes and 78 edges.

Second dataset is obtained from Oregon AS (Autonomous System)[4] router graphs. We have constructed a communication graph in which nodes are participating routers and an edge between two routers represents direct peering relationship among them. A number of Oregon graphs are

[1]https://www.dropbox.com/s/aaq5ly4mcxhijmg/Netshieldplus.tar

[2]`https://www.dropbox.com/sh/n7hwjc4imh62pe6/AADCyHG7uMGX6o9xtr1pdH6Qa?dl=0`

[3]http://konect.uni-koblenz.de/networks/ucidata-zachary

[4]http://snap.stanford.edu/data/oregon1.html

available and each graph is made from communication log of one week. We have selected a graph containing 10,670 nodes and 22,002 edges.

The third data set (AA) is from DBLP[5] dataset. In this graph a node represents an author and presence of an edge between two nodes shows that two authors have a co-authorship. In DBLP there is total node count of 418,236 and the number of edges among nodes is 2,753,798. We extracted smaller sub-graphs by selecting co-authorship graphs of individual journals (e.g Displays, International Journal of Computational Intelligence and Applications, International Journal of Internet and Enterprise Management, etc.). We ran our experiments on 20 different smaller co-authorship graphs of different journals. For the smaller sub graphs that we have extracted from DBLP dataset, node count goes up to few thousands and edge count goes up to few ten thousands. Details of sub graphs of DBLP data set is given in Table 6.3. These subgraphs are also undirected and unweighted.

| **Network** | **Number of Nodes** | **Number of Edges** |
|---|---|---|
| AI Communication | 1,203 | 2,204 |
| APJOR | 1,132 | 1,145 |
| Computer In Industry | 2,844 | 4,466 |
| Computing And Informatics (CAI) | 1,598 | 2,324 |
| Decision Support Systems (DSS) | 4,926 | 14,660 |
| Display | 1,374 | 3,204 |
| Ecological Informatics | 1,990 | 4,913 |
| Engineering Application of AI | 4,164 | 6,733 |
| IJCIA | 848 | 975 |

Table 5.2: Summary of DBLP subgraphs

We performed extensive experimentation with varying number of nodes to be immunized in graph. In the results shown, x-axis shows the count of nodes being immunized denoted by $k$ while y-axis shows the benefit achieved in terms of percentage of eigen drop after immunizing $k$ nodes in graph. It is clear from results that our algorithm beats other variants for immunizing the graph in terms of effectiveness. Our algorithm has less computational cost than its competitors and is scalable to larger values of $k$ and also for large size graphs. It is worth mentioning that our

[5]http://dblp.uni-trier.de/xml/

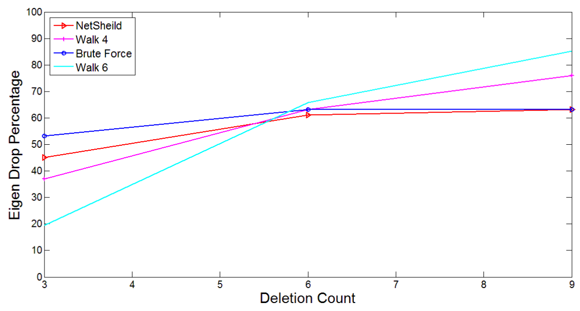


Figure 5.2: Eigendrop of Karate Graph

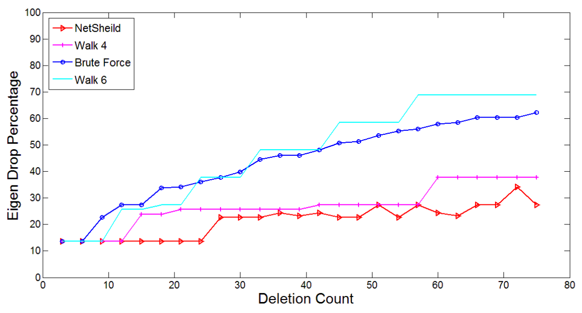


Figure 5.3: Eigendrop of Computer In Industry Graph

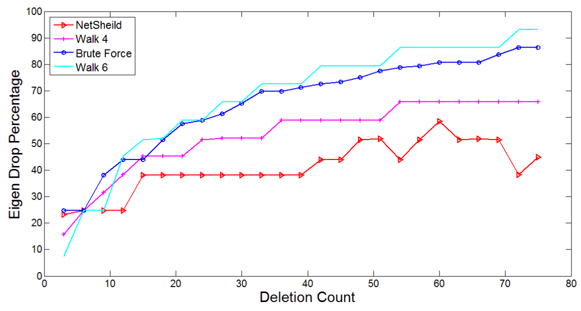


Figure 5.4: Eigendrop of CAI Graph

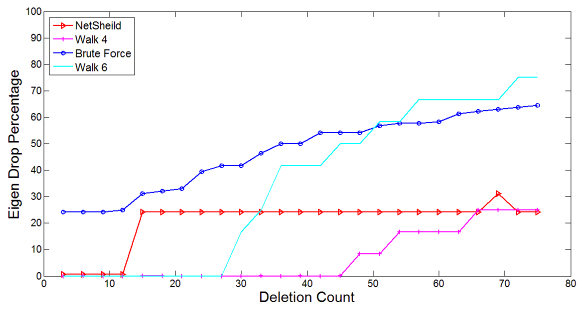


Figure 5.5: Eigendrop of DSS Graph

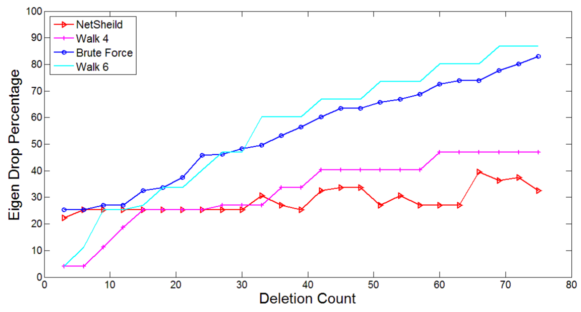


Figure 5.6: Eigendrop of Displays Graph

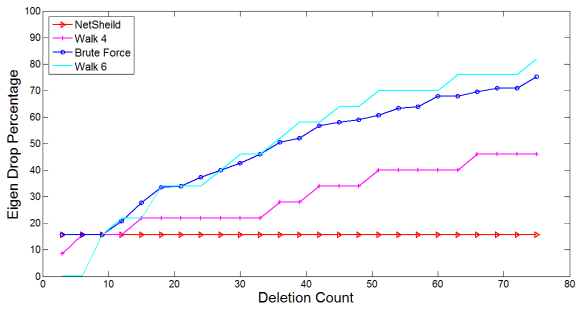


Figure 5.7: Eigendrop of Ecological Informatics Graph

algorithm achieves high accuracy in terms of approximation with very small $k$. Hence for very large graphs, our algorithm will achieve reasonable level of accuracy in very little time.

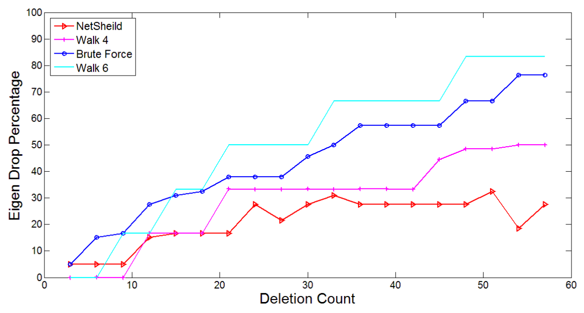


Figure 5.8: Eigendrop of IJCIA Graph

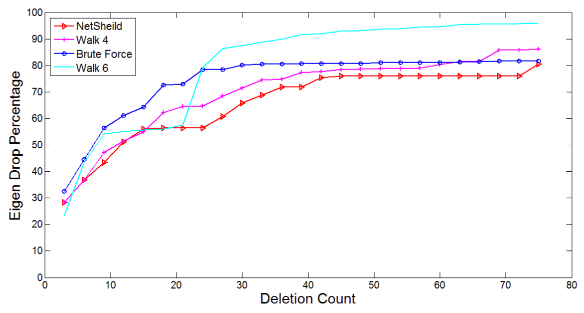


Figure 5.9: Eigendrop of Oregon Graph

## 5.4 Summary

In this work, we explored some links between established graph vulnerability measure and other spectral properties of even powers of adjacency matrix of the graph. We define a shield value in terms of the trace of the adjacency matrix of the graph. Based on these insights, we present a greedy algorithm that iteratively selects $k$ nodes such that the impact of each node is maximum in the graph, in the respective iteration. Thus, we maximally reduce the spread of a potential infection in the graph by removing those vertices. Our algorithm is scalable to large graphs and we have done experimentation on different real world communication graphs to prove the accuracy and efficiency of our algorithm. Our method beats the state-of-the-art algorithms in performance as well as in quality.

In the next chapter, we extend our solution to compute closed walks of length $8$ and evaluate its performance.

# Chapter 6

# Combinatorial trace method for network immunization

In this chapter, we give solution to network immunization problem based on the number of closed walks of length 8, $\mathcal{W}_8(v, G)$ passing through node $v$ in graph $G$. First, in Section 6.1, we give a closed form expression to compute $\mathcal{W}_8(v, G)$. We give our approximate algorithm in Section 6.2 to immunize nodes based on $\mathcal{W}_8(v, G)$. Experimental evaluation of the goodness of our solution is presented in Section 6.3.

## 6.1 Computing walks of length $8$

The proposed shield value, $score_p(S)$, quantifies the importance of set $S$ based on the number of $p$-length closed walks containing nodes from $S$. Building $S$ requires $\mathcal{W}_p(v, G)$ for all $v \in V$. A closed-form of $\mathcal{W}_p(v, G)$ depends on the actual value of $p$. In practice, the value of $p = 8$ produces the set $S$ with sufficient quality. We select nodes in a graph based on the number of closed walks of length $8$ (referred to as $8$-walks) for immunization purposes.

### 6.1.1 Justification for p=8

Recall that our aim is to find a set $S$ that minimizes $\lambda_{max}(A|_{-S})$. From (3.1), we get that for large $p$, $trace(A^p)$ approaches $\lambda_{max}(A)^p$. Hence, we find a set $S$ with minimum $trace(A|^p_{-S})$. We show
1070 that in practice $trace(A^8) = \sum_{i=1}^{n=|V|} \lambda_i(A^8)$ is sufficiently close to $\lambda_{max}(A^8)$ . Table 6.1 demonstrates that in real world graphs $\frac{\lambda_{max}(A^8)}{\sum_{i=1}^{n}\lambda_i(A^8)} = \frac{\lambda_{max}(A^8)}{trace(A^8)}$ is close to $1$ specially if there is significant *eigen-gap* $\big(\lambda_{max}(A) - \lambda_2(A)\big)$. In other words, $\lambda_{max}(A^8)$ is the most dominant term in $trace(A^8)$ and the combined effect of the other terms $\big(\lambda_2(A^8) + \cdots + \lambda_n(A^8)\big)$ diminishes.

Table 6.1: Ratio of $\lambda_{max}(A^8)$ to $\sum_{i=1}^{n}\lambda_i(A^8)$ is shown. Note that as relative eigen-gap increases, the ratio approaches to $1$. We show the ratio only for moderately large graphs because computing all $n$ eigen values for very large graphs takes very long time.

| **Network** | $\lvert V\rvert$ | $\lambda_{max}(A)$ | $\lambda_2(A)$ | $\frac{\lambda_{max}(A^8)}{\sum_{i=1}^{n}\lambda_i(A^8)}$ |
|---|---|---|---|---|
| EngineeringApplicationofAI | 4164 | 16 | 13.2 | 0.756 |
| Facebook | 4039 | 162.4 | 125.5 | 0.859 |
| Email | 1005 | 77.2 | 36.9 | 0.993 |
| AICommunication | 1203 | 33 | 12.1 | 0.999 |

### 6.1.2 Closed-Form expression for $\mathcal{W}_8(v, G)$

1075 We derive a closed-form expression for computing $\mathcal{W}_8(v, G)$. To the best of our knowledge, we are the first one to derive such expression.

**Theorem 10.**

$$\begin{aligned}\mathcal{W}_8(v, G) =& 8A^8(v,v) - 8A^2(v,v)A^6(v,v) - 8A^3(v,v)A^5(v,v) - 4(A^4(v,v))^2 \\ &+ 8A^2(v,v)(A^3(v,v))^2 + 8(A^2(v,v))^2A^4(v,v) - 2(A^2(v,v))^4\end{aligned}$$

*Proof.* An $8$-walk in $G$ is represented as $W = (a, b, c, d, e, f, g, h, a)$ and the goal is to compute the number of $8$-walks containing a node $v$. Node $v$ can occur at most four times in an $8$-walk and we consider each case of the number of occurrences of $v$ as follows.

Let $T_{\{l_1,\cdots,l_i\}}, 1 \leq i \leq 4$ be the collection of $8$-walks containing $v$ exactly $i$ times. For $W \in T_{\{l_1,\cdots,l_i\}}$, then $W$ starts and ends at $v$ and can be written as concatenation of walks of lengths $l_1, \cdots, l_i$, each starting and ending at $v$. We note that $2 \leq l_k \leq 8$, for $1 \leq k \leq 4$, and $\sum_{k=1}^{i} l_k = 8$. For example $T_{\{2,3,3\}}$ contains the walks of type $(v, a, v, b, c, v, d, e, v)$ i.e. it is sequence of $(v, a, v), (v, b, c, v)$ and $(v, d, e, v)$.

The rotations of nodes in a walk give different, and sometimes distinct, walks. Given a walk $(a, b, c, d, e, f, g, h, a)$, one vertex left rotation will produce another walk $(b, c, d, e, f, g, h, a, b)$. So recurrent, one vertex, rotations of walks in $T_{\{l_1,\cdots,l_i\}}$ can give up to $8|T_{\{l_1,\cdot,l_i\}}|$ different walks.

We count the walks of each type i.e. walks in $T_{\{2,2,2,2\}}, T_{\{2,2,4\}}, T_{\{2,3,3\}}, T_{\{2,6\}}, T_{\{3,5\}}, T_{\{4,4\}}, T_{\{8\}}$ and their distinct rotations. In counting there are cases when walks in $T_{\{2,3,3\}}$ are considered and these are different from walks in $T_{\{3,2,3\}}$, but $|T_{\{2,3,3\}}| = |T_{\{3,2,3\}}|$.

First, we count the number of walks containing $v$ exactly $4$ times. The walk $(v, a, v, b, v, c, v, d, v)$, where $\{a, b, c, d\} \in N(v)$, is represented as $T_{\{2,2,2,2\}}$ as concatenation of $4$ closed walks of length 2. The number of such walks is $(A^2(v, v))^4$. In this case, only one vertex rotation is possible which gives $(a, v, b, v, c, v, d, v, a)$ because a second rotation gives the same original walk. Hence, the number of walks containing $v$ exactly $4$ times is $2(A^2(v, v))^4$.

The walks having $v$ exactly 3 times are contained in $T_{\{2,3,3\}}$ and $T_{\{2,2,4\}}$. The number of walks in $T_{\{2,3,3\}}$ is $A^2(v, v)(A^3(v, v))^2$ and for each walk in this set, $8$ distinct walks are possible after rotations. The total number of walks containing $v$ 3 times is $8\big[A^2(v, v)(A^3(v, v))^2\big]$.

A walk in $T_{\{2,2,4\}}$ is concatenation of $(v, a, v), (v, b, v)$, $(v, c, d, e, v)$, where $d \neq v$. Number of all walks of form $(v, a, v, b, v, c, d, e, v)$ is at most $8(A^2(v, v))^2A^4(v, v)$ but this number includes walks with $d = v$ as well. To exclude those, we note that when $d = v$, walk is of type $T_{\{2,2,2,2\}}$ which we have already counted in first case. Subtracting the instance when $d = v$ in $(v, a, v, b, v, c, d, e, v)$, we get $|T_{\{2,2,4\}}| = (A^2(v, v))^2A^4(v, v) - (A^2(v, v))^4$. All 8 vertex rotations of walks in $T_{\{2,2,4\}}$ give distinct walks. The total number of 8-walks containing $v$ thrice is

$$\begin{aligned}
&=8|T_{\{2,2,4\}}| + 8|T_{\{2,3,3\}}| \\
&=8\left[(A^2(v, v))^2A^4(v, v) - (A^2(v, v))^4\right] + 8\left[A^2(v, v)(A^3(v, v))^2\right]
\end{aligned}$$

Walks containing $v$ exactly twice are represented as $T_{\{3,5\}}$, $T_{\{2,6\}}$ and $T_{\{4,4\}}$. A walk in $T_{\{3,5\}}$ is of the form $(v, a, b, v, c, d, e, f, v)$ where $d, e \neq v$. The number of walks with $d = v$ and $e = v$ is $|T_{\{3,2,3\}}|$ and $|T_{\{3,3,2\}}|$. So $|T_{\{3,5\}}| = A^3(v, v)A^5(v, v) - 2A^2(v, v)(A^3(v, v))^2$. In this case, vertex rotations give $8$ distinct walks.

Walks in $T_{\{2,6\}}$ are of the form $(v, a, v, b, c, d, e, f, v)$ where $c, d, e \neq v$. There are maximum $A^2(v, v)A^6(v, v)$ walks of type $T_{\{2,6\}}$ but these include walks with $c = v$, $d = v$, $e = v$ and $c, e = v$. For $c = v$ and $e = v$, we get walks of types $T_{\{2,2,4\}}$ and $T_{\{2,4,2\}}$ respectively while if $d = v$ then it is a walk of type $T_{\{2,2,2,2\}}$. For $d = v$, we get walk of type $T_{\{2,3,3\}}$.

$$\begin{aligned}|T_{\{2,6\}}| =&A^2(v, v)A^6(v, v) - 2|T_{\{2,2,4\}}| - |T_{\{2,3,3\}}| - |T_{\{2,2,2,2\}}|\\ =&A^2(v, v)A^6(v, v) - 2(A^2(v, v))^2A^4(v, v) - A^2(v, v)(A^3(v, v))^2 + (A^2(v, v))^4\end{aligned}$$

In the case of $T_{\{2,6\}}$, rotations of vertices give $8$ different walks.

The number of walks of type $T_{\{4,4\}}$ in $(A^4(v, v))^2$ but it also includes $|T_{\{2,4,4\}}|$ and $|T_{\{2,2,2,2\}}|$. Therefore, we get

$$\begin{aligned}|T_{\{4,4\}}| &= (A^4(v, v))^2 - 2|T_{\{2,2,4\}}| - |T_{\{2,2,2,2\}}|\\ &= (A^4(v, v))^2 - 2(A^2(v, v))^2A^4(v, v) + (A^2(v, v))^4\end{aligned}$$

In this case, only the first $4$ vertex rotations give different walks and $5^{th}$ rotation gives the original walk. The total number of walks containing $v$ exactly twice is

$$\begin{aligned}=&8|T_{\{3,5\}}| + 8|T_{\{2,6\}}| + 4|T_{\{4,4\}}|\\ =&8\left[A^3(v, v)A^5(v, v) - 2A^2(v, v)(A^3(v, v))^2\right] + 8[A^2(v, v)A^6(v, v)\\ &- 2(A^2(v, v))^2A^4(v, v) - A^2(v, v)(A^3(v, v))^2 + (A^2(v, v))^4] + 4\big[(A^4(v, v))^2\\ &- 2(A^2(v, v))^2A^4(v, v) + (A^2(v, v))^4\big]\\ =&8A^3(v, v)A^5(v, v) + 8A^2(v, v)A^6(v, v) + 4(A^4(v, v))^2 - 24A^2(v, v)(A^3(v, v))^2\\ &- 24(A^2(v, v))^2A^4(v, v) + 12(A^2(v, v))^4\end{aligned}$$

$T_{\{8\}}$ consists of walks containing $v$ only once and are of the form $(v, a, b, c, d, e, f, g, v)$. The number of such walks is $A^8(v,v)$. But this includes walks with some combinations of $b, c, d, e, f$ equal to $v$ as well. Subtracting already counted walks from $T_{\{8\}}$ gives

$$\begin{aligned}|T_{\{8\}}| =& A^8(v,v) - 2A^2(v,v)A^6(v,v) - 2A^3(v,v)A^5(v,v) - (A^4(v,v))^2 \\ &+ 3A^2(v,v))^2A^4(v,v) + 3A^2(v,v)(A^3(v,v))^2 - (A^2(v,v))^4\end{aligned}$$

In $T_{\{8\}}$, vertex rotations give $8$ distinct walks so the number of walks containing $v$ once is

$$\begin{aligned}8|T_{\{8\}}| =& 8A^8(v,v) - 16A^2(v,v)A^6(v,v) - 16A^3(v,v)A^5(v,v) - 8(A^4(v,v))^2 \\ &+ 24A^2(v,v))^2A^4(v,v) + 24A^2(v,v)(A^3(v,v))^2 - 8(A^2(v,v))^4\end{aligned}$$

Combining all the four cases of occurrence of $v$ in $8$-walk gives

$$\begin{aligned}\mathcal{W}_8(v,G) =& 2|T_{\{2,2,2,2\}}| + 8|T_{\{2,2,4\}}| + 8|T_{\{2,3,3\}}| + 8|T_{\{3,5\}}| + 8|T_{\{2,6\}}| \\ &+ 4|T_{\{4,4\}}| + 8|T_{\{8\}}| \\ =& 8A^8(v,v) - 4(A^4(v,v))^2 - 8A^2(v,v)A^6(v,v) - 8A^3(v,v)A^5(v,v) \\ &+ 8A^2(v,v)(A^3(v,v))^2 + 8(A^2(v,v))^2A^4(v,v) - 2(A^2(v,v))^4\end{aligned}$$

□

## 6.2 Proposed algorithm

In this section, we give our algorithm to compute the number of $8$-walks passing through each vertex and select nodes for immunization. Recall from Theorem 10 that computing number of $8$-walks requires $8^{th}$ power of the adjacency matrix $A$. Let $f(n)$ be the running time for taking $8^{th}$ power of $A$. Computing $\mathcal{W}_8(v,G)$ for all $v \in V$ using Theorem 10 takes $O(n + f(n))$ time. Note that while for many real-world graphs $A$ is sparse; this does not necessarily hold for $A^2$ and higher powers of $A$. The above runtime, therefore is prohibitive for real-world graphs, since best-known bounds on $f(n)$ are super-quadratic.

We propose to approximately compute $\mathcal{W}_8(v,G)$ from a summary of $G$ [7, 25, 26]. Given a graph $G = (V(G), E(G))$ on $n$ nodes, a summary $H$ of $G$, $H = (V(H), E(H))$ is a graph on $t$ nodes with weights on both its nodes and edges. $V(H) = \{V_1, \ldots, V_t\}$ is a partition of $V(G)$, i.e. $V_i \subset V(G)$ for $1 \leq i \leq t$, $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{t} V_i = V(G)$. Each $V_i$ (called supernode) is associated with two integers $n_i = |V_i|$ and $e_i = |\{(u,v)|u,v \in V_i, (u,v) \in E(G)\}|$. Weight of an edge $(V_i, V_j) \in E(H)$ (called superedge), is $e_{ij}$ : the number of edges in the bipartite subgraph induced between $V_i$ and $V_j$ i.e. $e_{ij} = |\{(u,v)|u \in V_i, v \in V_j, (u,v) \in E(G)\}|$. The original graph $G$ is approximately reconstructed from $H$ as the expected adjacency matrix, $A'_{n\times n}$ with a row and column corresponding to each $u \in V(G)$ given as:

$$A'(u,v) = \begin{cases} 0 & \text{if } u = v \\ \frac{e_i}{\binom{n_i}{2}} & \text{if } u, v \in V_i \\ \frac{e_{ij}}{n_i n_j} & \text{if } u \in V_i, v \in V_j \end{cases}$$

Let $H$ be a summary graph of $G$ on $t$ supernodes and let $C$ be its adjacency matrix. Clearly, $C^p(i,j)$ is the number of walks of length $p$ from nodes in $V_i$ to nodes in $V_j$. We estimate the contributions of $v \in V_i$ to $C^p(i,i)$ by $\alpha_p(v).C^p(i,i)$, where $\alpha_p(v) = \dfrac{d_G(v)^p}{\sum_{u\in V_i} d_G(u)^p}$. Our estimate for $\mathcal{W}_8(v,G)$ is

$$\begin{aligned}\mathcal{W}'_8(v,G) =& 8C^8(i,i)\alpha_8(v) - 8d_G(v)6C^6(i,i)\alpha_6(v) - 8C^5(i,i)\alpha_5(v)C^3(i,i)\alpha_3(v) - \\ & 4\left(C^4(i,i)\alpha_4(v)\right)^2 + 8d_G(v)\left(C^3(i,i)\alpha_3(v)\right)^2 + 8d_G(v)^2C^4(i,i)\alpha_4(v) - 2d_G(v)^4\end{aligned} \tag{6.1}$$

This expression is same as that of Theorem 10 except for $p \geq 3$, $A^p(v,v)$ is substituted by $\alpha_p(v).C^p(i,i)$ where $V_i \ni v$. Note that $A^2(v,v) = d_G(v)$.

We construct a summary $H$ of $G$ by randomly partitioning $V(G)$ into $t$ parts. There are better techniques [7, 25, 26] for graph summarization that might result in enhanced estimates.

### 6.2.1 Proposed Walk-8 algorithm

We select a subset $S$ that approximately maximizes $score_8(S)$ as given in (3.2). In Algorithm 8, Line 3 computes $W$ vector using (6.1) and $W[i]$ is the estimated number of walks of length 8 containing vertex $v_i$. In each iteration of Lines 7-15, we greedily extend $S$ by adding a node with the highest score (Line 11). Line 13 excludes nodes already selected in $S$ from further consideration.

**Algorithm 8** : WALK-8($A$,$k$,$t$)

```
1: $S \leftarrow \emptyset$
2: $W_2, Score \leftarrow$ ZEROS($n$)
3: $W \leftarrow$ ESTIMATEWALKS($A, t$)  ▷ compute approx. count of walks using super graph of order $t$ based on Eq. (6.1)
4: $\gamma \leftarrow \max_i W[i]$
5: for $i = 1$ to $n$ do
6:     $W_2[i] \leftarrow \gamma W[i]^2$
7: for $i = 1$ to $k$ do
8:     $\mathbf{u} \leftarrow A[:, S] * W[S]$
9:     for $j = 1$ to $n$ do
10:        if $j \notin S$ then
11:            $Score[j] \leftarrow W_2[j] - 2\mathbf{u}[j]W[j]$
12:        else
13:            $Score[j] \leftarrow -1$
14:    $maxNode \leftarrow \arg\max_j Score[j]$
15:    $S \leftarrow S \cup \{maxNode\}$
16: return $S$
```

### 6.2.2 Runtime analysis of WALK-8

We derive analytical bounds on the runtime of Algorithm 8. Partitioning $G$ into $t$ supernodes takes $O(n)$ time as it can be done with a linear scan on $V(G)$ to put nodes in respective buckets (supernodes). Computing the summary graph (populating the weighted adjacency matrix, $C$) requires traversing the edges $E(G)$ and incrementing the appropriate entry of $C$. This takes a total of $O(m)$ time, where $m = |E(G)|$. The powers of $C$ matrix can be computed in $O(t^3)$ time. Thus ESTIMATEWALKS function takes $O(n + m + t^3)$ time. Line 4 and the first for loop (Lines 5-6) takes $O(n)$ steps. An iteration of the inner for loop (Lines 9-13) takes $O(n + nk)$ and Line 14 takes $O(n)$ steps. This shows that the outer loop (Line 7-15) takes $\sum_{i=1}^{k} O(n + nk) = O(nk^2)$. Therefore, Algorithm 8 takes total $O(n + m + t^3 + nk^2)$ time.

## 6.3 Experimental evaluation

We present the results of the detailed experimentation of our proposed solution in this section. Experiments are performed on several real-world datasets to analyze the performance of our method and results are compared with NETSHIELD[1], the state-of-the-art algorithm, to evaluate quality, scalability and efficiency. NETSHIELD computes the score of each node using the eigenvector corresponding to the largest eigenvalue $\lambda_{max}$ of the original graph. WALK-6 and WALK-8 versions of our algorithm select nodes for immunization based on 6-walks and 8-walks respectively passing through each node.

We evaluate the performance of our algorithm across a range of budgets for the number of nodes to be immunized in the graphs and different counts of supernodes for approximation. First, we evaluate the quality of our approximation technique. To show that our approach maximally reduces the spread of the virus across the graph, we give results for the virus spread simulation on graphs immunized by NETSHIELD, WALK-6 and WALK-8. Furthermore, we measure quality in terms of the reduction in $\lambda_{max}$ (*vulnerability*) of the graph after immunizing the set $S$ of selected nodes. We report results using *eigendrop percentage*, which is $\frac{\Delta\lambda(S)}{\lambda_{max}(A)} \times 100$. Finally, we give runtime comparisons for the above-mentioned techniques.

We performed experiments on a standard desktop machine with 3.6 GHz Intel Core i7-7700 and 8 GB of main memory. The MATLAB code for our algorithm is available [2] for reproducibility and further experimentation.

### 6.3.1 Datasets

Experiments are performed on real-world graphs of order ranging from a few thousands to a few millions nodes. All graphs are undirected and unweighted. HEP-TH[3] is a collaboration network of High Energy Physics - Theory category extracted from the e-print arXiv. A node in the network represents an author and an edge between two authors shows collaboration between them. Face-

[1]https://www.dropbox.com/s/aaq5ly4mcxhijmg/Netshieldplus.tar
[2]`https://www.dropbox.com/sh/n7hwjc4imh62pe6/AADCyHG7uMGX6o9xtr1pdH6Qa?dl=0`
[3]https://snap.stanford.edu/

Table 6.2: Statistics of Datasets

| Network | Number of nodes | Number of edges | $\lambda_{max}(A)$ |
|---|---|---|---|
| HEP-TH | 9,877 | 25,998 | 31.03 |
| Facebook | 4,039 | 88,234 | 162.37 |
| Gowalla | 196,591 | 950,327 | 170.94 |
| Dblp | 317,080 | 1,049,866 | 115.85 |
| Amazon | 334,863 | 925,872 | 23.98 |
| AA | 418,236 | 2,753,798 | - |
| Youtube | 1,134,890 | 2,987,624 | 210.40 |
| Skitter | 1,696,415 | 11,095,298 | 670.35 |

book[1] graph shows the friendship network among users in which people are represented as nodes and relationships among two users are shown as edges.

To test our algorithm on large networks we use five different real-world graphs as given in Table 6.2. Gowalla[1] dataset shows friendship relations in a location-based social network. Amazon[1] is a co-purchasing graph of products where each node is a product and there is an edge between two nodes if the products are purchased by a user in a single basket. Dblp[1] is a co-authorship network in which two authors are connected if they have co-authored at least one publication. Youtube[1] graph shows the friendship network of users in the Youtube social network. Skitter[1] is an internet topology network where nodes correspond to autonomous systems and communication between them constitutes edges.

Table 6.3: Statistics of AA subgraphs

| Network | Number of nodes | Number of edges | $\lambda_{max}(A)$ |
|---|---|---|---|
| Applied Mathematics and Computing (AMC) | 18,371 | 24,224 | 10.99 |
| Decision Support Systems (DSS) | 4,926 | 14,660 | 12.0 |
| Ecological Informatics (EI) | 1,990 | 4,913 | 16.68 |
| Communication ACM | 11,476 | 16,687 | 32.90 |

The dataset AA[4] is a co-authorship network extracted from DBLP archive data. We select $4$ different smaller co-authorship subgraphs each corresponding to manuscripts in a distinct journal. Node count goes up to a few thousands and edge count goes up to a few ten thousands for extracted

[4]http://dblp.uni-trier.de/xml/

subgraphs. Details of the subgraphs of AA data set are provided in Table 6.3.

1185 ### 6.3.2 Approximation quality of WALK-8

In order to evaluate the goodness of our approximate method, we compare it with the exact solution as described in Theorem 10. The exact number of closed walks of length $8$ can be computed using the original adjacency matrix $A$ as given in Theorem 10 instead of using a summary graph. We analyze the quality of our approximation method by comparing the eigendrop percentages achieved using the exact and approximate method. We report comparison results of the exact solution with the summary graphs of order $\{100, 500, 1000\}$. 1190

It is clear from Figure 6.1 that the performance of our approximate method improves with the

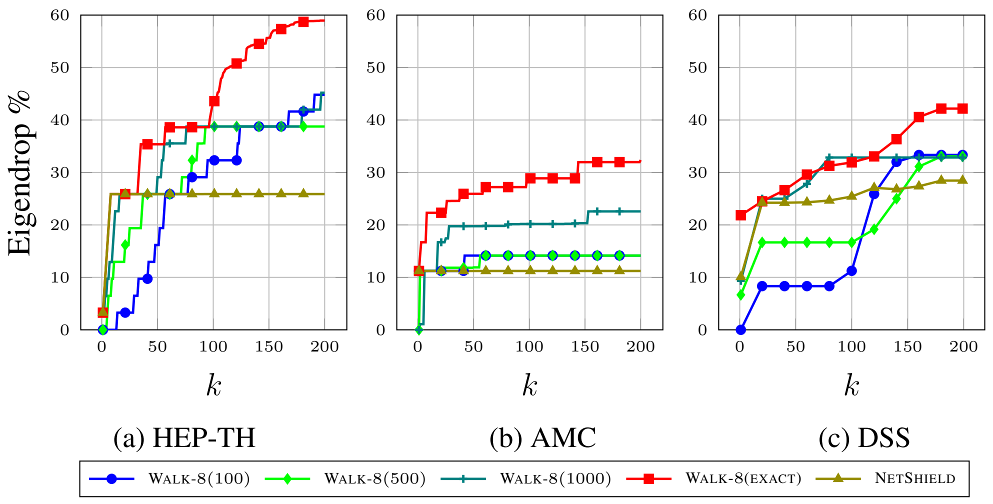


Figure 6.1: The effect of the order of summary graph on the quality of the approximation. Eigendrop percentages using different numbers of supernodes have been reported (WALK-8$(t)$, where $t$ is the number of supernodes). It is clear that as $t$ increases, the quality of approximation tends to match with that of the exact solution.

increase in the number of supernodes in the summary graph. As the order of the summary graph increases, the achieved benefit tends to match with that of the exact solution. Note that we compute the exact number of walks for small graphs having the order of a few thousands only as it is computationally infeasible to compute the exact solution for large graphs. 1195

### 6.3.3 Virus spread simulation

Another criterion used for quality evaluation is to estimate the spread of virus propagation in the immunized version of the graph. We use SIR virus propagation model to observe the spread of contagion after immunizing a small subset ($\sim$ 5 %) of nodes in a graph. Let $s = \lambda_{max} \times \beta/\delta$ be the virus strength (larger value of $s$ corresponds to more strength of virus while the virus gradually dies out if $s \leq 1$), where $\beta$ and $\delta$ denote the infection and recovery rate respectively. In our experimentation, we immunize $k$ nodes in a graph and infect all the nodes in the immunized version of the graph. We then observe the spread of the virus under different virus strengths with varying values of $\beta$ and $\delta$. Results in Figure 6.2 show that the graphs immunized by our approach have less number of infected nodes as compared to NETSHIELD. We report the average of 3 runs of experiments to mitigate the effect of randomness.

### 6.3.4 Eigendrop percentage comparison

We compare the quality of approximate versions of our algorithms with NETSHIELD in terms of eigendrop and results are shown in Figure 6.3. For smaller graphs and subgraphs of AA which consist of a few thousand nodes, a budget of up to $100$ nodes is used and for large graphs with more than $100,000$ nodes, we immunize up to $1000$ nodes. We have used summary graphs with different orders $(100, 500, 1000)$ to perform experiments. Time complexity increases as the number of supernodes increases but we observe that there is a proportionately minor improvement in the quality of solution for increasing order of graph after a certain threshold is reached. For smaller graphs, we report results for supernode count of $500$ and for large graphs, the number of supernodes is set to $1000$.

We observe that the immunizing quality of our algorithm clearly outperforms NETSHIELD in terms of eigendrop. The improvement in quality of solution is particularly evident on large graphs Gowalla Figure 6.3c, Youtube Figure 6.3d, and Skitter Figure 6.3e. For reasonably large budget, WALK-8 outperforms both NETSHIELD and WALK-6. Experiments also reveal that NETSHIELD performs better than our approach for very small values of budget $k$ but as the count of nodes to be immunized increases, its effectiveness degrades.

### 6.3.5 Run time comparison

We also present comparable computational cost while achieving much better quality as one of the merits of our algorithm as discussed in the theoretical time complexity in Section 3.3. Comparison of runtimes of NETSHIELD, WALK-6 and WALK-8 is provided in Figure 6.4. Results show that the runtime of our algorithm matches with that of NETSHIELD. The results are reported with $1000$ supernodes ($t$) in summary graphs.

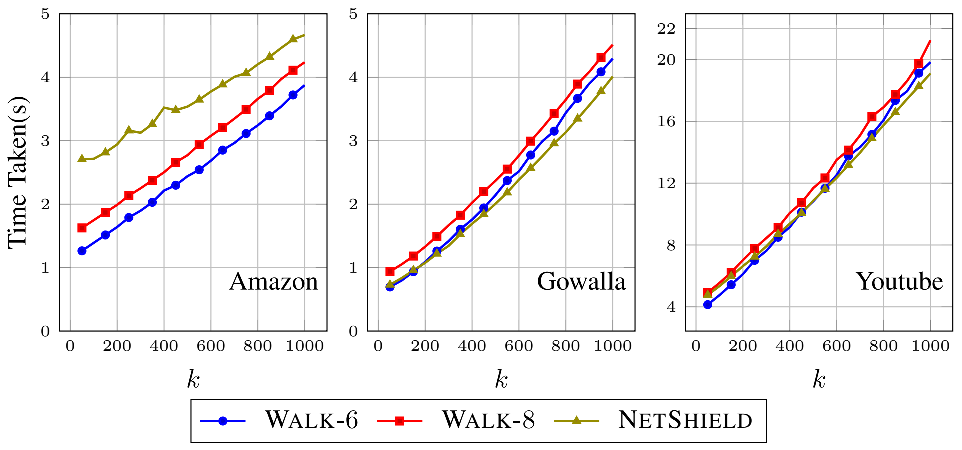
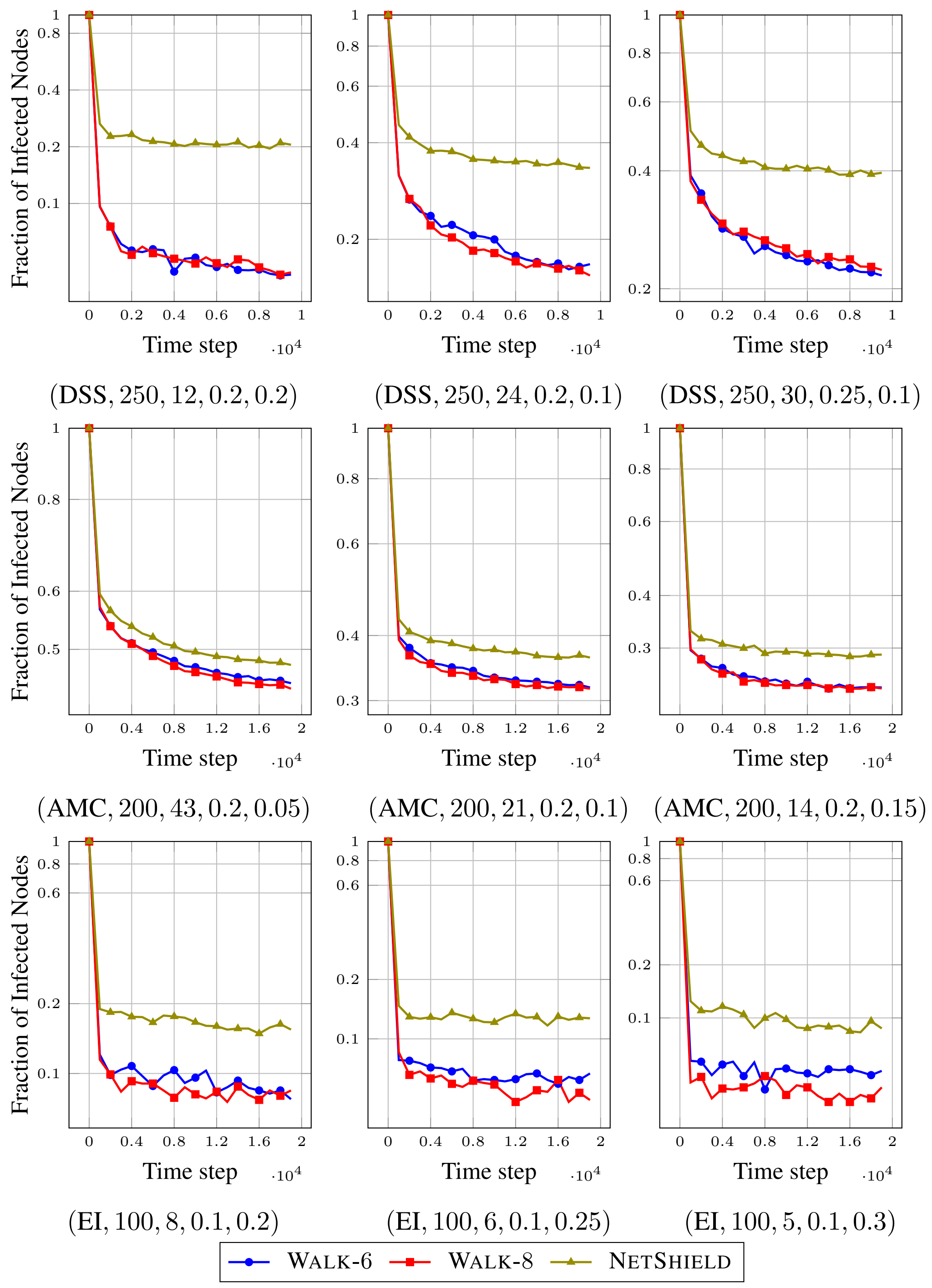


Figure 6.4: Comparison of time taken (in seconds) to immunize graphs using NETSHIELD, WALK-6 and WALK-8 approach against the number of nodes to be immunized ($k$). Results are reported on summaries with $1000$ supernodes.

Recall that runtime of our algorithm is $O(n + |E(G)| + t^3 + nk^2)$, where the first three terms comprise runtime of constructing a summary of order $t$ and computing the $\mathcal{W}_8(v, G)$ for all $v \in V(G)$, while the last term ($nk^2$) is the runtime to select the best $k$ nodes (NETSHIELD also requires $O(nk^2)$ for this task). Hence, runtime of our algorithm depends quadratically only on $k$, which generally is a small constant. We note that our runtime is superior to that of NETSHIELD in the sense that in relatively less time we achieve significantly more eigendrop even for a small value of $t$ (see Figure 6.1).

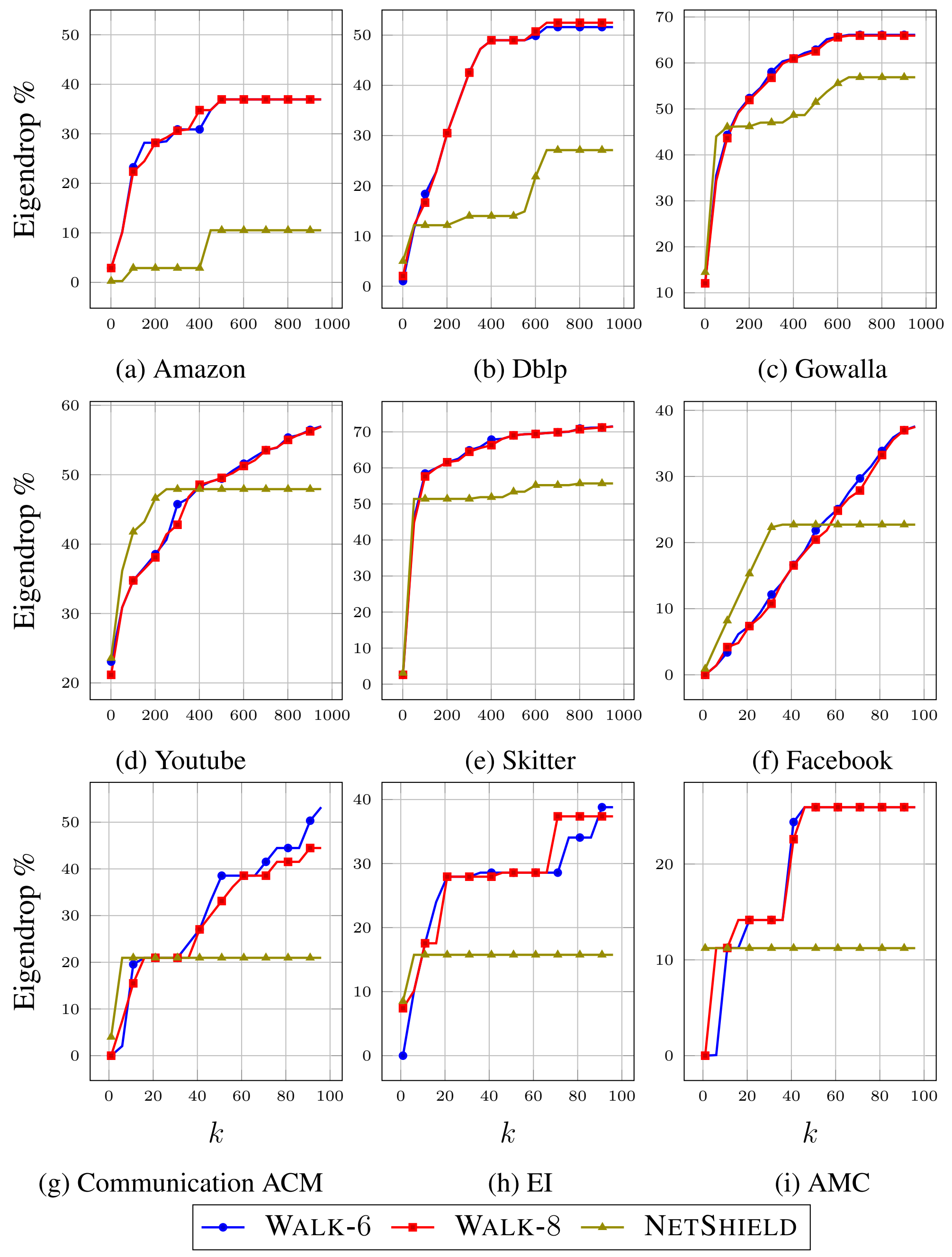


$(\mathrm{DSS}, 250, 12, 0.2, 0.2)$ $(\mathrm{DSS}, 250, 24, 0.2, 0.1)$ $(\mathrm{DSS}, 250, 30, 0.25, 0.1)$

$(\mathrm{AMC}, 200, 43, 0.2, 0.05)$ $(\mathrm{AMC}, 200, 21, 0.2, 0.1)$ $(\mathrm{AMC}, 200, 14, 0.2, 0.15)$

$(\mathrm{EI}, 100, 8, 0.1, 0.2)$ $(\mathrm{EI}, 100, 6, 0.1, 0.25)$ $(\mathrm{EI}, 100, 5, 0.1, 0.3)$

Figure 6.2: Virus propagation simulation for varying virus strength $s$ on the immunized version of graphs. Caption of each plot represents (graph name, number of immunized nodes $k$, $s$, infection rate $\beta$, recovery rate $\delta$). Initially, all the nodes in the graphs were contaminated and the plots show the fraction of infected nodes ($y$-axis logged scale) as the time proceeds. We select nodes for immunization using a summary graph having $500$ supernodes.



(a) Amazon (b) Dblp (c) Gowalla

(d) Youtube (e) Skitter (f) Facebook

(g) Communication ACM (h) EI (i) AMC

Figure 6.3: Comparison of NETSHIELD, WALK-6 and WALK-8 in terms of eigendrop percentages ($y$-axis) against budget $k$, number of nodes immunized, ($x$-axis). WALK-6 and WALK-8 achieve significantly higher eigendrop for increasing $k$. Results in (a)-(e) are computed using $1000$ supernodes while in (f)-(i) experiments are performed using summary graph of order $= 500$. The range for $k$ is chosen keeping in view the number of nodes in the host graphs.

## 6.4 Summary

In this work, we address the problem of finding a small subset of nodes in a network whose immunization results in a significant reduction in network vulnerability towards the spread of undesirable content. We explored the relationships between spectral and graph-theoretic properties of networks and exploit these relationships to design an efficient algorithm to find crucial nodes in the network. We select a subset of nodes for immunization based on the number of closed walks of length $8$. With the use of easily computable local graph properties and approximation techniques, the running time of our technique is linear in the size of the graph. Thus, our method is scalable and can be applied to large graphs. Experiments on large real-world networks suggest that our algorithm provides better results than previously employed methods and is significantly faster in terms of time complexity. The approximation quality comparison shows that our method is a close approximation of the exact solution. Experimental results for various quality measures like virus spread simulation, reduction in network vulnerability and the run time comparison show that our method performs better than the state-of-the-art solution.

In the next chapter, we give our solution to graph summarization problem. Initially, we aim to devise a summarization method as a sub-routine to improve the quality of approximate immunization algorithm. However, we realized that graph summarization, itself, is a standalone area of research in the domain of graph theory. So, we include our contribution to graph summarization problem as a separate line of research and we are sure that the use of specialized graph summarization approaches will improve the approximation quality of our immunization algorithm.

# Chapter 7

# Scalable Approximation Algorithm for Graph Summarization

## 7.1 Introduction

Graphs analysis is a fundamental task in various applications. Graphs with millions of nodes and billions of edges are ubiquitous in many research fields such as e-commerce, social networks analysis, bioinformatics, internet of things, etc [75, 76]. The magnitude of these graphs poses significant computational challenges for graph processing. A practical solution is to compress the graph into a summary that retains the essential structural information and the important properties of the original graph. Processing and analyzing the summary graph is significantly faster and also reduces the storage and communication overhead.

Graph summarization has various applications in a variety of domains. It plays a pivotal role in privacy preservation while allowing to draw insights from a network [77–80]. Graph summarization is also used to protect networks from infection spread and identify source nodes of infection [81]. he summary of a graph is used to estimate the combinatorial trace of the original graph for immunizing a select few nodes to minimize the infection spread in the graph [22, 23, 82, 83]. Moreover, summarization techniques help generate graph descriptors, mapping graphs to a fixed dimensional vector space [84, 85]. These descriptors make classification, clustering, and other graph processing tasks computationally efficient. Furthermore, a summary makes visualization of very large graphs possible by removing less useful information [86, 87].

The summary graph is represented as a supergraph with weights on supernodes and superedges. A supernode is a subset of nodes and the weight of a supernode is the density of the subgraph induced by the subset. While an edge between two supernodes is termed as superedge and the weight on a superedge is the density of the bipartite subgraph induced by the subset of nodes in respective supernodes. Broadly there are two types of graph summarization approaches: $i)$ *lossless* and $ii)$ *lossy*. In lossless graph summarization, the original graph can be reconstructed exactly from the summary graph. The exact reconstruction is done by storing extra information as *edge corrections*, the edges to be inserted or deleted in the reconstructed graph. Storing edge corrections along with the summary poses an additional memory overhead. In *lossy* graph summarization, the original graph is approximately reconstructed from the summary. The quality of a lossy summary is measured by the extent to which the original graph can be reconstructed. For lossy summaries, *reconstruction error*, a widely adopted quality measure of a summary, quantifies a norm of the structural difference between the original and the reconstructed graph. Another goodness criterion of the summary is the accuracy of answers to queries related to graph topology.

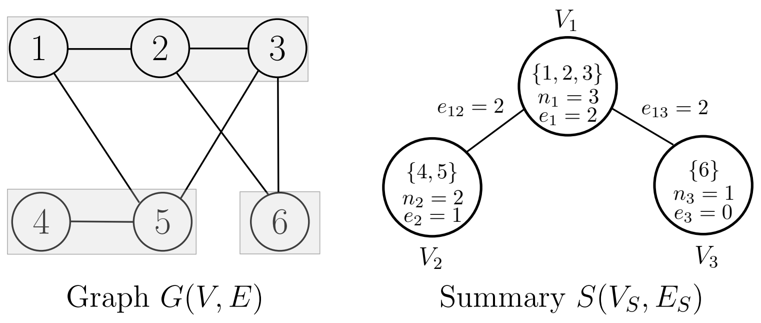
As the number of possible summaries (vertex set partitions) is exponential, computing the *'best'* summary is challenging [7, 10]. A well-known method, GRASS [7] uses an agglomerative approach to form a summary of the given graph. Initially, each node is considered as a supernode, and in every iteration, two supernodes are merged until the desired number of supernodes are formed. In this setting, selecting a pair of nodes to merge and computing the error incurred after merging a pair are computationally expensive challenges. At iteration $t$, let $n(t)$ be the number of supernodes in the summary, the total possible number of pairs of supernodes is $\binom{n(t)}{2}$. To reduce this search space, GRASS proposed a sampling approach in which only $O(n(t))$ pairs of nodes are considered randomly for merging. The selection and merging of the best pair are made in $O(n(t))$. The overall runtime to summarize the graph on $n$ nodes is $O(n^3)$, which is infeasible even for graphs with a few thousand nodes. On the other hand, S2L [10], represents each vertex by an $n$-dimensional vector and applies vector-clustering to find supernodes. The runtime of S2L to compute a summary with $k$ supernodes is $O(n^2 t)$, where $t$ is the number of iterations before convergence. This runtime is also infeasible for large graphs.

In this paper, we devise a lossy summarization approach that iteratively computes the summary of the input graph in an agglomerative fashion. In each iteration, we consider a pair of nodes from the logarithmic-sized sample for merging. We assign a weight to each node to estimate the contribution of the node in the score of pairs containing it. Based on the node weights, we probabilistically select a sample of node pairs of better quality and yields summaries of comparable quality with the logarithmic-sized sample. For each pair in the sample, we define a score that quantifies the goodness of the pair for merging. To improve the efficiency, we approximate this score through a closed-form expression and storing constant extra variables at each node. The overall runtime of our approach to compute a summary is $O(n \log^2 n)$. We perform experiments on several benchmark real-world networks to demonstrate the effectiveness and efficiency of our approach as compared to GRASS and S2L.

The chapter is organized as follows. In section 7.2 we formally define the problem with its background. We present our algorithm along with its analysis in section 7.3. In section 7.4 we report results of experimental evaluation of our algorithm on several graphs. We also provide comparisons with existing solutions both in terms of runtime and quality.

## 7.2 Problem Definition

Given a graph $G = (V, E)$ on $n$ vertices, let $A$ be its adjacency matrix. For $k \in \mathbb{Z}$, a summary of $G$, $S = (V_S, E_S)$ is a weighted graph on $k$ vertices. Let $V_S = \{V_1, \ldots, V_k\}$, each $V_i \in V_S$ is referred to a supernode and represents a subset of $V$. More precisely, $V_S$ is a partition of $V$, i.e. $V_i \subset V$ for $1 \leq i \leq k$, $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{k} V_i = V$. Each supernode $V_i$ is associated with two integers $n_i = |V_i|$ and $e_i = |\{(u, v)|u, v \in V_i, (u, v) \in E\}|$. For an edge $(V_i, V_j) \in E_S$ (known as superedge), let $e_{ij}$ be the number of edges in the bipartite subgraph induced between $V_i$ and $V_j$, i.e. $e_{ij} = |\{(u, v)|u \in V_i, v \in V_j, (u, v) \in E\}|$. Given a summary $S$, the graph $G$ is


1
2
3
4
5
6
Graph $G(V, E)$
$V_1$
$\{1,2,3\}$ $n_1 = 3$ $e_1 = 2$
$e_{12} = 2$
$e_{13} = 2$
$\{4,5\}$ $n_2 = 2$ $e_2 = 1$
$\{6\}$ $n_3 = 1$ $e_3 = 0$
$V_2$
$V_3$
Summary $S(V_S, E_S)$

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 2/3 & 2/3 & 1/3 & 1/3 & 2/3 \\ 2 & 2/3 & 0 & 2/3 & 1/3 & 1/3 & 2/3 \\ 3 & 2/3 & 2/3 & 0 & 1/3 & 1/3 & 2/3 \\ 4 & 1/3 & 1/3 & 1/3 & 0 & 1 & 0 \\ 5 & 1/3 & 1/3 & 1/3 & 1 & 0 & 0 \\ 6 & 2/3 & 2/3 & 2/3 & 0 & 0 & 0 \end{array}$$

$$\bar{A}$$

Figure 7.1: A graph G having 6 vertices and its summary graph $S$ are shown. The summary graph consists of 3 supernodes and each supernode $V_i$ in $S$ stores the number of nodes ($n_i$) and number of edges ($e_i$) between nodes in $V_i$. Each superedge $e_{ij}$ also contains the number of edges between $V_i$ and $V_j$. $\bar{A}$ shows the expected adjacency matrix constructed from $S$.

approximately reconstructed by the expected adjacency matrix, $\bar{A}$, where $\bar{A}$ is a $n \times n$ matrix with

$$\bar{A}(u, v) = \begin{cases} 0 & \text{if } u = v \\ \frac{e_i}{\binom{n_i}{2}} & \text{if } u, v \in V_i \\ \frac{e_{ij}}{n_i n_j} & \text{if } u \in V_i, v \in V_j \end{cases}$$

The quality of a summary $S$ is assessed by $l_p$-norm of element-wise difference between $\bar{A}$ and $A$.

**Definition 11.** *($l_p$-Reconstruction Error ($RE_p$)): The (unnormalized) $l_p$ reconstruction error of a summary $S$ of a graph G is*

$$RE_p(G|S) = RE_p(A|\bar{A}) = \left( \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} |\bar{A}(i, j) - A(i, j)|^p \right)^{1/p} \tag{7.1}$$

Note that the case $p = 1$ considered in [7] and $p = 2$ considered in [10] are closely related to each other. In this paper we use $p = 1$ and refer to $RE_1(G|S)$ as $RE(G|S)$. The derivation of following closed form expression of $RE(G|S)$ is given in Section 7.3.

$$RE(G|S) = RE(A|\bar{A}) = \sum_{i=1}^{k} 4e_i - \frac{4e_i^2}{\binom{n_i}{2}} + \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} 2e_{ij} - \frac{2e_{ij}^2}{n_i n_j} \tag{7.2}$$

Formally, we address the following problem.

**Problem 6.** *Given a graph $G(V, E)$ and a positive integer $k \leq |V|$, find a summary $S$ for $G$ with $k$ super nodes such that $RE(G|S)$ is minimized.*

Another measure to assess quality of a summary $S$ of $G$ is by the accuracy of answers of queries about structure of $G$ based on $S$ only. In the following we list how certain queries used in the literature are answered from $S$.

**Adjacency Queries:** Given two vertices $u, v \in V$, the query whether $(u, v) \in E$ is answered with $\bar{A}(u, v)$. This can either be interpreted as the expected value of an edge being present between $u$ and $v$ or as returning a 'yes' answer based on the outcome of a biased coin with probability $\bar{A}(u, v)$.

**Degree Queries:** Given a vertex $v \in V$, the query about degree of $v$ is answered as $\bar{d}(v) = \sum_{j=1}^{n} \bar{A}(v, j)$.

**Eigenvector-Centrality Queries:** Eigenvector-centrality of a vertex $v$, $p(v)$ measures the relative importance of $v$ [88]. For a vertex $v \in V$, this query is answered as $\bar{p}(v) = \frac{\bar{d}(v)}{2|E|}$, where $E$ represents edges in the graph.

**Triangle density queries:** Let $t(G)$ be the number of triangles in $G$. $t(G)$ is estimated from $S$ by counting the expected number of triangles within each super node, the expected number of triangles made with one vertex in one supernode and two in another, and that made with one vertex each from three different super nodes. More precisely, this query is answered as follows. Let $\pi_i = \frac{e_i}{\binom{n_i}{2}}$ and $\pi_{ij} = \frac{e_{ij}}{n_i n_j}$, then $\bar{t}(G)$, the estimate for $t(G)$, is

$$\sum_{i=1}^{k} \left[ \binom{n_i}{3} \pi_i^3 + \sum_{j=i+1}^{k} \left( \pi_{ij}^2 \left[ \binom{n_i}{2} n_j \pi_i + \binom{n_j}{2} n_i \pi_j \right] + \sum_{l=j+1}^{k} n_i n_j n_l \pi_{ij} \pi_{jl} \pi_{il} \right) \right].$$

## 7.3 Algorithm

Given a graph $G$ and an integer $k$ our algorithm produces a summary $S$ on $k$ super nodes as follows. Let $S_{t-1}$ be the summary before iteration $t$ with $n(t-1)$ super nodes, i.e. $S_0 = G$, and let $\bar{A}_t$ be the expected adjacency matrix of $S_t$. For $1 \leq t \leq n-k$, we select a pair of supernodes $(u, v)$ and merge it to get $S_t$. To select an approximately optimal pair, we define weight of each node $v$

that closely estimates the contribution of this node to score of pairs $(v, *)$. We randomly sample nodes for each pair with probability proportional to their weights and evaluate score of the pairs. We derive a closed form formula to evaluate score of a pair. Furthermore, in this form these scores can be approximately computed very efficiently. Based on approximate score, we select the best pair in the sample and merge it to get $S_t$. In what follows, we discuss implementation of each of these subroutines and their analyses.

**Lemma 12.** *A pair $(u, v)$ of nodes in $S_t$, can be merged to get $S_{t+1}$ in time $O(deg(u) + deg(v))$.*

*Proof.* In the adjacency list format, one needs to iterate over neighbors of each $u$ and $v$ and record their information in a new list of the merged node. However, updating the adjacency information at each neighbor of $u$ and $v$ could potentially lead to traversal of all the edges. To this end, as a preprocessing step, for each $(x, y)$, in the adjacency list of $x$ at node $y$, we store a pointer to the corresponding entry in the adjacency list of $y$. With this constant (per edge) extra book keeping, we can update the merging information at each neighbor in constant time by traversing just the list of $u$ and $v$ (see Figure 7.2). It is easy to see that this preprocessing of storing the pointers corresponding to each edge can be done in time $O(|E|)$ once at the initialization. $\square$

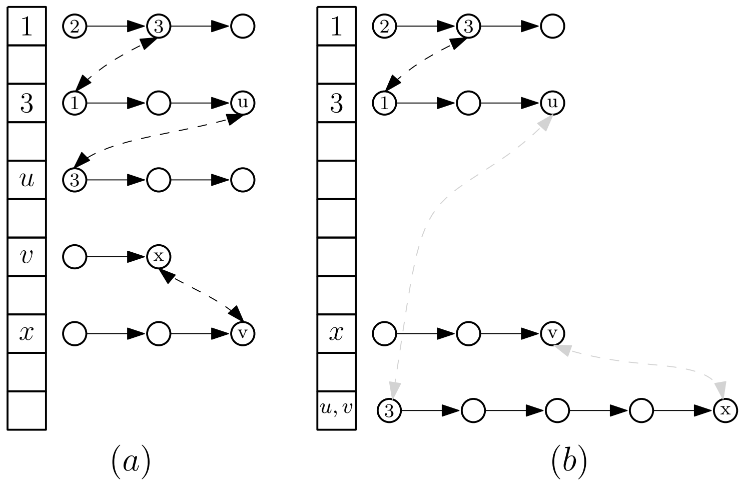


Figure 7.2: (a) The adjacency list of a graph is shown in which neighbors store the pointer to the corresponding node in their adjacency lists. (b) When two nodes $u$ and $v$ are merged, there is a need of informing the neighbors of merged nodes about the merging (c) The merging information of $u$ and $v$ is updated at their neighbors by traversing the pointers stored at the entries in the adjacency list of the merged node.

**Derivation of closed-form expression for RE:** We derive a closed-form expression to efficiently compute $RE(G|S)$, which is the total error incurred in the estimation of $A$ from $S$ only. We first calculate the contribution of $V_a \in V_S$ in $RE(G|S)$ and then extend it to a general expression that sums the contribution of all the supernodes in $RE(G|S)$. The cumulative contribution of all the supernodes sums to $RE(G|S)$. We have $V_a = \{v_{a1}, v_{a2}, \cdots, v_{an_a}\}$, where $v_{aj} \in V, a \leq k, j \leq n_a$. $V_a$ contributes to all the entries/edges in $\bar{A}$, which have one or both the endpoints in $V_a$. We predict the presence of each of the possible internal edge (edge with both the endpoints in $V_a$) as $\frac{e_a}{\binom{n_a}{2}}$. However, there are a total of $e_a$ edges in $V_a$ and analogously, $2e_a$ corresponding positions in $A$ have value 1. The error at these $2e_a$ locations in $A$ and $\bar{A}$ is $2e_a(1 - \frac{e_a}{\binom{n_a}{2}})$. The error at the remaining $2[\binom{n_a}{2} - e_a]$ positions is $2(\binom{n_a}{2} - e_a)(\frac{e_a}{\binom{n_a}{2}})$. In case of superedges $(V_a, V_b) \in E_S, V_a, V_b \in V_S, a, b \leq k, a \neq b$, we predict the presence of an edge $(u, v), v \in V_a, u \in V_b, u, v \in V$ as $\frac{e_{ab}}{n_a n_b}$. Doing the similar calculation as above, the contribution of $V_a$ in the error at the locations $(u, v), u \in V_a, v \in V_b$ is $e_{ab}(1 - \frac{e_{ab}}{n_a n_b}) + (n_a n_b - e_{ab})(\frac{e_{ab}}{n_a n_b})$. The total error introduced by supernode $V_a$ is $2e_a(1 - \frac{e_a}{\binom{n_a}{2}}) + 2(\binom{n_a}{2} - e_a)(\frac{e_a}{\binom{n_i}{2}}) + e_{ab}(1 - \frac{e_{ab}}{n_a n_b}) + (n_a n_b - e_{ab})(\frac{e_{ab}}{n_a n_b})$. With some simplification, the total error accumulated while reconstructing $A$ using summary $S$ with $k$ supernodes is given as

$$RE(G|S) = RE(A|\bar{A}) = \sum_{i=1}^{k} 4e_i - \frac{4e_i^2}{\binom{n_i}{2}} + \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} 2e_{ij} - \frac{2e_{ij}^2}{n_i n_j} \tag{7.3}$$

**Score computation of a pair of nodes:** The next important step is to determine the *goodness* of a pair $(a, b)$. This can be done by temporarily merging $a$ and $b$ and then evaluating (7.1) or (7.3) respectively taking $O(n^2)$ and $O(n(t))$. For a pair of nodes $(a, b)$ in $S_{t-1}$, let $S_t^{a,b}$ be the graph obtained after merging $a$ and $b$. We define score of a pair $(a,b)$ to be

$$\begin{aligned}
score_t^{RE}(a,b) =& RE(G|S_{t-1}) - RE(G|S_t^{a,b}) \\
=& 4e_a + 4e_b - \frac{4e_a^2}{\binom{n_a}{2}} - \frac{4e_b^2}{\binom{n_b}{2}} + \sum_{\substack{i=1\\ i\neq a,b}}^{k} 4e_i - \frac{4e_i^2}{\binom{n_i}{2}} + 2(2e_{ab} - \frac{2e_{ab}^2}{n_a n_b}) + \sum_{\substack{i=1\\ i\neq a,b}}^{k} 4e_{ai} \\
& - \frac{4e_{ai}^2}{n_a n_i} + \sum_{\substack{i=1\\ i\neq a,b}}^{k} 4e_{bi} - \frac{4e_{bi}^2}{n_b n_i} + \sum_{\substack{i,j=1\\ i,j\neq a,b}}^{k} 2e_{ij} - \frac{2e_{ij}^2}{n_i n_j} - 4\big(e_a + e_b + e_{ab}\big) \\
& + \frac{4\big(e_a + e_b + e_{ab}\big)^2}{\binom{n_a+n_b}{2}} - \sum_{\substack{i=1\\ i\neq a,b}}^{k} 4e_i - \frac{4e_i^2}{\binom{n_i}{2}} \\
& - 4\sum_{\substack{i=1\\ i\neq a,b}}^{k} \left(\big(e_{ai} + e_{bi}\big) - \frac{\big(e_{ai}+e_{bi}\big)^2}{\big(n_a+n_b\big)n_i}\right) - \sum_{\substack{i,j=1\\ i,j\neq a,b}}^{k} 2e_{ij} - \frac{2e_{ij}^2}{n_i n_j} \\
=& - \frac{4e_a^2}{\binom{n_a}{2}} - \sum_{\substack{i=1\\ i\neq a}}^{n(t)} \frac{4e_{ai}^2}{n_a n_i} + \frac{4e_{ab}^2}{n_a n_b} - \frac{4e_b^2}{\binom{n_b}{2}} - \sum_{\substack{i=1\\ i\neq b}}^{n(t)} \frac{4e_{bi}^2}{n_b n_i} + \frac{4\big(e_a + e_b + e_{ab}\big)^2}{\binom{n_a+n_b}{2}} \\
& + \frac{4}{\big(n_a+n_b\big)} \sum_{\substack{i=1\\ i\neq a,b}}^{n(t)} \Big(\frac{e_{ai}^2}{n_i} + \frac{e_{bi}^2}{n_i} + \frac{2e_{ai}e_{bi}}{n_i}\Big) \qquad (7.4)
\end{aligned}$$

**Fact 5.** *Since $S_{t-1}$ is fixed, minimizing $RE(G|S_t^{a,b})$ is equivalent to maximizing $score_t(a,b)$.*

**Remark 1.** *Except for the last summation in* (7.4) *all other terms of $score_t(a,b)$ can be computed in constant time. Since $n_a, n_b, e_a$, and $e_b$ are already stored at $a$ and $b$, this can be achieved by storing an extra real number $D_a$ at each super node $a$ such that, $D_a = \sum_{\substack{i=1\\ i\neq a}}^{n(t)} \frac{e_{ai}^2}{n_i}$. Note that $D_a$ can be updated in constant time after merging of any two vertices $x, y \neq a$, i.e. after merging $x, y$, while traversing their neighbors for $a$ we subtract $e_{xa}/n_x$ and $e_{ya}/n_y$ from $D_a$ and add back $(e_x + e_y)/(n_x + n_y)$ to it. This value can be similarly updated at the merged node too.*

The last summation in (7.4), $\sum_{\substack{i=1\\ i\neq a,b}}^{n(t)} \frac{2e_{ai}e_{bi}}{n_i}$, in essence is the inner product of two $n(t)$ dimensional vectors $\mathcal{A}$ and $\mathcal{B}$, where the $i^{th}$ coordinate of $\mathcal{A}$ is $\frac{e_{ai}}{\sqrt{n_i}}$ ($\mathcal{B}$ is similarly defined). Storing these vectors will take $O(n(t))$, moreover computing score will take time $O(n(t))$. However, $\langle \mathcal{A}, \mathcal{B}\rangle = \mathcal{A} \cdot \mathcal{B}$ can be very closely approximated with a standard application of *count-min* *sketch* [89].

**Theorem 13.** *(c.f [89] Theorem 2) For $0 < \epsilon, \delta < 1$, let $\left\langle \widehat{\mathcal{A}, \mathcal{B}} \right\rangle$ be the estimate for $\langle \mathcal{A}, \mathcal{B} \rangle$ using the count-min sketch. Then*

- $\left\langle \widehat{\mathcal{A}, \mathcal{B}} \right\rangle \geq \langle \mathcal{A}, \mathcal{B} \rangle$
- $Pr[\left\langle \widehat{\mathcal{A}, \mathcal{B}} \right\rangle < \langle \mathcal{A}, \mathcal{B} \rangle + \epsilon ||\mathcal{A}||_1 ||\mathcal{B}||_1] \geq 1 - \delta.\ \mathcal{A}, \mathcal{B}$

*Furthermore, the space and time complexity of computing $\left\langle \widehat{\mathcal{A}, \mathcal{B}} \right\rangle$ is $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$. While after a merge, the sketch can be updated in time $O(\log \frac{1}{\delta})$.*

Hence, for a pair of nodes $(a, b)$ in $S_{t-1}$, $score_t(a, b)$ can be closely approximated in constant time. The bounds on time and space complexity, though constants are quite loose in practice.

The next important issue the quadratic size of search space. This is a major hurdle to scalability to large graphs. We define weight of a node $a$ as

$$f(a) = -\frac{4e_a^2}{\binom{n_a}{2}} - \sum_{\substack{i=1 \\ i \neq a}}^{n(t)} \frac{4e_{ai}^2}{n_a n_i} \qquad w(a) = \begin{cases} \frac{-1}{f(a)} & \text{if } f(a) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

We select pairs by sampling nodes according to their weights so as the pairs selected will likely have higher scores. With this weighted sampling a sample of size $O(\log n)$ outperforms a random sample of size $O(n)$. Let $W = \sum_{i=1}^{n(t)} w(i)$ be the sum of weights, we select a vertex $w$ with probability $w(a)/W$. Weighted sampling though can be done in linear time at a given iteration. In our case it is very challenging since the population varies in each iteration; two vertices are merged into one and weights of some nodes also change. To overcome this challenge, we design special data structure $\mathcal{D}$ that has the following properties.

**Theorem 14.** *$\mathcal{D}$ can be implemented as a binary tree such that*

1. *it can be initially populated in $O(n)$,*
2. *a node can be sampled with probability proportional to its weight in $O(\log n)$,*

*3. inserting, deleting or updating a weight in $\mathcal{D}$ takes time $O(\log n)$.*

**Remark 2.** *We designed this data structure independently, but found out that it has been known to the statistics community since 1980 [90]. We note that this technique could have many applications in sampling from dynamic graphs.*

Algorithm 9 is our main summarization algorithm that takes as input $G$, integers $k$ (summary size), $s$ (sample size), $w$ and $d$ ( $w = \frac{1}{\epsilon}$ and $d = \log \frac{1}{\delta}$ are parameters for count-min sketch).

**Algorithm 9** : ScalableSumarization($G = (V, E)$, $k$, $w$, $d$)

```
1: $\mathcal{D} \leftarrow$ BUILDSAMPLINGTREE$(V, W, 1, n)$          ▷ $W[1 \ldots n]$ is initialize as $W[i] = w(v_i)$
2: while $G$ has more than $k$ vertices do
3:     $samplePairs \leftarrow$ GETSAMPLE$(\mathcal{D}, s)$          ▷ $s$ calls to Algorithm 12
4:     $scores \leftarrow$ COMPUTEAPPROXSCORE$(samplePairs)$          ▷ Uses (7.4) and Theorem 13
5:     $bestPair \leftarrow$ MAX$(scores)$
6:     MERGE$(bestPair)$          ▷ Lemma 12
7:     for each neighbor $x$ of $u, v \in bestPair$ do
8:         UPDATEWEIGHT$(x, \mathcal{D})$
```

For each vertex $a$ we maintain a variable $D_a$ (Remark 1). Hence the weight array can be initialized in $O(n)$ time using (7.5). By Claim 14, $\mathcal{D}$ can be populated in $O(n)$ time. By Claim 14, Line 3 takes $O(s \log n)$ time, by Theorem 13 and (7.4) Line 4 takes constant time per pair, and by Lemma 12 merging can be performed in $O(\Delta)$ time. Since delete and update in $\mathcal{D}$ takes time $O(\log n)$ and the while loop is executed $n - k + 1$ times, total runtime of Algorithm 9 is $O((n - k + 1)(s \log n + \Delta \log n)$. Generally $k$ is $O(n)$ (typically a fraction of $n$) and in our experiments we take $s$ to be $O(\log n)$ and $O(\log^2 n)$. Furthermore, since many real world graphs are very sparse, ($\Delta$ which is worst case upper bound is constant), we get that overall complexity of our algorithm is $O(n \log^2 n)$ or $O(n \log^3 n)$.

**Data Structure for sampling:** We implement $\mathcal{D}$ as a balanced binary tree, where leaf corresponds to (super) node in the graph and stores weight and id of the node (shown in Figure 7.3). Each internal node stores the sum of values of the two children. The value of the root is equal to $\sum_{i=1}^{n(t)} w(i)$. Furthermore, at each node in the graph we store a pointer to the corresponding leaf. We give pseudocode to construct this tree in Algorithm 10 along with the structure of a tree node.

By construction, it is clear that height of the tree is $\lceil \log n \rceil$ and running time of building the tree and space requirement of $\mathcal{D}$ is $O(n)$.

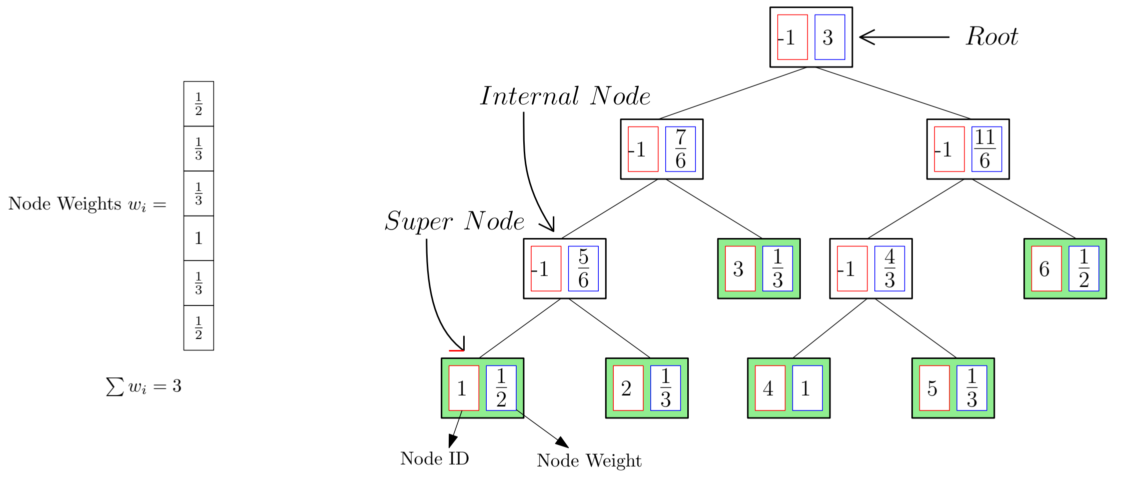


Figure 7.3: The tree data structure used to sample nodes based on node weights is shown.

**Algorithm 10** BuildSamplingTree($A$,$W$,$st$,$end$)

```
1: if $A[st] = A[end]$ then
2:     $leaf \leftarrow$ CreateNode()
3:     $leaf.weight \leftarrow W[st]$
4:     $leaf.vertexID \leftarrow A[st]$
5:     return $leaf$
6: else
7:     $mid = \frac{end+st}{2}$
8:     $left \leftarrow$ CreateNode()
9:     $left \leftarrow$ BuildSamplingTree($A, W, st, mid$)
10:    $right \leftarrow$ CreateNode()
11:    $right \leftarrow$ BuildSamplingTree($A, W, mid+1, end$)
12:    $parent \leftarrow$ CreateNode()
13:    $parent.weight \leftarrow left.weight + right.weight$
14:    $left.parent \leftarrow parent$
15:    $right.parent \leftarrow parent$
16:    return $parent$
```

**Structure** TreeNode

```
    int $vertexID$
    double $weight$
    TreeNode $*left$
    TreeNode $*right$
    TreeNode $*parent$
```

The procedure to sample a vertex with probability proportional to its weight using $\mathcal{D}$ is given in Algorithm 12. This takes as input a uniform random number $r \in [0, \sum_{i=1}^{n(t)} w(i)]$. Since it traverses a single path from root to leaf, the runtime of this algorithm is $O(\log n)$. The update procedure is very similar, whenever weight of a node changes, we start from the corresponding leaf (using the stored pointer to leaf) and change weight of that leaf. Following the parent pointers, we update weights of internal nodes to the new sum of weights of children. Deleting a node is very similar, it amounts to updating weight of the corresponding leaf to $0$. Inserting a node (the super node representing the merged nodes) is achieved by changing the weight of the first empty leaf in $\mathcal{D}$. A reference to first empty node is maintained as a global variable.

**Algorithm 12** :GetLeaf($r$,$node$)

```
1: if $node.left = $ NULL $\wedge node.right = $ NULL then
2:     return $node.vertexID$
3: if $r < node.left.weight$ then
4:     return GETLEAF($r, node.left$)
5: else
6:     return GETLEAF($r - node.weight, node.right$)
```

## 7.4 Evaluation

We evaluate the performance of our algorithm in terms of runtime, reconstruction error and accuracies of answers to queries on standard benchmark graphs [1]. We demonstrate that our algorithm substantially outperforms existing solutions, GRASS [7] and S2L [10] in terms of quality while achieving an order of magnitude speed-up over them. Our Java Implementation is available at [2]. We also report the accuracies in query answered based on summaries only and show that error is very small and we save a lot of time. Errors reported are normalized by $|V|$. All runtimes are in seconds.

**Dataset Description and Statistics:** We perform experiments on unweighted real-world graphs of varying sizes. We treat all the datasets as undirected. The order of the graphs ranges from a few thousand to a few million. The brief statistics of the graphs used in experiments are given in Table 7.1.

[1]http://snap.stanford.edu

[2]`https://bitbucket.org/M_AnwarBeg/scalablesumm/`

| **Name** | **Number of Nodes** | **Number of Edges** | **Network Type** |
|---|---|---|---|
| Facebook | $4,039$ | $88,234$ | social network |
| Email | $36,692$ | $183,831$ | social network |
| Stanford | $281,903$ | $1,992,636$ | web network |
| Amazon | $403,394$ | $2,443,408$ | co-purchasing network |
| Youtube | $1,157,828$ | $2,987,624$ | social network |
| Skitter | $1,696,415$ | $11,095,298$ | internet topology |
| Wiki-Talk | $2,394,385$ | $4,659,565$ | social network |

Table 7.1: Brief statistics and type of datasets used for experimental evaluation.

- Facebook[1]: The dataset is an anonymized subgraph of Facebook, where a node is a user and an edge shows the friendship between the respective nodes.
- Email[1]: This is an email network, where a node represents an email address and an exchange of a email between two email addresses constitutes an edge.
- Stanford[1]: The dataset consists of webpages from Stanford university and a hyperlink is represented as an edge in the dataset.
- Amazon[1]: Amazon is a co-purchasing graph of products, where a product is represented by a node and there is an edge between two nodes if the two products are bought together.
- Youtube[1]: Youtube graph is the largest connected component of Youtube friendship network. In the graph, a node represents a user and an edge between two users shows the friendship between them.
- Skitter[1]: It is an internet topology network where nodes correspond to autonomous systems and communication between them constitutes edges.
- Wiki-Talk[1]: The dataset shows the Wikipedia Talk network in which a node is a user and an editing activity by a user on the page of another user makes an edge between the two respective users.

**Impact of Sample Size on RE and Runtime:** We show the impact of sample size on RE and the time taken to compute the summary. We evaluate our approach with $s \in \{\log n(t), 5\log n(t),$

$\sqrt{n(t)}, \log^2 n(t)\}$ and $w = 200$ to show the impact of $s$ on the quality of the summary and runtime. In Figure 7.4, it is evident that the error decreases with the increase in $s$, however, the benefit in quality is not proportional to the increase in computational cost. We show the results for Email, Facebook and Stanford datasets; other datasets show the same trend regarding the quality of summary and computational time with varying $s$. 1470

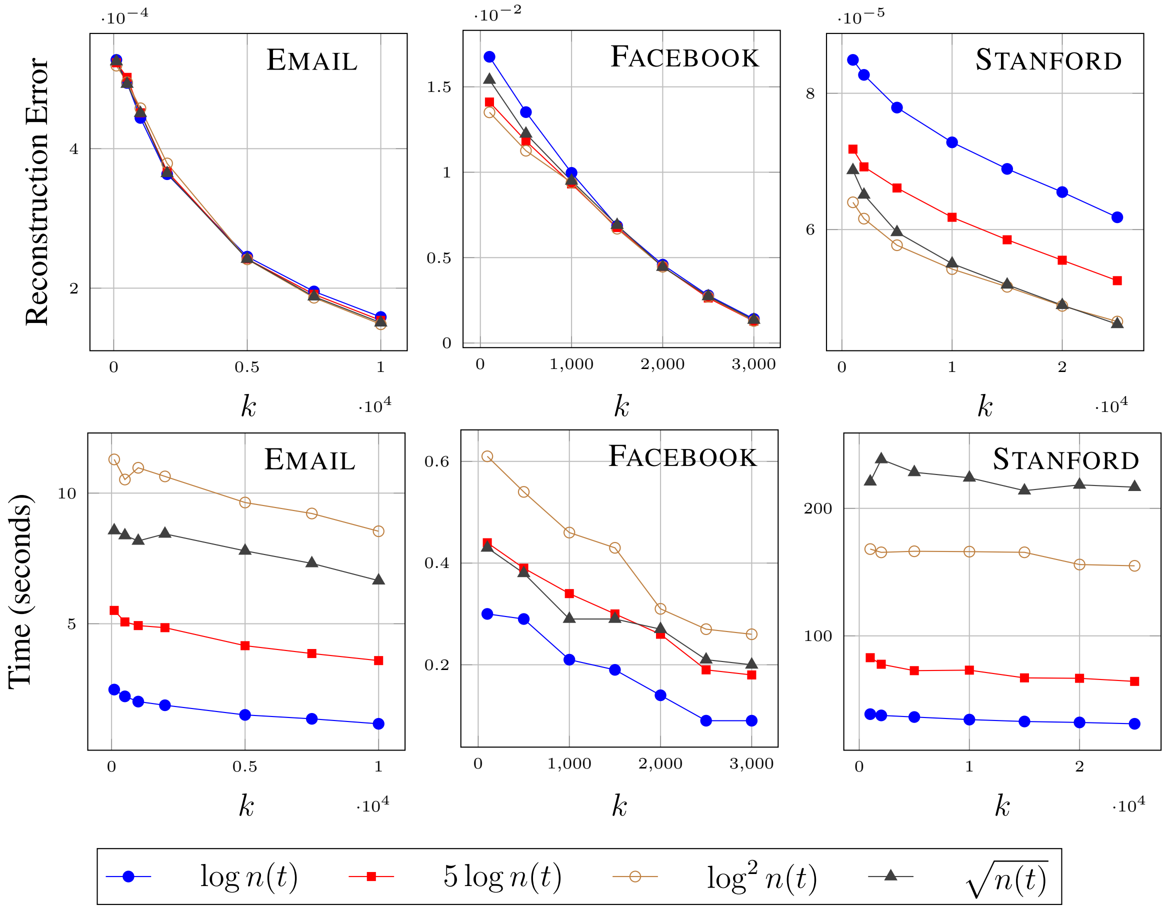


Figure 7.4: Impact of sample size on RE (left column) and runtime (right column). We take sample size $s \in \{\log n(t), 5\log n(t), \log^2 n(t), \sqrt{n(t)}\}$, where $n(t)$ is the number of supernodes in the summary. Increasing $s$ results in significant increase in runtime, however, there is relatively less reduction in RE.

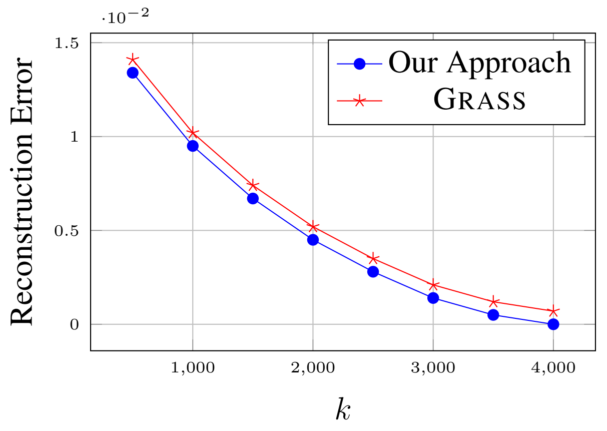


Figure 7.5: Comparison of our approach and GRASS based on RE for varying summary sizes $k$ on Facebook dataset. Note that to build a summary with $k = 500$, our method and GRASS took $0.33$ and $11.16$ seconds, respectively.

**Reconstruction Error and Runtime Comparison:** We compare the reconstruction error (RE) and the runtime to compute summaries by our approach and the competitor methods, GRASS and S2L. We normalize RE by the number of entries in the expected adjacency matrix $\bar{A}$ and show the comparison of normalized RE with GRASS and S2L in Figure 7.5 and Table 7.2, respectively. Note that GRASS only works for graphs with a few thousand nodes, so we show the comparison with GRASS on a small graph only. From the results, it is clear that our method computes the summary in much less time and RE of the summary produced by our solution remains comparable with that of the known methods. Similarly, the quality of the summary also improves by increasing the width of the count-min sketch (increase in approximation budget). To show the approximation quality of score computation, we also report results for exact score computation. Results show that, especially for large graphs, the count-min sketch closely approximates the exact score of a pair for merging.

| | | | Reconstruction Error | | Time Taken(seconds) | |
|---|---|---|---|---|---|---|
| Graph | $k$ | $w$ | Our Approach | S2L | Our Approach | S2L |
| Facebook | 100 | 50 | 2.07E−2 | 1.06E−2 | 0.34 | 1.45 |
| | | 100 | 1.99E−2 | | 0.25 | |
| | | 200 | 1.65E−2 | | 0.19 | |
| | | □ | 1.62E−2 | | 0.32 | |
| | 500 | 50 | 1.92E−2 | 8.61E−3 | 0.33 | 4.68 |
| | | 100 | 1.80E−2 | | 0.24 | |
| | | 200 | 1.35E−2 | | 0.22 | |
| | | □ | 1.32E−2 | | 0.37 | |
| Email | 100 | 50 | 5.39E−4 | 5.00E−4 | 1.91 | 45.94 |
| | | 100 | 5.35E−4 | | 1.84 | |
| | | 200 | 5.28E−4 | | 1.96 | |
| | | □ | 5.26E−4 | | 2.57 | |
| | 500 | 50 | 5.29E−4 | 4.49E−4 | 2.73 | 55.40 |
| | | 100 | 5.26E−4 | | 2.52 | |
| | | 200 | 5.18E−4 | | 2.79 | |
| | | □ | 4.89E−4 | | 4.10 | |
| Stanford | 1000 | 50 | 8.31E−5 | 5.37E−5 | 81.27 | 305.19 |
| | | 100 | 7.54E−5 | | 74.44 | |
| | | 200 | 7.16E−5 | | 89.43 | |
| | | □ | 7.10E−5 | | 108.78 | |
| | 2000 | 50 | 8.17E−5 | 4.65E−5 | 75.74 | 425.95 |
| | | 100 | 7.40E−5 | | 68.89 | |
| | | 200 | 6.91E−5 | | 80.60 | |
| | | □ | 6.87E−5 | | 109.36 | |
| Amazon | 1000 | 50 | 5.98E−5 | 5.91E−5 | 418.52 | 993.00 |
| | | 100 | 5.98E−5 | | 420.91 | |
| | | 200 | 5.98E−5 | | 496.17 | |
| | | □ | 5.97E−5 | | 632.35 | |
| | 2000 | 50 | 5.97E−5 | 5.81E−5 | 385.21 | 1115.78 |
| | | 100 | 5.96E−5 | | 502.39 | |
| | | 200 | 5.96E−5 | | 793.57 | |
| | | □ | 5.95E−5 | | 1096.01 | |

Table 7.2: Comparison of reconstruction error (RE) and runtime (in seconds) on summary graphs, with $k$ supernodes, produced by our approach and the competitor approach, S2L. We take sample size $s = 5 \log n(t)$ and compute the approximate score of a pair nodes for merging using varying approximation budgets (count-min sketch widths) $w \in \{50, 100, 200\}$. It is clear that RE decreases with an increase in $w$ as expected and our estimated score is a close approximation of the exact score ($w = \square$).

**Scalability of Our Approach** To demonstrate the scalability of our approach, we perform experiments on large graphs having more than $1$ million nodes. Table 7.3 contains quality and runtime for large graphs, on which GRASS and S2L are not applicable. We use the same parameter values for $s$ and $w$ as mentioned earlier to generate summaries. We note that increasing the value of $w$ does not improve the quality summaries of large graphs. As on these graphs, none of the competitor methods works, we report results for our approach only.

| | | Skitter $\|E\|$= 1,696,415 $\|V\|$= 11,095,298 | | Wiki-Talk $\|E\|$ = 2,394,385 $\|V\|$ = 4,659,565 | | Youtube $\|E\|$ =1,157,828 $\|V\|$ = 2,987,624 | |
|---|---|---|---|---|---|---|---|
| $k \times (10^3)$ | w | $RE$ | Time(s) | $RE$ | Time(s) | $RE$ | Time(s) |
| 10 | 50 | 2.29E−6 | 521.43 | 1.56E−6 | 311.10 | 3.34E−6 | 207.38 |
| | 100 | 2.23E−6 | 516.03 | 1.51E−6 | 328.19 | 3.22E−6 | 222.22 |
| | 200 | 2.19E−6 | 559.91 | 1.46E−6 | 363.37 | 3.11E−6 | 251.94 |
| | □ | 2.14E−6 | 649.82 | 1.44E−6 | 319.95 | 3.09E−6 | 242.58 |
| 50 | 50 | 2.11E−6 | 481.40 | 1.23E−6 | 285.89 | 2.50E−6 | 184.67 |
| | 100 | 1.96E−6 | 480.85 | 1.21E−6 | 299.98 | 2.38E−6 | 195.85 |
| | 200 | 1.88E−6 | 524.94 | 1.20E−6 | 329.39 | 2.37E−6 | 215.87 |
| | □ | 1.97E−6 | 591.35 | 1.20E−6 | 273.24 | 2.36E−6 | 199.48 |
| 100 | 50 | 1.91E−6 | 436.84 | 1.09E−6 | 266.15 | 1.97E−6 | 160.11 |
| | 100 | 1.70E−6 | 445.27 | 1.09E−6 | 276.32 | 1.94E−6 | 167.67 |
| | 200 | 1.66E−6 | 486.88 | 1.09E−6 | 303.44 | 1.94E−6 | 183.64 |
| | □ | 1.66E−6 | 535.02 | 1.09E−6 | 248.73 | 1.94E−6 | 164.80 |
| 250 | 50 | 1.38E−6 | 332.27 | 9.07E−7 | 223.65 | 1.30E−6 | 103.25 |
| | 100 | 1.24E−6 | 350.47 | 9.05E−7 | 232.39 | 1.30E−6 | 107.79 |
| | 200 | 1.23E−6 | 376.89 | 9.03E−7 | 256.05 | 1.30E−6 | 118.73 |
| | □ | 1.23E−6 | 392.58 | 9.03E−7 | 203.93 | 1.30E−6 | 98.70 |

Table 7.3: We report the reconstruction error (RE) and the time (in seconds) to compute the summaries using our approach on large graphs. S2L does scale to these large graphs. On these large graphs, we compute summaries of relatively larger sizes with approximate score computation using count-min sketch width $w \in \{50, 100, 200\}$ and using exact score computation ($w = \square$).

**Accuracy in Query Answers:** Recall that a measure to evaluate the quality of $S$ is the accuracy in answers to queries based on $S$ only. We run different structural queries including degree, eigenvector-centrality of a node, and triangle density of the graph to quantify the goodness of our summary. In Table 7.4, we report the average error along with standard deviation for the degree query and for triangle density, we show relative triangle density error computed as $\frac{\bar{t}(G)-t(G)}{t(G)}$. As the eigenvector-centrality query is computed based on degree, we show results for the degree query and omit the results for the centrality query. Results are reported with increasing values of $k$, the number of supernodes in summary, and varying width $w$ for the count-min sketch. The results follow the expected trend that the accuracy of query answers improves with the increase in $w$. Similarly, the error in query answers reduces as $k$ increases.

|  |  |  | Avg. Degree Error |  | Triangle Density Error |  |
|---|---|---|---|---|---|---|
| Graph | $k$ | $w$ | Our Approach | S2L | Our Approach | S2L |
| Facebook | 100 | 200 | $16.74 \pm 21$ | $9.89 \pm 12$ | -0.52 | -0.30 |
|  |  | $\square$ | $17.18 \pm 20$ |  | -0.49 |  |
|  | 500 | 200 | $11.99 \pm 15$ | $7.21 \pm 8$ | -0.30 | -0.32 |
|  |  | $\square$ | $11.22 \pm 12$ |  | -0.28 |  |
| Email | 100 | 200 | $6.79 \pm 25$ | $5.70 \pm 16$ | -0.80 | -0.77 |
|  |  | $\square$ | $6.31 \pm 14$ |  | -0.76 |  |
|  | 500 | 200 | $5.71 \pm 19$ | $4.79 \pm 12$ | -0.65 | -0.73 |
|  |  | $\square$ | $5.15 \pm 10$ |  | -0.63 |  |
| Stanford | 1000 | 200 | $7.69 \pm 41$ | $5.11 \pm 36$ | -0.68 | -0.26 |
|  |  | $\square$ | $8.06 \pm 42$ |  | -0.67 |  |
|  | 2000 | 200 | $7.38 \pm 40$ | $4.06 \pm 10$ | -0.64 | -0.23 |
|  |  | $\square$ | $7.60 \pm 38$ |  | -0.63 |  |
| Amazon | 1000 | 200 | $5.63 \pm 13$ | $5.37 \pm 11$ | -1.00 | -0.96 |
|  |  | $\square$ | $5.64 \pm 13$ |  | -1.00 |  |
|  | 2000 | 200 | $5.57 \pm 12$ | $5.13 \pm 9$ | -0.99 | -0.92 |
|  |  | $\square$ | $5.60 \pm 13$ |  | -0.99 |  |

Table 7.4: Accuracy in answers to queries from summaries produced by our approach and S2L. We report the average error along with the standard deviation on the degree query and the error in the relative triangle density of the graphs. $k$ is the number of supernodes in the summary and $w$ is the width of the count-min sketch($w = \square$ shows exact score computation for pairs of nodes).

## 7.5 Conclusion

In this work, we devise a sampling-based efficient approximation algorithm for graph summarization. We derive a closed form for measuring the suitability of a pair of nodes for merging. We approximate this score with theoretical guarantees on error. Another major contribution of this work is the efficient weighted sampling scheme to improve the quality of samples. This enables us to work with substantially smaller sample sizes without compromising summary quality. Our algorithm is scalable to large graphs on which previous algorithms are not applicable. Extensive evaluation on a variety of real-world graphs shows that our algorithm significantly outperforms existing solutions both in quality and time complexity.

# Chapter 8

# Conclusion and Future Directions

In this thesis, we solve two problems of network immunization and graph summarization based on approximation techniques. Firstly, we address the problem of network immunization in which we immunize a small subset of nodes to reduce the network vulnerability against the spread of malicious content. We use the relationship between spectral and graph-theoretic properties of networks and in our series of work, we select a subset of nodes for immunization based on closed walks of lengths $4, 6$ and $8$. We also devise a technique to approximately count the number of closed walks, which makes our approach scalable to large graphs. Experimental evaluation on large real-world networks suggests that our method is a close approximation of the exact solution. Moreover, results for various quality measures like virus spread simulation, reduction in network vulnerability and the run time comparison show that our method performs better than the state-of-the-art solution.

Secondly, we give a sampling-based algorithm for graph summarization. We derive a closed form for measuring the error introduced after merging a pair of nodes. The score also gives the suitability of the pair for merging. We then give an approximate method to compute this score with theoretical guarantees on error. Another major contribution of this work is the efficient weighted sampling scheme to improve the quality of samples. This enables us to work with substantially smaller sample sizes without compromising summary quality. Our algorithm is scalable to large graphs on which previous algorithms are not applicable. Extensive evaluation on a variety of real-world graphs shows that our algorithm significantly outperforms existing solutions both in quality and time complexity.

As future work, we aim for a non-preemptive approach for network immunization in which the information of healthy and infected nodes is available at each time stamp and nodes are immunized based on the available information. Another line of work is the reverse propagation of network immunization in which given a snapshot of an infected graph, the goal is to identify the potential (culprit) nodes from where the infection has started. Some other potential extensions of this work also include i) utilizing specialized graph summarization methods, which will further reduce computational cost as well as improve the immunization performance of the solution ii) extending this work to incorporate dynamic graphs which evolve with time through the addition or deletion of edges iii) exploring non-preemptive graph immunization approaches, where the immunization process starts after the virus attack and the information of infected nodes is available.

In graph summarization, we aim to further reduce the reconstruction error and storage size of the summary graphs. The reconstruction error can be reduced by computing the usefulness of superedges in the summary graph. In this regard, by doing some reverse calculations, we can compute the contribution of retaining and dropping a particular superedge in the summary graph. This step during the construction of the summary can help further reduce the reconstruction error. Another related extension is the sparsification of the summary graph in which superedges with very small weights are dropped from the summary graph, which helps in reducing the storage size of the summary graph. Another related research area is of representing nodes in a low-dimensional feature space and do summarization by making clusters of close-by/similar nodes.

# Bibliography

[1] R. Angles and C. Gutierrez, *An Introduction to Graph Data Management*, pp. 1–32. Cham: Springer International Publishing, 2018.

[2] T. Suel and J. Yuan, “Compressing the graph structure of the web,” in *Data Compression Conference, DCC*, pp. 213–222, 2001.

[3] M. Adler and M. Mitzenmacher, “Towards compressing web graphs,” in *Data Compression Conference, DCC*, pp. 203–212, 2001.

[4] C. Song, W. Hsu, and M. L. Lee, “Node immunization over infectious period,” in *ACM International Conference on Information and Knowledge Management, CIKM*, pp. 831–840, 2015.

[5] C. Chen, H. Tong, B. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. Chau, “Node immunization on large graphs: Theory and algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 113–126, 2016.

[6] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, “Epidemic spreading in real networks: An eigenvalue viewpoint,” in *Symposium on Reliable Distributed Systems, SRDS*, pp. 25–34, 2003.

[7] K. LeFevre and E. Terzi, “GraSS: Graph Structure Summarization,” in *International Conference on Data Mining, SDM*, pp. 454–465, 2010.

[8] K. Khan, W. Nawaz, and Y. Lee, “Set-based approximate approach for lossless graph summarization,” *Computing*, vol. 97, no. 12, pp. 1185–1207, 2015.

[9] G. Strang, *Linear Algebra and its Applications*. Academic Press, 1988.

[10] M. Riondato, D. García-Soriano, and F. Bonchi, "Graph Summarization with Quality Guarantees," in *International Conference on Data Mining, ICDM*, pp. 947–952, 2014.

[11] P. Shakarian, A. Bhatnagar, A. Aleali, E. Shaabani, and R. Guo, *The Independent Cascade and Linear Threshold Models*, pp. 35–48. Springer International Publishing, 2015.

[12] S. Goel, A. Anderson, J. M. Hofman, and D. J. Watts, "The structural virality of online diffusion," *Management Science*, vol. 62, pp. 180–196, 2016.

[13] V. Blavatska and Y. Holovatch, "Spreading processes in post-epidemic environments," *Physica A: Statistical Mechanics and its Applications*, vol. 573, p. 125980, 2021.

[14] A. Ganesh, L. Massoulié, and D. Towsley, "The effect of network topology on the spread of epidemics," in *IEEE Annual Joint Conference of the Computer and Communications Societies, INFOCOM*, pp. 1455–1466, 2005.

[15] P. Van Mieghem, "Exact markovian Sir and Sis epidemics on networks and an upper bound for the epidemic threshold," *arXiv preprint arXiv:1402.1731*, 2014.

[16] P. Van Mieghem, F. D. Sahnehz, and C. Scoglioz, "An upper bound for the epidemic threshold in exact markovian sir and sis epidemics on networks," in *IEEE Conference on Decision and Control, CDC*, pp. 6228–6233, 2014.

[17] H. J. Ahn and B. Hassibi, "Global dynamics of epidemic spread over complex networks," in *IEEE Conference on Decision and Control, CDC*, pp. 4579–4585, 2013.

[18] A. Khanafer, T. Basar, and B. Gharesifard, "Stability properties of infected networks with low curing rates," in *American Control Conference, ACC*, pp. 3579–3584, 2014.

[19] A. Khanafer, T. Başar, and B. Gharesifard, "Stability properties of infection diffusion dynamics over directed networks," in *IEEE Conference on Decision and Control, CDC*, pp. 6215–6220, 2014.

[20] P. Van Mieghem, D. Stevanović, F. Kuipers, C. Li, R. Van De Bovenkamp, D. Liu, and H. Wang, "Decreasing the spectral radius of a graph by link removals," *Physical Review E*, vol. 84, no. 1, p. 016101, 2011.

[21] M. Ahmad, J. Tariq, M. Shabbir, and I. Khan, "Spectral methods for immunization of large networks," *Australasian Journal of Information Systems*, vol. 21, 2017.

[22] J. Tariq, M. Ahmad, I. Khan, and M. Shabbir, "Scalable approximation algorithm for network immunization," in *Pacific Asia Conference on Information Systems, PACIS*, pp. 1–12, 2017.

[23] M. Ahmad, J. Tariq, M. Farhan, M. Shabbir, and I. Khan, "Who should receive the vaccine?," in *Australasian Data Mining Conference, AusDM*, pp. 137–145, 2016.

[24] S. Abbas, J. Tariq, A. Zaman, and I. Khan, "Sampling based efficient algorithm to estimate the spectral radius of large graphs," in *IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW*, pp. 175–180, 2017.

[25] M. Riondato, D. García-Soriano, and F. Bonchi, "Graph summarization with quality guarantees," *Data Mining and Knowledge Discovery*, vol. 31, no. 2, pp. 314–349, 2017.

[26] M. A. Beg, M. Ahmad, A. Zaman, and I. Khan, "Scalable approximation algorithm for graph summarization," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD*, pp. 502–514, 2018.

[27] G. Zhang, H. Rong, F. Neri, and M. J. Pérez-Jiménez, "An optimization spiking neural P system for approximately solving combinatorial optimization problems," *International Journal of Neural Systems*, vol. 24, no. 5, pp. 1440006:1–1440006:16, 2014.

[28] L. Pan, G. Puaun, G. Zhang, and F. Neri, "Spiking neural *P* systems with communication on request," *International Journal of Neural Systems*, vol. 27, no. 8, pp. 1750042:1–1750042:13, 2017.

[29] T. Wu, F. Bîlbîe, A. Paun, L. Pan, and F. Neri, “Simplified and yet turing universal spiking neural P systems with communication on request,” *International Journal of Neural Systems*, vol. 28, no. 8, pp. 1850013:1–1850013:19, 2018.

[30] C. J. Kuhlman, G. Tuli, S. Swarup, M. V. Marathe, and S. Ravi, “Blocking simple and complex contagion by edge removal,” in *IEEE International Conference on Data Mining, ICDM*, pp. 399–408, 2013.

[31] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, “Gelling, and melting, large graphs by edge manipulation,” in *ACM International Conference on Information and Knowledge Management, CIKM*, pp. 245–254, 2012.

[32] Y. Zhang and B. A. Prakash, “Dava: Distributing vaccines over networks under prior information,” in *SIAM International Conference on Data Mining, SDM*, pp. 46–54, 2014.

[33] Y. Zhang and B. A. Prakash, “Scalable vaccine distribution in large graphs given uncertain data,” in *ACM International Conference on Conference on Information and Knowledge Management, CIKM*, pp. 1719–1728, 2014.

[34] B. A. Prakash, J. Vreeken, and C. Faloutsos, “Spotting culprits in epidemics: How many and which ones?,” in *IEEE International Conference on Data Mining, ICDM*, pp. 11–20, 2012.

[35] D. Bienstock and P. Seymour, “Monotonicity in graph searching,” *Journal of Algorithms*, vol. 12, no. 2, pp. 239–245, 1991.

[36] P. Flocchini, M. J. Huang, and F. L. Luccio, “Decontamination of hypercubes by mobile agents,” *Networks*, vol. 52, no. 3, pp. 167–178, 2008.

[37] P. Flocchini, M. J. Huang, and F. Luccio, “Decontaminating chordal rings and tori using mobile agents,” *International Journal of Foundations of Computer Science*, vol. 18, no. 03, pp. 547–563, 2007.

[38] P. Fraigniaud and N. Nisse, “Monotony properties of connected visible graph searching,” *Information and Computation*, vol. 206, no. 12, pp. 1383–1393, 2008.

[39] Y. Daadaa, A. Jamshed, and M. Shabbir, "Network decontamination with a single agent," *Graphs and Combinatorics*, vol. 32, no. 2, pp. 559–581, 2016.

[40] F. Morone and H. A. Makse, "Influence maximization in complex networks through optimal percolation," *Nature*, vol. 524, no. 7563, p. 65, 2015.

[41] D. Erdös, V. Ishakian, A. Lapets, E. Terzi, and A. Bestavros, "The filter-placement problem and its application to minimizing information multiplicity," *PVLDB*, vol. 5, no. 5, pp. 418–429, 2012.

[42] A. Arulselvan, C. W. Commander, P. M. Pardalos, and O. Shylo, "Managing network risk via critical node identification," *Risk Management in Telecommunication Networks, Springer*, 2007.

[43] J. Li, P. M. Pardalos, B. Xin, and J. Chen, "The bi-objective critical node detection problem with minimum pairwise connectivity and cost: theory and algorithms," *Soft Computing*, pp. 1–16, 2019.

[44] M. Ventresca and D. Aleman, "Efficiently identifying critical nodes in large complex networks," *Computational Social Networks*, vol. 2, no. 1, p. 6, 2015.

[45] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," in *International Conference on World Wide Web, WWW*, pp. 595–602, 2004.

[46] J. You, Q. Pan, W. Shi, Z. Zhang, and J. Hu, "Towards graph summary and aggregation: A survey," in *Social Media Retrieval and Mining*, Springer, 2013.

[47] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *International Conference on Management of Data, SIGMOD*, pp. 419–432, 2008.

[48] M. Beg, M. Ahmad, A. Zaman, and I. Khan, "Scalable approximation algorithm for graph summarization," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD*, pp. 502–514, 2018.

[49] S. Ali, M. Shakeel, I. Khan, S. Faizullah, and M. Khan, "Predicting attributes of nodes using network structure," *ACM Transactions on Intelligent Systems and Technology*, vol. 12, no. 2, pp. 1–23, 2021.

[50] K. Khan, W. Nawaz, and Y. Lee, "Set-based unified approach for summarization of a multi-attributed graph," *World Wide Web*, vol. 20, no. 3, pp. 543–570, 2017.

[51] Z. Liu, J. X. Yu, and H. Cheng, "Approximate homogeneous graph summarization," *Journal of Information Processing*, vol. 20, no. 1, pp. 77–88, 2012.

[52] Y. Tian, R. Hankins, and J. Patel, "Efficient aggregation for graph summarization," in *International Conference on Management of Data, SIGMOD*, pp. 567–580, 2008.

[53] A. Khan, L. Golab, M. Kargar, J. Szlichta, and M. Zihayat, "Compact group discovery in attributed graphs and social networks," *Information Processing & Management*, vol. 57, no. 2, p. 102054, 2020.

[54] K. Khan, B. Dolgorsuren, N. Tu, W. Nawaz, and Y. Lee, "Faster compression methods for a weighted graph using locality sensitive hashing," *Information Sciences*, vol. 421, pp. 237–253, 2017.

[55] F. Zhou, Q. Qu, and H. Toivonen, "Summarisation of weighted networks," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 29, no. 5, pp. 1023–1052, 2017.

[56] L. Shi, H. Tong, J. Tang, and C. Lin, "VEGAS: visual influence graph summarization on citation networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 12, pp. 3417–3431, 2015.

[57] I. Tsalouchidou, G. Morales, F. Bonchi, and R. Baeza-Yates, "Scalable dynamic graph summarization," in *International Conference on Big Data, BigData*, pp. 1032–1039, 2016.

[58] J. Ko, Y. Kook, and K. Shin, "Incremental lossless graph summarization," in *International Conference on Knowledge Discovery and Data Mining, SIGKDD*, pp. 317–327, 2020.

[59] Q. Qu, S. Liu, F. Zhu, and C. Jensen, "Efficient online summarization of large-scale dynamic networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3231–3245, 2016.

[60] P. Boldi, M. Santini, and S. Vigna, "Permuting web and social graphs," *Internet Mathematics*, vol. 6, no. 3, pp. 257–283, 2009.

[61] K. Shin, A. Ghoting, M. Kim, and H. Raghavan, "Sweg: Lossless and lossy summarization of web-scale graphs," in *International Conference on World Wide Web, WWW*, pp. 1679—-1690, 2019.

[62] N. Tang, Q. Chen, and P. Mitra, "Graph stream summarization: From big bang to big crunch," in *International Conference on Management of Data, SIGMOD*, pp. 1481–1496, 2016.

[63] A. Khan and C. Aggarwal, "Toward query-friendly compression of rapid graph streams," *Social Network Analysis and Mining*, vol. 7, no. 1, pp. 23:1–23:19, 2017.

[64] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "VOG: summarizing and understanding large graphs," in *International Conference on Data Mining, SDM*, pp. 91–99, 2014.

[65] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3077–3089, 2014.

[66] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys*, vol. 51, no. 3, pp. 62:1–62:34, 2018.

[67] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos, "Epidemic thresholds in real networks," *ACM Transactions on Information and System Security*, vol. 10, no. 4, pp. 1:1–1:26, 2008.

[68] D. Serre, *Matrices*. Springer, 2002.

[69] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, “An analysis of approximations for maximizing submodular set functions,” *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[70] D. West, *Introduction to graph theory*. Prentice Hall, 2001.

[71] F. Neri, *Linear Algebra for Computational Sciences and Engineering*. Springer, 2019.

[72] E. Kreyszig, *Introductory functional analysis with applications*. Wiley, 1978.

[73] J. Bondy and U. Murty, *Graph theory*. Springer, 2008.

[74] C. R. R. Thomas H.. Cormen, Leiserson and C. Stein, *Introduction to algorithms*, vol. 6. MIT press Cambridge, 2001.

[75] C. Song, K. Shu, and B. Wu, “Temporally evolving graph neural network for fake news detection,” *Information Processing & Management*, vol. 58, no. 6, pp. 102712:1–102712:18, 2021.

[76] C. Chen, F. Cai, X. Hu, W. Chen, and H. Chen, “HHGN: A hierarchical reasoning-based heterogeneous graph neural network for fact verification,” *Information Processing & Management*, vol. 58, no. 5, pp. 102659:1–102659:14, 2021.

[77] C. Aggarwal and P. Yu, “A condensation approach to privacy preserving data mining,” in *International Conference on Extending Database Technology, EDBT*, pp. 183–199, 2004.

[78] L. Sweeney, “k-Anonymity: A Model for Protecting Privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[79] P. Samarati, “Protecting Respondents’ Identities in Microdata Release,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[80] M. Hay, G. Miklau, D. Jensen, D. Towsley, and C. Li, “Resisting structural re-identification in anonymized social networks,” *Journal on Very Large Data Bases*, vol. 19, no. 6, pp. 797–823, 2010.

[81] B. Prakash, J. Vreeken, and C. Faloutsos, "Spotting culprits in epidemics: How many and which ones?," in *International Conference on Data Mining, ICDM*, pp. 11–20, 2012.

[82] M. Ahmad, J. Tariq, M. Shabbir, and I. Khan, "Spectral methods for immunization of large networks," *Australasian Journal of Information Systems*, vol. 21, pp. 1–18, 2017.

[83] M. Ahmad, S. Ali, J. Tariq, I. Khan, M. Shabbir, and A. Zaman, "Combinatorial trace method for network immunization," *Information Sciences*, vol. 519, pp. 215–228, 2020.

[84] Z. Hassan, M. Shabbir, I. Khan, and W. Abbas, "Estimating descriptors for large graphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD*, pp. 779–791, 2020.

[85] A. Ahmed, Z. R. Hassan, and M. Shabbir, "Interpretable multi-scale graph descriptors via structural compression," *Information Sciences*, vol. 533, pp. 169–180, 2020.

[86] J. Zhang, S. S. Bhowmick, H. H. Nguyen, B. Choi, and F. Zhu, "Davinci: Data-driven visual interface construction for subgraph search in graph databases," in *International Conference on Data Engineering, ICDE*, pp. 1500–1503, 2015.

[87] Z. Shen, K. Ma, and T. Eliassi-Rad, "Visual Analysis of Large Heterogeneous Social Networks by Semantic and Structural Abstraction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1427–1439, 2006.

[88] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[89] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[90] C. Wong and M. Easton, "An efficient method for weighted sampling without replacement," *SIAM Journal on Computing*, vol. 9, no. 1, pp. 111–113, 1980.