MANAGING INFORMATION DIFFUSION IN ONLINE SOCIAL NETWORKS VIA STRUCTURAL ANALYSIS

CHONGGANG SONG
Bachelor of Engineering
Nankai University, China

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2017

Supervisor:
Professor Wynne Hsu

Examiners:
Professor Tan Kian Lee
Associate Professor Stephane Bressan
Associate Professor Liu Qi, University of Science and Technology of China

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Chonggang Song
2017

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors, Professor Wynne Hsu and Professor Mong Li Lee for their continuous support and valuable guidance. They have advised me with their immense knowledge, given me freedom to choose my own topic of interest, shown infinite patience when I had no research experience and made time for me whenever I asked for a discussion. I could never have imagined any better advisor and mentor for my doctoral study.

Besides my advisors, I would like to thank my thesis committee, Professor Kian-Lee Tan and Professor Stephane Bressan for their insightful comments and constructive feedback that motivated me to refine my works.

I would like to thank my fellow labmates Dr. Zhao Gang, Dr. Li Furong, Dr. Chen Wei, Dr. Zeng Zhong and Dr. Xu Enliang for the stimulating discussions, for the warm encouragements when I met with setbacks and for all the fun we had over the past few years. Thank you for infusing my Ph.D study with colours and joy. I would also like to thank my roommate Mr. Huang Zhuqing for persistently helping me when I was frustrated and selflessly sharing his life experience when I feel at sea. Friendships with them will be one of the most valuable things obtained during my Ph.D years.

Last but not the least, I would like to thank my parents for supporting me unconditionally over the past 27 years. I could never have completed my Ph.D degree without their love and understanding. They have been and will always be my utmost aspiration in life.

Abstract

With the advent of Web 2.0, social networks like Facebook and Twitter offer a platform for millions of users to share information, accelerating the diffusion of information among web users. Many studies have been conducted to understand as well as control the information diffusion process. This thesis seeks to address the challenge of developing methods for expanding as well as preventing information diffusion in highly dynamic and time-sensitive online social networks. We first identify users that are can boost the influence of some information to a specific region or across different communities in social networks. We then look at the users that can help us block the diffusion of misinformation via immunization or starting truth campaign from them.

To maximize the influence of good information, we first consider the task of maximizing the number of participants in an event via propagating the event information in a location-based social network. Consider an event that is taking place at a specific position after a few days, we wish to propagate the event information to the users that are located near the event venue before the event taking place. This task is an extension of the tradition influence maximization that asks for a set of k nodes in a given graph G, such that it can reach the largest expected number of remaining nodes in G. Existing methods have either considered that the influence be targeted to meet certain deadline constraint, or be restricted to specific geographical region. However, considering the location and deadline independently may lead to a less than optimal set of users. In this thesis, we formalize the problem of targeted influence maximization problem in social networks. We adopt a login model where each user is associated with a login probability and he can be influenced by his neighbors only when he is online. We develop a sampling based algorithm that returns a $(1-1/e-\varepsilon)$- approximate solution. Experiments on real-world social network datasets demonstrate the effectiveness and efficiency of our proposed method.

We then examine the users that are located between remote clusters in social networks. The theory of brokerage in sociology suggests if contacts between two parties are enabled through a third party, the latter occupies a strategic position of controlling information flows. Such individuals are called *brokers* and they play a key role in disseminating information. However, there is no systematic approach to identify brokers in online social networks. We formally define the

problem of detecting top-k brokers given a social network and show that it is NP-hard. We develop a heuristic algorithm to find these brokers based on the weak tie theory. In order to handle the dynamic nature of online social networks, we design incremental algorithms: WeakTie-Local for unidirectional networks and WeakTie-Bi for bidirectional networks. We use two real world datasets, DBLP and Twitter, to evaluate the proposed methods. We also demonstrate how the detected brokers are useful in diffusing information across communities and propagating tweets to reach more distinct users.

To minimize the influence of misinformation, we first consider locating nodes to immunize in computer/social networks to control the spread of virus or rumors. In real-world contagions, nodes may get infected by external sources when the propagation is underway. While most studies formalize the problem in a setting where contagion starts at one time point, we model a more realistic situation where there are likely to be many breakouts of contagions over a time window. We call this the *node immunization over infectious period* (NIIP) problem. We show that the NIIP problem is NP-hard and remains so even in directed acyclic graphs. We propose a NIIP algorithm to select k nodes to immunize over a time period. Simulation is performed to estimate a good distribution of k over the time period. For each time point, the NIIP algorithm will make decisions which nodes to immunize given the estimated value of k for that time point. Experiments show that the proposed NIIP algorithm outperform the state-of-the-art algorithms in terms of both effectiveness and efficiency.

Furthermore, we realize it is more effective to identify nodes to start a truth campaign to block the spreading of rumor. Existing works on limiting misinformation propagation do not take into account the delays of information diffusion or the time point beyond which propagation of misinformation is no longer critical. In this paper, we consider a more realistic situation where information is propagated with delays and the goal is to reduce the number of rumor-infected users before a deadline. We call this the *Temporal Influence Blocking* (TIB) problem, and propose a two-phase solution called TIB-Solver to select k nodes to start a truth campaign such that the number of users reached by a rumor is minimized. Experiments show that the proposed TIB-Solver outperforms the state-of-the-art algorithms in terms of both effectiveness and efficiency.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With the rapid development of online social networks, the way people receive and spread information has been dramatically changed. Web users post timelines on Twitter[1], share photoes on Instagram[2] and join events posted on Facebook[3]. All these social network services allow users to view, reply to as well as share others' posts, enabling popular information to be viewed and shared by an enormous number of users [BRMA12, LXC+14].

Given the impact of online social networks in disseminating information, many researchers have conducted extensive studies to understand the propagation of information. The diffusion of information over social networks is the process that some news, innovations or advertisements starting from some users on a social platform propagates through the network via social behaviors like sharing, liking and messaging. Information diffusion has attracted many research interests such as modeling the propagation process, predicting influential users and preventing the spread of misinformation.

Researchers often model social networks as a graph with nodes and edges. Each node represents an individual user while the edges between any pair of users stand for the relationships (e.g. friendship, kinship, trade relations etc.). An edge from user u to v represents that user u follows user v, and we say u is a *follower* of v and v is a *followee* of u. There have been a variety of techniques to capture the diffusion process with the graph representaion of social networks [KKT03, AH11, WWX12]. The most prevalent model that has been studied is the *Independent Cascade* (IC) model [KKT03]. We will speak of each individual node as being *active* (influenced by some information) or *inactive*. In the IC model, we start with an initial set of active nodes and the diffusion process unfolds in discrete time steps according to the following randomized rule: when a node v first becomes active at time step t, it is given a single chance to activate each currently inactive neighbor w; it succeeds with the influence probability $p(v, w)$ independently of the history thus far. If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order. If v succeeds, then w will

[1] http://www.twitter.com/

[2] https://instagram.com/

[3] http://www.facebook.com/

become active in step $t+1$, but, whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds. The process runs until no more activations are possible.

With the IC model, we are able to simulate the propagation of some information and measure the influence of it by counting the number of users reached. Many techniques have been developed to manage the diffusion of information given different incentives. On one hand, we want to maximize the influence of good information such as news, innovations as well as marketing advertisements. On the other hand, we wish to minimize the influence of misinformation such as malicious rumors. In the following section, we introduce the main research challenges in managing information diffusion from those two aspects.

1.1 Research Challenges

In this section, we elaborate the research challenges in maximizating the influence for good information and minimizing the influence of misinformation.

1.1.1 Maximizing the Influence of Good Information

Influence maximization (IM) [DR01, KKT03] is a fundamental data mining problem that finds k nodes in a given network G whose adoptions of some ideas, opinions or innovations can trigger the largest expected number of adoptions by the other nodes. This problem has been extensively studied and applied to web applications such as viral marketing. Kemp et al. [KKT03] proved that the basic influence maximation problem is NP-hard and provided a $(1-1/e-\varepsilon)$-approximate solution by greedily selecting k nodes where each node maximizes the marginal gain of influence spread. Since then, many methods have been developed to improve the efficiency while maintaining the quality of returned nodes [BBCL14, CWW10, DSGZ13, LKG+07, TSX15, TXS14, YMPH16]. In the following, we discuss two unhandled challenges in broadcasting information.

Challenge One: Considering Temporal and Geographical Information. Recently, researchers have realized that in real life, most influence maximization applications have associated deadlines [CLZ12, CDPW14, LCXZ12]. For example, a department store plans to have a Christmas sale from 21st Dec to 25th Dec. If the news of this event reaches a user after 25th Dec, then this information has zero value to the user since the sale is already over. As such, the authors in [CLZ12, CDPW14, LCXZ12] take into consideration the deadlines, as well as possible delays in the information diffusion process, and propose heuristic algorithms to solve this deadline-aware influence maximization problem.

Furthermore, [LCF+14] showed that location is also important in influence maximization tasks and aimed to find k users who can maximize the number of influenced users within a

specific geographical region. They extend the heuristic MIA algorithm introduced in [CWW10] by counting only the users in the target region when computing the influence spread of each node. [ZCL+15] defined a location-based influence maximization model to take into account the probability that online influence will translate to sales in a physical shop. This requires altering the target of the influence maximization to find users whose location preferences are close to the physical shop.

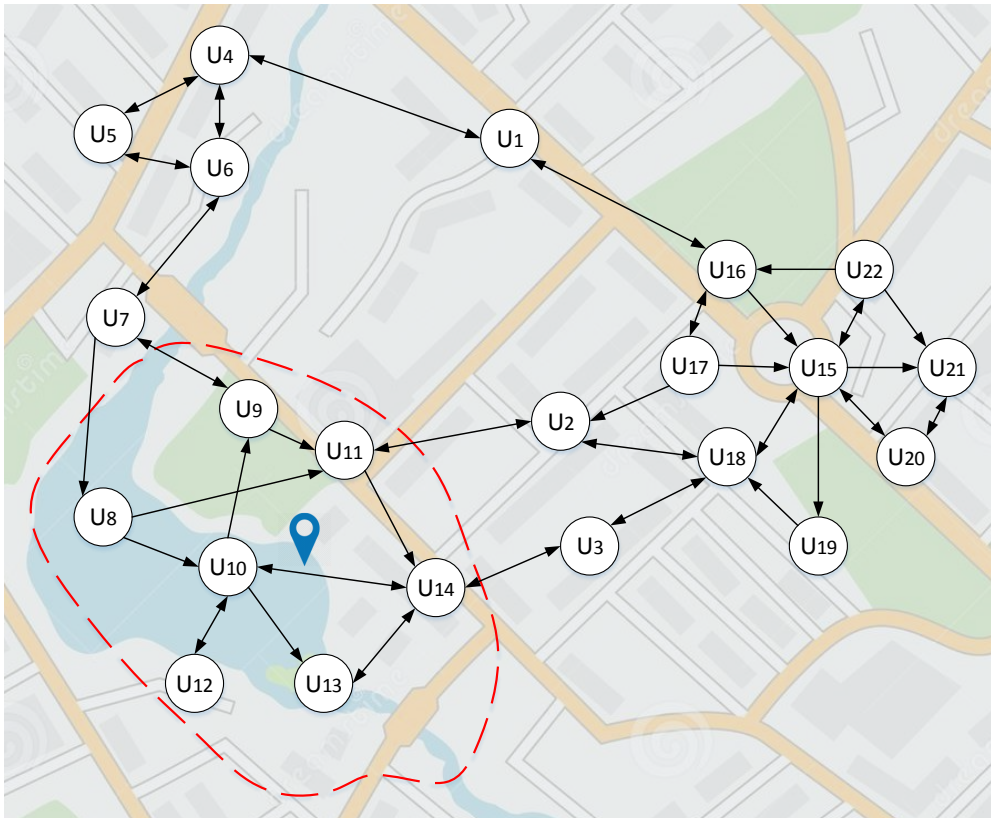


Figure 1.1: Example Location-Based Social Network

Figure 1.1 shows the locations of users on a social network where the edges denote the direction of influence. Suppose an event organizer is hosting an upcoming event at a location indicated by the blue pin, and he has the budget to broadcast the event information to only one user in the social network. To simplify discussion, we assume that if two users u and v are connected by an edge, then it takes one time point for u to influence v. For example, it will take two time points for user u_1 to influence u_5 (via u_4). Further, only users in the red circled region are willing to travel to the event location. Then location-based influence maximization methods will select u_7 as he can reach the largest number of users in the red circled region. However, if we impose a deadline of 2 time points, then u_1 can only influence 4 nodes in the region. On the other hand, time-critical influence maximization solutions will select u_{15} as he can influence 9 nodes in 2 time points as u_{15} is located at the center of a cluster of nodes. Note that none of the influenced nodes are in the red circled region. Both approaches have missed the optimal user u_8 who is able to influence 5 other nodes in

the targeted region within 2 time points.

This example shows the need to consider both time constraint and geographical region when solving a *targeted influence maximization* problem. We consider a user is *influenced* if his neighbors have propagated information of an event to him. A user is *registered* if he is *influenced* and has decided to participate in the event. Unlike traditional IM problem trying to *influence* the largest number of users online, the *targeted influence maximization problem* aims to identify k users that result in most *registered* participants.

Challenge Two: Influencing Remote Users. In social theory, transmitting actors that enable the contacts between other parties are referred to as *brokers*. Sociologists have long recognized that such users, situated between otherwise disconnected or remote users, possess advantageous positions [Bur07]. On one hand, such users serve as bridges and enable the interaction between their neighbors; on the other hand, they can access remote information sources and control the information flow between them.

Consider the example social network shown in Figure 1.2. This example uses the same network as in Figure 1.1 without the underneath map as there is no location constraint. PageRank based algorithms will select u_{15} because it is incident with other high-degree nodes. Methods that utilize betweenness measure will find u_2 since it carries the shortest paths between the two clusters of nodes. However, we note that the removal of u_2 or u_7 does not have much impact on their neighbors as the neighbours can still reach other users without significant increase in the distance.

On the other hand, the situation is different for u_1. We see that u_1 enables u_4, u_5 and u_6 to reach u_{15} and its neighbours within 3 or 4 hops. However, when u_1 is removed, u_4, u_5 and u_6 need to travel a very long distance to reach u_7 and its neighbors. We say u_1 is the broker that brings remote nodes close to others. With the online social networks playing an increasingly significant role in spreading news and opinions, identifying such brokers is clearly advantageous. Information disseminated through these users have a much higher chance of reaching distant users whereas the centrality-based nodes may not reach those isolated individuals.

This example shows the role played by brokers in enabling the diffusion of information between remote clusters of users. Unlike traditional centrality-based measures that find nodes seated within a dense cluster, we aim to identify the *brokers* who largely reduce the pair-wise distance between other nodes in the network.

1.1.2 Minimizing the Influence of Misinformation

With the advent of the Internet, everyone is connected. This enables the data generated by any user to be easily accessed by other users all over the world. Despite the convenience and ease in sharing information, this has also led to the fast propagation of virus in computer

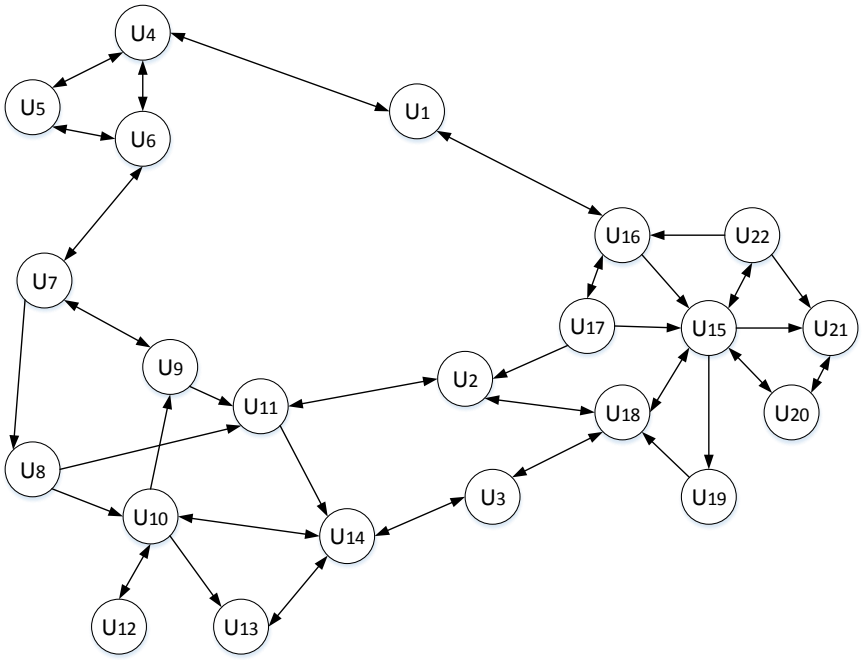


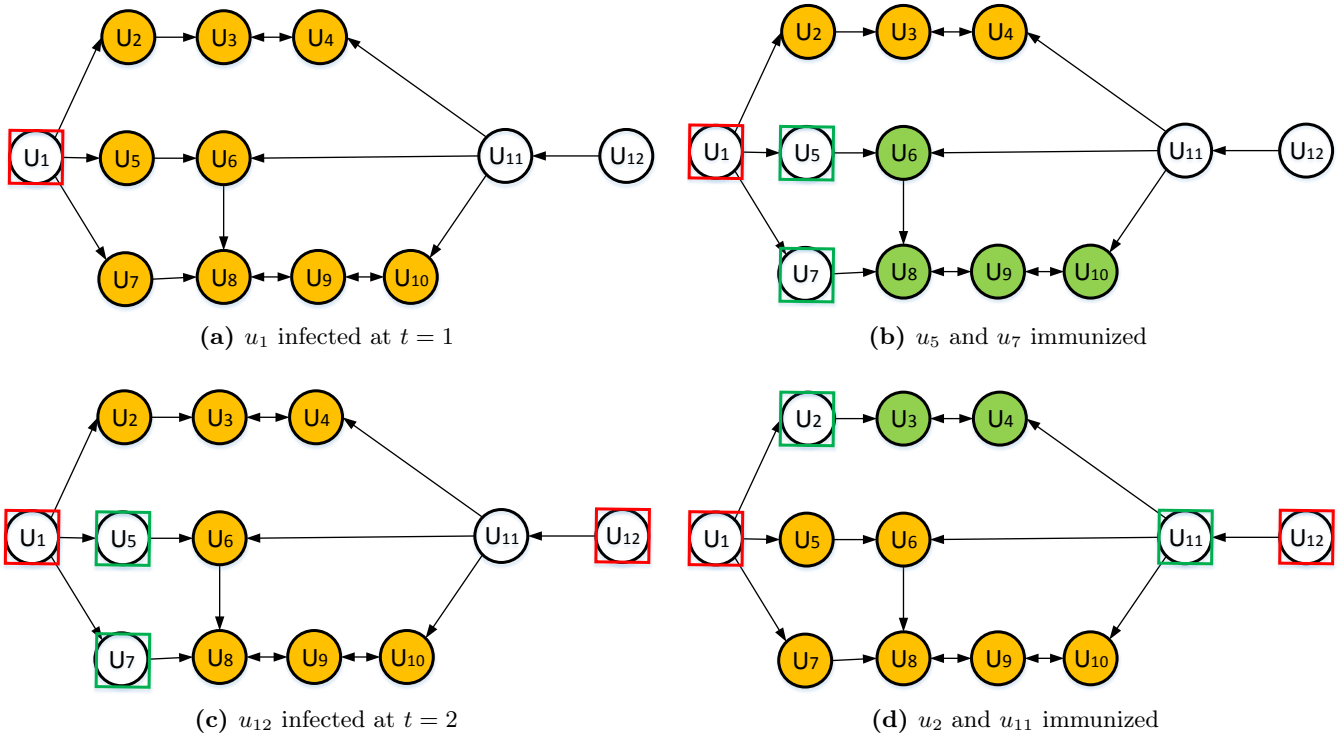
Figure 1.2: Example Social Network with Broker

networks as well as rumors in the information networks. For example, when the Malaysia Airlines Flight 370 was reported missing on the morning of March 8th 2014, millions of messages flooded the social network. A message saying "Flight MH370 has resurfaced and landed safely in Nanming, China." received millions of shares before it was proven to be a rumor. This is not only a waste of resources in information dissemination, but also causes confusions as well as panics in a very short time. In order to reduce the impact brought about by such epidemics, researchers study the problem of minimizing the influence of misinformation. The works in [KSM09, SHL15b, ZP15, WCF+16, ZP14b] have identified nodes/edges to be *immunized* such that rumors cannot propagate via these nodes/edges. Other works in [HSCJ11, TNT12, TBM10, BAEA11, CCRea11] focus on starting *truth campaigns* to combat rumors. They select a small set of users to spread the truth and assume that when a user is aware of the truth, s/he becomes immune to the rumor. In the following, we discuss the challenges in advancing these two tasks by considering temporal aspects of rumor breakouts and propagation.

Challenge One: Considering Infectious Period in Immunization. Given a graph, which nodes should we immunize so that the maximum number of nodes will remain *healthy* under attacks? In computer networks, a computer is healthy if it is not infected with some malware and in social networks, a user is said to be healthy if he/she does not share/re-tweet any rumor message. In the former case, immunizing a node may mean the shutting down of certain host servers, while in the latter case, this could mean the suspension of a user's account.

Existing works have mostly focused on immunizing nodes *before* the virus/rumor starts spreading [CHbA03, TPT+10, WCWF03]. Recently, the realization is that it is more practical and meaningful to identify nodes to immunize *while* the attack is underway. Zhang et al. [ZP14a] formalize the problem of node immunization as follows: given a social network and prior information about which are the infected nodes, select k nodes to immunize such that the number infected of nodes is minimized.

We further recognize that when an attack happens in the real world, there is a time period where multiple independent sources may actively infect different nodes [MZL12]. This infectious period is often observed in the early stage of a contagion [BAH12]. For example, mendacious messages about MH370's safe landing in Nanming were posted by many users independently, citing different websites, in the first six hours of its attack. This will create a problem when the resources available for immunization are limited.



(a) u_1 infected at $t = 1$

(b) u_5 and u_7 immunized

(c) u_{12} infected at $t = 2$

(d) u_2 and u_{11} immunized

Figure 1.3: Example Network with Rumor Breakouts in Two Time Steps

We will illustrate this issue with an example shown in Figure 1.3a where u_1 is the initial infected node (marked in red). Suppose the infectious period is 2 time points and u_{12} becomes infected at the second time point. We also assume we only have enough resources to immunize 2 nodes. Looking at time point 1, we would choose u_5 and u_7 and save the nodes colored in green (Figure 1.3b). However, when u_{12} becomes infected at time point 2, we have no resource to immunize additional nodes, as a result, the saved nodes are once again infected, leaving us with only 2 healthy nodes (Figure 1.3c). On the other hand, if we

immunize only u_2 at time point 1 and then immunize u_{11} at time point 2, we would have 4 healthy nodes at the end as shown in Figure 1.3d.

This example motivates us to consider the entire infectious period on the whole when making decision as to which nodes to immunize. Hence, we propose the *node immunization over infectious period* problem and identify the k nodes to immunize within the infectious period so as to minimize the number of rumor infected nodes.

Challenge Two: Blocking Misinformation in Time Delayed Diffusions. Instead of immunizing some nodes or edges, the works in [HSCJ11, TNT12, TBM10, BAEA11, CCRea11] focus on starting *truth campaigns* to combat rumors. They select a small set of users to spread the truth and assume that when a user is aware of the truth, s/he becomes immune to the rumor. Budak et al.[BAEA11] formally define the problem of *eventual influence limitation* or *influence blocking maximization* in order to identify the set of nodes to broadcast the truth information and minimize the spread of rumor. They design a Multi-campaign Independent Cascade model, in which both rumor and truth campaigns are actively propagating in the network. They propose heuristic methods based on centrality measures for finding the truth starters.

Recent studies [MSM15, CGD12] have recognized that the diffusion of information is time-sensitive, in particular, the propagation of information may incur a certain amount of time delay [CLZ12, CDPW14]. Unfortunately, existing works [HSCJ11, TNT12, BAEA11] studying the EIL problem do not consider the time aspect of the propagation process. As a result, the nodes selected by existing methods are not optimal when time delays are introduced.

Figure 1.4 shows the same example network as in Figure 1.3. To illustrate the temporal effect. we associate each edge (u, v) with a value indicating the the time taken for information to diffuse from node u to v. Suppose u_1 is the rumor starter, and we can select only one node to start a truth campaign. If we do not consider time delays, we would select u_7 which would prevent u_8, u_9 and u_{10} from being influenced by the rumor since these nodes are nearer to u_7. However, if we consider time delay, then we should select u_5 because any information from u_5 will reach u_8 first before the rumor from u_1 can reach u_8 via u_7. In so doing, we prevent an additional node u_6 from being influenced by the rumor. We say that u_6, u_8, u_9 and u_{10} have been *saved*.

Further, we observe that there is a deadline beyond which a rumor will lose its effect naturally. For example, in the case of an election, once the voting day is over, any rumors concerning the candidates will have no effect on the outcome. Under such scenarios, we should minimize the number of users affected by rumors before the deadline. In Figure 1.4, if we set the deadline to 8, then we should start the truth campaign at u_2 instead of u_5 as this would prevent two nodes (u_3 and u_4) instead of one node (u_6) from being affected by the rumor within the deadline. Note that any rumor from u_1 can only affect the nodes u_7, u_8, u_9 and u_{10} after the deadline, which will not affect the election outcome.

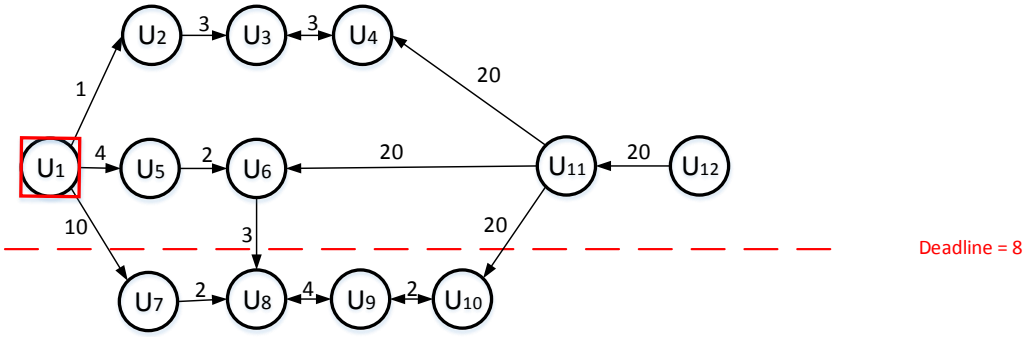


Figure 1.4: Example Network with Time Delays

Clearly, taking into account time delays and deadline to minimize the effect of rumors can result in very different solutions even in the same network. With this in mind, we study the *temporal influence blocking* problem that aims to minimize the number of nodes influenced by misinformation given time delays and deadline.

1.2 Thesis Contributions

This thesis aims to elevate the research of managing information diffusion from two aspects: maximizing good influence and minimizing bad influence. Specifically, the contributions of this thesis can be summarized as follows.

- We formalize the *targeted influence maximization* problem and show it is NP-hard. We introduce the *weighted reverse reachable* (WRR) trees and develop a sampling-based approximate algorithm called Target-IM that considers the event location and deadline. This algorithm generates a pool of WRR trees and greedily selects k nodes that can cover the largest number of WRR trees. Given a social network $G = (V, E)$ where V is the set of nodes and E is the set of edges, our proposed algorithm Target-IM is able to return a $(1 - 1/e - \varepsilon)$-approximate solution in $O(k^2(|V| + |E|)\log|V|/\varepsilon^2)$ time. The experimental results on real-world datasets demonstrate that Target-IM outperform the state-of-the-art approaches.
- We formally define the problem of finding top-k brokers in social networks and show its NP-hardness as it can be reduced from the k-densest subgraph problem. In order to tackle this problem, we design a heuristic solution to find the top-k brokers based on the theory of *weak ties* [Gra83]. We propose a connection-aware scoring function and design an algorithm called WeakTie to find the top-k brokers. Further, in order to handle the highly dynamic nature of social networks, we also develop incremental algorithms: WeakTie-Local for unidirectional networks and WeakTie-Bi for bidirectional

networks. Finally, we demonstrate the effectiveness of the proposed approach and show how the brokers can be useful in two information diffusion tasks: (a)structural spanner detection and (b) mention recommendation. We found our detected brokers improve the precision of spanner detection by 35%. For mention recommendation, brokers can help diffuse a message to users located far away from the author and reach 23% more distince users.

- We formalize the problem of node immunization over an infectious period where we are allowed to distribute the immunization resources. We show that the problem is NP-hard and present the NIIP algorithm. Our assumption is that the resources available for immunization are limited, that is, we only have resources to immunize k nodes. Hence, we propose a simulation-based approach to estimate how k should be distributed over the infectious period. At the same time, we design a scoring function to model a node's immunization ability in the network. Based on the estimated value of k at time point t, denoted as k_t, we select k_t nodes with the highest scores for immunization at time point t. After immunizing these nodes, we consider the new infected nodes and update the scores of all the affected nodes. The process repeats until we have selected k nodes to immunize. We conduct extensive experiments on several real world datasets covering computer networks, information networks and social networks to demonstrate the efficiency and effectiveness of the proposed NIIP algorithm.

- We study the temporal influence blocking problem to select the best k nodes to start a truth campaign so as to minimize the number of nodes influenced by rumors in the presence of time delays before some deadline α. We call this problem *Temporal Influence Blocking* (TIB). We present a sampling and greedy-based solution which consists of two phases. In the first phase, we evaluate the *threat level* of each node at each time point. In other words, we estimate how many nodes could possbily be influenced by a node if it is infected by rumor. In the second phase, we utilize *weighted reverse reachable* (WRR) trees to determine the set of nodes that can reach other nodes before the rumor starters reach them. We introduce a scoring function to estimate the expected number of nodes that can be saved by a node before a deadline. We generate a pool of WRR trees and greedily select k nodes with the highest scores as the final output. We conduct extensive experiments on multiple real-world social network datasets to demonstrate the effectiveness and efficiency of our proposed solution.

1.3 Organization of the Thesis

The remainder of the thesis is organized as follows:

- Chapter 2 presents a review of existing works on modeling propagation process, identifying influential users and preventing the spread of misinformation.
- Chapter 3 defines the targeted influence maximization problem that considers deadline and locations of users. This work is published in Proc. of 25^{th} ACM International Conference on Information and Knowledge Management [SHL16].
- Chapter 4 defines brokers in social networks and propose heuristic methods for identifying them in dynamic social networks. This work is published in Proc. of 24^{th} ACM International Conference on Information and Knowledge Management [SHL15a].
- Chapter 5 introduces the node immunization over infectious period problem and describes how we manage to assign immunization nodes over time to limit the spread of misinformation. This work is published in Proc. of 24^{th} ACM International Conference on Information and Knowledge Management [SHL15b].
- Chapter 6 introduces the temporal influence blocking problem that considers time delays and deadline in blocking the diffusion of misinformation with truth campaigns. This work is published in Proc. of 33^{rd} IEEE International Conference on Data Engineering [SHL17].
- Chapter 7 concludes this thesis and discusses future work.

Chapter 2

Related Work

Online social networks play a significant role in the dissemination of information at large scales. Much effort has been made in order to observe, analyze and understand this phenomenon, ranging from information diffusion modeling to influential spreaders identification. In this section, we review the existing efforts around managing information diffusion in online social networks.

2.1 Modeling Information Diffusion

Nowadays online social networks allow millions of users to produce and consume content on a global scale. News, events as well as rumors happen and diffuse in online social networks everyday and researchers have devoted much effort to capture and understand their diffusing patterns.

An online social network is formally represented by a graph where nodes are users and edges are relationships. Users publish messages to share different kinds of information, and followers of post authors can see the message and further choose to share it with their followers if they are interested. Globally, the content produced by the members of an online social network can form a sequence of sharing behaviors, with later people watching the content of earlier people. This phenomenon is called *information cascade*.

Modeling how information spreads is of great interest for understanding the formation of a popular event, stopping the spread of viruses and analyzing how misinformation spread. In this section, we first introduce the *explanatory models* that infers the underlying information cascade given a sequence of activations, and then discuss the *predictive models* that aim to predict how a specific diffusion process would unfold in a given network.

2.1.1 Explanatory Models

Explanatory models aim to trace the path of a information based on given data. Gomez et al. [GRLK10, GRBS11] propose to explore correlations in nodes infection times to infer the

structure of the spreading cascade. They assume that activated nodes influence each of their neighbors independently with some probability. Thus, the probability that one node had transmitted information to another is decreasing in the difference of their activation time. They propose an iterative algorithm NETINF based on submodular function optimization for finding the spreading cascade that maximizes the likelihood of given data. They extend their work by modeling the likelihood of a node infecting another at a given time via a probability density function depending on infection times and the transmission rate between the two nodes. They develop NETRATE to infer pairwise transmission rates and the graph of diffusion by formulating and solving a convex maximum likelihood problem.

Both NETINF and NETRATE consider the underlying network is static but real-world social networks evolve very quickly. Gomez et al. [GRLS13] extend NETRATE and propose a time-varying inference algorithm, INFOPATH, that uses stochastic gradients to provide online estimates of the structure and temporal dynamics of a network that changes over time.

In order to tackle the data aquisition bottleneck caused by crawling API limitations, Choudhury et al. [DCLS+10] analyze how missing data impacts the results of inferring diffusion paths. Based on experiments on Twitter data, they find that sampling methods that consider both network topology and users' attributes such as activity and localization allow capturing information diffusion with lower error in comparison to naive strategies, like random or only activity-based sampling. Given that some data will often be missing during crawling, Sadikov et al. [SMLGM11] develop a method based on a k-tree model to estimate the properties of the full diffusion cascade, such as its size or depth given only part of the actual activation sequence.

2.1.2 Predictive Models

Predictive models aim to predict how diffusion will unfold in a given network if some nodes are initially influenced. *Independent Cascade* (IC) model [GLM01] and *Linear Threshold* (LT) model [Gra78] are the two most widely used models. They assume the existence of a static graph structure underlying the diffusion and focus on the patterns of the diffusion process. They assume the diffusion takes place in a directed graph where each node can be activated or not with a monotonicity assumption, i.e. activated nodes cannot be deactivated any more. The IC model requires a diffusion probability to be associated to each edge whereas the LT model requires an influence degree to be defined on each edge and an influence threshold for each node. For both models, the diffusion process proceeds iteratively in a synchronous way along a discrete time-axis, starting from a set of initially activated nodes, commonly named *early adopters* [Rog04]. In the IC model, the newly activated nodes try once to activate their neighbors with the probability defined on the edge linking them at each iteration. In the LT model, at each iteration, the inactive nodes are activated by their

activated neighbors if the sum of influence degrees exceeds their own influence threshold. Successful activations are only effective at the next iteration. In both models, the process ends when no new transmission is possible, i.e. no node is newly activated at the current iteration.

Galuba et al. [GAC+10] use the LT model to simulate the graph of diffusion given the early adopters. Their model relies on parameters such as information virality, pairwise users' degree of influence and users' probabilities of adopting any information. The LT model is fitted on the data describing the beginning of the diffusion process by optimizing the parameters using the gradient ascent method. However, LT cannot reproduce realistic temporal dynamics. Saito et al. [SKOM09, SNK08, SOY+11] relax the synchronicity assumption of traditional IC and LT graph-based models by proposing asynchronous extensions. They proceed iteratively along a *continuous time axis* and require the same parameters as their synchronous counterparts plus a time-delay parameter on each edge of the graph. They provide a method to learn the functional dependency of the model parameters from nodes' attributes. They formulate the task as a maximum likelihood estimation problem and an update algorithm that guarantees the convergence is derived. However, they only experiment with synthetic data and fail to provide a practical solution. Guille et al. [GH12] also model the propagation process as asynchronous independent cascades. They develop the T-BAsIC model (i.e. Time-Based Asynchronous Independent Cascades), where parameters are not fixed numerical values but functions depending on time. The model parameters are estimated from social, semantic and temporal nodes' features using logistic regression. Based on the IC and LT models, the works in [BKS07, BFO10] design the Competitive Independent Cascade (CIC) model and Competitive Linear Threshold (CLT) model. The influence rules of CIC and CLT models are the same as in IC and LT models respectively. The difference is that in the competitive models, multiple parties are trying to broadcast their own information and nodes influenced by any party will not be influenced by other parties in future.

The above mentioned models are based on a specific network structure. Some other approaches are non-graph based. They do not assume the existence of a specific graph structure and have been mainly developed to model epidemiological processes. *SIR* and *SIS* are the two popularly studied models [Het00, New03], where S stands for "susceptible", I for "infected" and R for recovered. In both models, nodes in the S class switch to the I class with a fixed probability β. Then, in SIS, nodes in the I class switch to the S class with a fixed probability γ, whereas in the case of SIR they permanently switch to the R class. The percentage of nodes in each class is expressed by simple differential equations. Both models assume that every node has the same probability to be connected to another and thus connections inside the population are made at random.

Leskovec et al. [LMF+07] propose a simple and intuitive SIS model that requires a single parameter β. It assumes that all nodes have the same probability β to adopt the information

and nodes that have adopted the information become susceptible at the next time-step (i.e. $\gamma = 1$). Wang et al. [WWX12] propose a Partial Differential Equation (PDE) based model to predict the diffusion of an information injected in the network by a given node. The topology of the network is considered only in terms of the distance from each node to the source node, *i.e.* shorter distance from the seed node indicates higher probability of being influenced.

2.2 Identifying Influential Spreaders

Influence Maximization (IM) is the problem of choosing the most potential individuals in a network to spread out information in order to trigger a widespread adoption of a product. This problem has applications in viral marketing, where a company may wish to spread the advertisement of a new product via the most influential individuals in popular social networks. With online social networking sites such as Facebook and Twitter attracting hundreds of millions of people online each day, this motivates the research community to conduct extensive studies on various aspects of the influence maximization problem. There are two major types of approaches for identifying influential users in a social network. In this section, we first introduce the *centrality-based* approach that analyzes nodes' structural properties for measuring their influence, and then discuss the *diffusion model-based* approach that adopts a particular diffusion model, such as the IC or LT model, and computes high influence nodes based on it.

2.2.1 Centrality-based Approach

A large body of studies use the centrality measures to estimate the influence of a user. Centrality measures include degree [OAS10], betweenness [Fre77], closeness[Dan06] and so on. High degree nodes correspond to users with many connections to his/her friend, high betweenness nodes correspond to users who carry the largest amount the shortest paths traversing through them, and, high closeness nodes correspond to the users whose overall sum of distance to all other nodes is the smallest. Such measures estimate the general importance of nodes in a network but such importance does not entirely reflect the diffusion ability of users on social platform.

Kitsak et al. [KGH+10] show that the best spreaders are not necessarily the most connected people in the network. They find that the most efficient spreaders are those located within the core of the network as identified by the k-core decomposition analysis [Sei83]. Basically, the principle of the k-core decomposition is to assign a core index to each node such that nodes with the lowest values are located at the periphery of the network while nodes with the highest values are located in the center of the network. The innermost nodes thus forms the core of the network. Brown et al. [Fen11] observe that the results of the k-shell

decomposition on Twitter network are highly skewed. Therefore, they propose a modified algorithm that uses a logarithmic mapping, in order to produce fewer and more meaningful k-shell values.

Cataldi et al. [CDCS10] propose to use the well known PageRank algorithm [Pre02] to assess the distribution of influence throughout the network. The PageRank value of a given node is proportional to the probability of visiting that node in a random walk of the social network, where the set of states of the random walk is the set of nodes. [CSH+14, SGaMZ13] also design PageRank-based algorithms that iteratively updates the score of each node based on previous diffusion behaviors to identify users for viral marketing.

Other researchers have identified users that are located at critical positions in the network for diffusing information [LLS11, NPW01, WLG+13]. Cui et al. [CWL+11] and Luo et al. [LOTW13] focus on identifying influential retweeters. They examine the author's followers' retweet history, active time and interest to identify influential retweeters who are able to accelerate the diffusion of information. The work in [SS12] defines the importance of nodes based on the direction of information flow and whether the end users are members in the same community. [LT13] assumes that the set of communities is known and proposes two methods *i.e.* MaxD and HIS to find users who connect different communities and are responsible for information diffusion across communities. In the first method *MaxD*, the idea is to remove one node at a time such that the removal of this node will lead to the maximum decrease in the minimum cut in the given set of communities. The second method *HIS* looks for nodes that are incident with high degree nodes in different communities. The above mentioned methods aim to identify nodes that can maximize the influence for a certain individual or community, and may not be optimal for maximizing influence over the entire network.

2.2.2 Diffusion Model-based Approach

Kempe et al. [KKT03] propose to use the IC and LT (previously described in Section 2.1.2) models to tackle the influence maximization problem. This problem asks, for a parameter k, to find a k-node set of maximum influence in the network. The influence of a given set of nodes corresponds to the number of activated nodes at the end of the diffusion process according to IC or LT model, using this set as the set of initially activated nodes. They provide an approximation for this optimization problem using a greedy hill-climbing strategy based on submodular functions. In [KKT03], they define a function f that evaluates the influence of a given set of seed nodes by running Monte Carlo simulations a large number of times. Let G be the input network, the greedy algorithm for influence maximization problem is illustrated in Algorithm 2.1. Ever since, much research interest has been aroused to either improve the efficiency of the IM algorithm or to extend the IM problem by incorporating more constraints such as topic and temporal aspects.

Algorithm 2.1: Greedy Algorithm for IM

input : G, k, f
output : Seed Set S

Initialize $S = \emptyset$
while $|S| \leq k$ **do**
    select $u \leftarrow \arg\max_{w \in S-V}(f(S \cup \{w\}) - f(S))$
    $S = S \cup \{u\}$
end
Return S;

Improving IM algorithm

Leskovec et al. [LKG+07] exploit submodularity to develop an efficient algorithm called *Cost-Effective Lazy Forward* (CELF) selection algorithm, based on a lazy-forward optimization in selecting seeds. The idea is that marginal gain of a node in the current iteration cannot be better than its marginal gain in the previous iterations. This optimization avoids the recomputation of marginal gains of all the nodes in any iteration, except the first one. In [LKG+07], the authors empirically shows that CELF dramatically improves the efficiency of the greedy algorithm. Other optimization based on the CELF algorithms are developed in [GLL11, ZYF+14].

Kimura et al. in [KS06] assumes that each node is activated only through the shortest paths from an initial active set and develop the *Shortest-Path Model* (SPM) and *SP1 Model* (SP1M). These two models are special cases of the IC model. The idea is that the majority of the influence flows through shortest paths. In SPM, only the most efficient information spread can occur. SP1M, which slightly generalize SPM, instead considers the top-2 shortest paths from one node to another. For these models, the influence of each target set S can be exactly and efficiently computed, and the provable performance guarantee for the natural greedy algorithm can be obtained.

Based on the above contribution in SPM and SP1M, Chen et al. [CWW10, CWY09, CYZ10] extend this idea by considering *Maximum Influence Paths* (MIP) instead of shortest paths. A maximum influence path between a pair of nodes is the path with the maximum propagation probability from one node to another. The main idea of this heuristic scheme is to use local arborescence structures of each node to approximate the influence propagation. The maximum influence paths between every pair of nodes in the network can be computed by the Dijkstra shortest-path algorithm [Che03]. Then they ignore the MIPs with probability smaller than a influence threshold θ, which effectively restrict influence to a local region.

When considering the influence propagation through these local arborescences, the diffusion model refers to the Maximum Influence Arborescence (MIA) model [CWW10]. However, these heuristics would not perform well on high influence graphs, that is, when the influence probabilities through links are large. Wang et al. [WCSX10] propose an alternative approach. They argue that most of the diffusion happens only in small communities, even though the overall networks are huge. Taking this as an intuition, they first split the network into smaller communities, and then, restrict the influence spread to the community to which the node belongs to compute the marginal gain of a prospective seed node. Dinh et al. [DNT12] discover that the propagation in a social network often fades quickly within only few hops from the sources, counteracting the assumption on the self-perpetuating of influence considered in some literature. They investigate the cost-effective massive, and fast propagation (CFM) problem and proposed an algorithm *VirAds* to minimize the seeding cost and to tackle the problem on large-scale networks. In early stages, the algorithm behaves similar to the degree-based heuristics that favors vertices with high degree. However, after a certain number of vertices have been selected, VirAds will make the selection based on the information within d-hop neighborhood around the considered vertices, which is different from degree-based heuristic that considers only one-hop neighborhood.

The above mentioned methods adopt a greedy framework that, at each iteration, adds in one node with the highest estimated influence. Much effort has been made to accelerate computation by estimating the seed set's influence more efficiently. However, most of those the estimations are wasted since, in each iteration of a greedy approach, we are only interested in the node set with the largest expected spread. Borgs et al. [BBCL14] make a theoretical breakthrough and present a *Reverse Influence Sampling* (RIS) algorithm for influence maximization under the IC model. It avoids estimating the influence for all possible node sets, the limitation of the greedy approach, by introducing the reverse reachable (RR) set defined as follows.

Definition 1. *Let v be a node in network G, and g be a graph obtained by removing each edge e in G with $1 - p(e)$ probability. The reverse reachable (RR) set for v in g is the set of nodes in g that can reach v. (That is, for each node u in the RR set, there is a directed path from u to v in g.)*

By definition, if a node u appears in an RR set generated for a node v, then u can reach v via a certain path in G. As such, u should have a chance to activate v if we run an influence propagation process on G using $\{u\}$ as the seed set. Borgs et al. [BBCL14] show a result that is consistent with the above observation: If an RR set generated for v has β probability to overlap with a node set S, then, when we use S as the seed set to run an influence propagation process on G, we have β probability to activate v. Based on this result, RIS algorithm runs in two steps: it first generates a certain number of random RR sets from G, and then, it selects k nodes to cover the maximum number of RR sets generated. They show that their algorithm returns a $(1 - \frac{1}{e} - \epsilon)$-approximate solution with at least $1 - n^{-l}$

probability, and prove that it is near-optimal. Their method was adopted and accelarated by Tang et al. [TSX15, TXS14]. Nguyen et al. [NTD] further improve the efficiency of RIS algorithm by developing a SSA sampling framework that largely reduced the number of generated RR sets.

Extending IM problem

Apart from improving the efficiency of IM solutions, some other works have extended influence maximization problem by incorporating more realistic aspects. Li et al. [LCF+14] consider influencing the set of users within a given region and extend the heuristic method in [CWW10] to calculate the regional incremental influence of a user. They further analyz the upper and lower bounds of a user's incremental spread to prune out users with low influence. Zhou et al. [ZCL+15] introduce a function to measure the likelihood of a user's offline adoption of a product given the locations of the user and the product. The works in [DYM+14, FCBM14, YDG+15] study influence propagation in event-based social networks by analyzing users' behaviors for predicting user interest in the nearby events.

The works in [CLZ12, CDPW14, LCXZ12] recognize the significance of considering deadline in influence maximization tasks. Chen et al. [CLZ12] observe that when a user adopts an idea, he needs to wait for a meeting event to happen in order to influence his neighbor, thus introducing some delay in the propagation process. They proposed a MIA-M algorithm based on the notion of *maximum influence arborescence* (MIA). Each MIA keeps track of the path with the highest influence probability between each pair of nodes. Cohen et al. [CDPW14] propose timed influence maximization by assuming that traversing an edge takes a certain amount of time. The authors in [LCXZ12] assign to each edge a distribution of meeting probability, indicating how likely the pair of nodes will meet with each other over a certain time frame. They propose an *Influence Spreading Path* (ISP) algorithm for computing the influence spread of each node. Chen et al. further extend the Independent Cascade model to incorporate log-in events to form the IC-L model [CLZ12]. They demonstrate that estimating the additional influence spread of a node under IC-L model is complicated since they need to consider the order of different nodes' log-in time. They employ a dynamic programming approach to compute the incremental influence on a node by explicitly enumerating possible log-in ordering of its in-neighbors.

Li et al. [LZT15] and Chen et al. [CFL+15] consider the diffusion of information is related to users' interests: it is more important to diffuse an advertisement to the users who are interested in the product. They model each user's interest with a weighted term vector that captures the preference in different topics. The relavance between an advertisement and a user can be obtained by applying the tf-idf model on the topic space. They extend the RIS algorithm by sampling high relavance nodes with higher probability. Li et al. [LZT15] further adopts offline sampling to achieve real-time response to influence maximization queries.

2.3 Preventing Misinformation Propagation

Apart from identifying influential spreaders, it is also important to prevent the propagation of misinformation or rumors. Existing efforts on limiting the spread of misinformation can be categorized into three approaches: *pre-emptive*, *immunization* and *truth-campaign*. In this section, we introduce the state-of-the-art methods for limiting information spread from these three approaches.

2.3.1 Pre-emptive Approach

Pre-emptive approach focuses on reducing the network's ability to diffuse any information. This approach does not require the knowledge of rumor starters and selects nodes to immunize *before* the start of an epidemic [BLP03, CDK10, CHbA03, MKC+04]. Abbassi and Heidari [AH11] consider nodes with high degrees and betweenness values tend to be good candidates for immunization. Tong et al. [TPER+12, TPT+10] finds the set of users whose removal would slow down the propagation of information in a network. They assume that information propagation follows some epidemic model in the form of a matrix whose first eigenvalue corresponds to the speed of propagation. Based on this, they iteratively remove a node at a time to minimize the first eigenvalue. Their method only applies to the specific diffusion model and does not show how these users are placed in the network. Similarly, Wang et al. [CWW+08, WCWF03] analyze the topology of an arbitrary graph and found the larger the first eigenvalue of the graph is, the easier for the virus to propogate. Hence, they identify the nodes and edges whose removal would minimize the largest eigenvalue of the graph. They show that their methods is more effective than centrality-based methods.

Cohen et al. [CHbA03] propose a heuristic *acquaintance immunization* strategy, which is further studied in [BLP03]. They select the random acquaintances of random nodes, in which case high degree nodes will be selected with a higher chance. Experiments show that their strategy can greatly reduce the immunization threshold from 80% to 40%. Kimura et al. [KSM08] consider finding a set of k links to block such as to minimize the expected contamination area of the undesirable misinformation and propose an algorithm for efficiently finding an approximate solution on the basis of a naturally greedy strategy. Many works also compare the performance of a limited number of pre-determined sequences of interventions (like school closure, antiviral for treatment) within simulation models [DPV+07, FCF+06]. All these works propose pre-emptive strategies for immunization without the awareness of infected nodes.

2.3.2 Immunization Approach

The work in [ZP14a] points out that taking into account the infected nodes to make decision about whom to immunize is more practical and meaningful. They formalize the problem of Data-Aware Vaccination problem given the initial set of infected nodes and propose a polynomial-time heuristic algorithm called Dava. Dava constructs a *dominator tree* by aggregating the infected nodes into one node as the root and examine the paths from the root to all other nodes in the network. In the dominator tree, a node u is a child of node v if every path from the root must traverse v to reach u. Thus, with the dominator tree, we can say that any descendant of node u in the tree will be healthy if u is immunized. Given this observation, they iteratively select the node that maximizes the expected number of saved descendant nodes under the *Susceptible-Infected-Recovered* model [Het00]. Once selected, this node and its descendants are removed and a new dominator tree is constructed and the process is repeated to select the next node for immunization. The algorithm greedily chooses one node at a time until k nodes are selected. Zhang et al. further extend their work in [ZP14b] by considering there is a probability that infected nodes are not detected. They apply the Sample Average Approximation framework [KSHdM02] to reduce the stochastic optimization problem to a deterministic version by sampling the uncertainty distribution to generate a finite number of deterministic cases. Based on those sampled cases, they apply the Dava algorithm to to generate the solution. However, the Dava algorithm does not consider the joint immunization effect of two nodes together. In the case that immunizing node u and v together can save the most nodes but immunizing u or v alone cannot save any node, the optimal solution $\{u, v\}$ will never be returned by Dava since u or v will never be selected for immunization at any iteration.

2.3.3 Truth-Campaign Approach

Instead of immunizing some nodes or edges, the works in [HSCJ11, TNT12, TBM10, BAEA11, CCRea11] focus on starting *truth campaigns* to propagate the truth in order to combat the effect of rumors actively. Budak et al.[BAEA11] formally define the problem of *eventual influence limitation* or *influence blocking maximization* under the IC model in order to identify the set of nodes to broadcast the truth information and minimize the spread of rumor. They design a Multi-campaign Independent Cascade model, in which both rumor and truth campaigns are actively propagating in the network. They propose heuristic methods based on centrality measures for finding the truth starters. The first heuristic is degree centrality which identifies the nodes with the highest degrees. The second method is called early infectees which selects the nodes that are expected to be infected at an early stage of the rumor propagation. Lastly, the most effective method in [BAEA11] is to select largest infectees. This method runs a sufficient number of simulations to choose the nodes that are expected to infect the highest number of nodes if they were to be infected themselves.

The works in [HSCJ11, BFO10] study the problem of limiting information spread under the LT model. They propose a competitive LT model that allows multiple campaigns to compete. They show that the problem of influence blocking under competitive LT model is submodular and develop a greedy approximation algorithm by estimating the influence of each node with its local structure. Tsai et al. [TNT12] further extend the problem by considering both competing campaigns actively adjust its own strategy regarding its opponent's strategy. They propose a *Local Shortest-paths for Multiple Influencers* (LSMI) algorithm to measure the incremental gain of choosing one node given a set of selected nodes and the competitive campaign based on the local shortest paths. Their method follows the assumption in [KS06] that each node is most likely activated through the shortest paths from other nodes. Such assumption saves much computation time but, on the other hand, makes its estimation of influence inaccurate.

Chapter 3

Targeted Influence Maximization

Influence maximization problem has attracted much attention since it was introduced by Domingos et al. [DR01] for viral marketing. The essential idea is that by targeting a small set of users, it is possible to trigger a large range of diffusion through the word-of-mouth effect in social networks. In the chapter, we address the first challenge in influence maximization tasks, *i.e.*incorporating user locations as well as deadline for the marketing task at the same time. We formally define the targeted influence maximization problem and propose our sampling based approach that provides an approximation guarantee.

3.1 Problem Definition

We adopt the Independent Cascade model with Login events (IC-L) [CLZ12] as our diffusion model. Given a network $G = (V, E)$ where V is a set of nodes and E is a set of edges, we have an initial seed set of nodes S at time step 0. Each node $v \in G$ has a probability $login(v)$ to be online and active at each time step. When a node $v \in G$ logs in at time point $t > 0$, its neighboring node u will have a probability $p(u, v)$ of influencing v if there is an edge from u to v, u is influenced at $t' < t$, and t is the first time v logs in after t'.

In the real world, users tend to go to an event if it is near to them rather than a remote event in a different city or country. For each user $v \in G$, v is associated with a location l_v. Given an event's location γ, we have a function $f(\gamma, l_v) \in [0, 1]$ to measure the likelihood of v participating in the event. Note that $f(\gamma, l_v) \to 1$ when l_v is close to γ.

We aim to select k users such that they can influence the largest number of *registered* participants before the deadline.

Definition 2 (Targeted Influence Maximization)**.** *Let $G = (V, E)$ be a network where V is the set of nodes and E is the set of edges. Each node $v \in V$ is associated with a location l_v and a login probability $login(v)$ to be online, while each edge $\langle u, v \rangle$ has an influence probability $p(u, v)$. Given a deadline α and an event location γ, we aim to find a set of k*

users S that maximizes the expected number of registered users:

$$\Phi(S) = \sum_{v \in I(S,\alpha)} f(\gamma, l_v)$$

where $I(S, \alpha)$ is the set of influenced users within α time points under the IC-L model.

Note that the location-based influence maximization problem in [LCF+14] and the time-critical influence maximization problem in [CLZ12] are special cases of the targeted influence maximization problem. For the former, we set $\alpha = \infty$, and for the latter, we have $f(\gamma, l_v) = 1$ for all $v \in V$.

3.2 Proposed Approach

In this section, we present a sampling-based method called Target-IM to solve the targeted influence maximization problem.

The work in [BBCL14] introduces a *Reverse Influence Sampling (RIS)* algorithm for traditional influence maximization. RIS generates a set of Reverse Reachable (RR) sets by randomly sampling nodes in the graph. It then applies a greedy selection process based on maximum coverage [Vaz01] to find k nodes that cover the largest number of RR sets. Once a node v is selected, all RR sets that contain v are considered covered and can be removed.

In order to take deadline constraint into consideration, instead of RR sets, we define a Weighted Reverse Reachable tree to model the propagation delay incurred by login events.

Definition 3 (Weighted Reverse Reachable Tree)**.** *Given a graph G, let g be a graph instance of G obtained by removing each edge $\langle u, v \rangle$ in G with probability $1 - p(u, v)$. Let α be the deadline. A WRR tree for a node r, denoted as T_r, is a $(\alpha+1)$-level tree such that each path $p \in T_r$ from r to a child node v corresponds to a path from v to r in the graph instance g. Each node $v \in T_r$ is associated with the probability of v influencing r within α steps, given by reach$(v \xrightarrow{\alpha} r)$.*

Note that if there are q paths between v and r in g, we will create q copies of v, denoted as v^i, $1 \leq i \leq q$, in the WRR tree and there will be q branches in T_r from r to each of the copies of v.

Target-IM (see Algorithm 3.1) works in two phases. In the first phase (lines 1-3), we sample θ number of weighted reverse reachable trees. In the second phase (line 4), we greedily select k nodes that cover the most number of WRR trees generated in the first phase. However in our case, the selected node from a WRR tree T_r may only have partial influence on the root r, that is to say, other nodes in T_r may continue to exert some degree of influence on r because selecting these nodes can increase the probability of r being influenced. Hence,

even if we have selected a node from T_r, we cannot remove T_r from subsequent consideration to find the next node. Finally, the algorithm returns the k nodes (line 5).

Algorithm 3.1: Target-IM

input : 1. Social network G
2. Deadline α
3. Location γ

output : Seed set S

1 Initiate $\mathcal{T} = \emptyset$.
2 **while** $|\mathcal{T}| < \theta$ **do**
3 $\quad \mathcal{T} = \mathcal{T} \cup$ WRRGenerate(G, α)
4 **end**
5 Seed set S = GreedySelect($\mathcal{T}$)
6 Return S

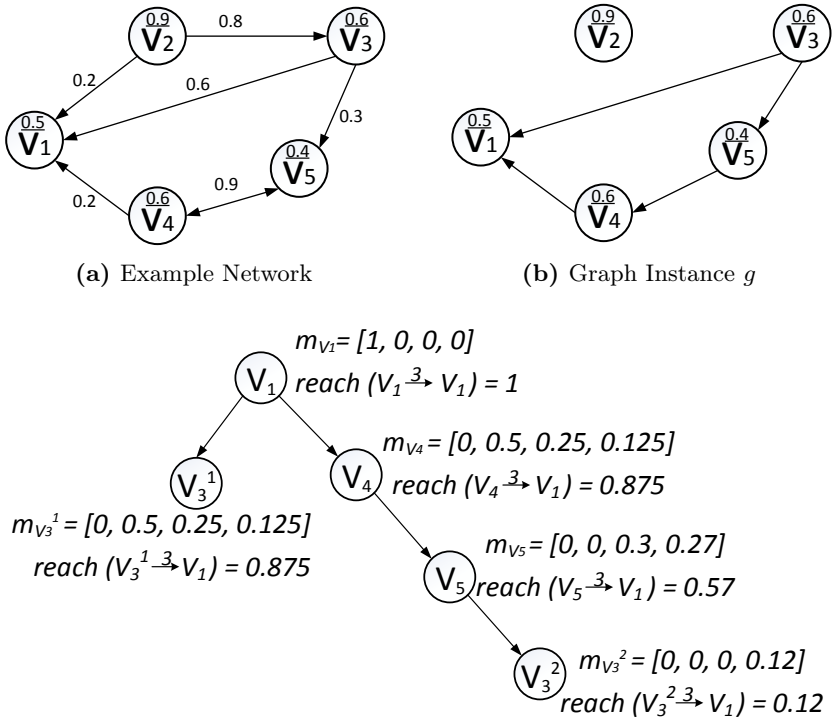
In the following subsections, we elaborate on how to generate a single weighted reverse reachable tree, and describe the greedy selection process. We also give a theoretical analysis of the performance bounds of Target-IM.

3.2.1 Generation of WRR Trees

Given the graph G, we create a graph instance g of G by flipping a coin for each edge $< u, v >$ such that there is a probability $p(u, v)$ that the edge will be retained in g. With this graph instance, we can generate a WRR tree rooted at r as follows. We perform a breath-first traversal starting from r following the in-links. Each time we reach a node v, we create a corresponding node and add the node and its associated edge to the WRR tree. Note that if v has been visited before, a new copy of v is created.

Figure 3.1a shows an example social network where the influence probability is shown on each edge, and the login probability of each user is underlined in the node. Consider the node v_1. The probabilities of including the edges $\langle v_2, v_1 \rangle$, $\langle v_3, v_1 \rangle$ and $\langle v_4, v_1 \rangle$ are 0.2, 0.6, and 0.2 respectively. Figure 3.1b shows a possible graph instance g obtained. Then we perform a breadth-first traversal starting from v_1. Since v_3 will be visited twice, we have two copies of v_3, namely v_3^1 and v_3^2, in the generated WRR tree rooted at v_1 (see Figure 3.1c).

Next, we describe how to compute the value associated with each node in the WRR tree, that is, $reach(v \xrightarrow{\alpha} r)$. Recall that $reach(v \xrightarrow{\alpha} r)$ denotes the probability of v influencing r within the deadline α. We use a $(\alpha + 1)$-vector m_v to aid in this computation where the j^{th} entry of m_v, denoted as $m_v[j]$, keeps track of the probability of v reaching the root in exactly j steps.



(a) Example Network **(b)** Graph Instance g

(c) Generated WRR Tree

Figure 3.1: Illustration of Generating WRR Tree.

Suppose v is at the d^{th} level in the WRR tree. It will take at least d steps for v to reach r as there are at least d nodes along the path from v to r. When $j < d$, the probability that v can reach r in exactly j steps is 0.

However, when $j \geq d$, let node w be the immediate parent of v along the path from v to r, then the probability that v reaches r in exactly j steps is the probability that the parent w logs in on the $(i+1)^{th}$ step multiplied by the probability that w takes exactly $j-i-1$ steps to reach r. The probability w logs in on the $(i+1)^{th}$ step is the probability w does not log in in the first i steps and logs in on the $(i+1)^{th}$ step, that is $(1-login(w))^i \cdot login(w)$. The probability w takes $j-i-1$ steps to reach r is given in $m_w[j-i-1]$, hence we have

$$m_v[j] = \sum_{i=0}^{j-1}((1-login(w))^i \cdot login(w) \cdot m_w[j-i-1]) \tag{3.1}$$

where $login(w)$ is the login probability of w.

With this, we have:

$$reach(v \xrightarrow{\alpha} r) = \sum_{i=0}^{\alpha} m_v[i] \tag{3.2}$$

Back to our example in Figure 3.1. Suppose $\alpha = 3$, we have $m_{v_1}[0] = 1$ since v_1 can reach itself in 0 step with a probability 1. Consider node v_3^1 is now influenced, the probability of v_1 logging in at the next step is 0.5. Then the probability of v_3^1 influencing v_1 in 1 step is 0.5. Hence, we have

$m_{v_3^1}[0] = 0$

$m_{v_3^1}[1] = (1 - 0.5)^0 \times 0.5 \times m_{v_1}[0] = 0.5$

Similarly, the probability that node v_1 logs in at step 2 is $(1 - 0.5)^1 \times 0.5$. Then we have

$m_{v_3^1}[2] = (1 - 0.5)^1 \times 0.5 \times m_{v_1}[0] = 0.25$

The final WRR tree is shown in Figure 3.1c.

Algorithm WRRGenerate (see Algorithm 3.2) gives the details. Given a network G and deadline α, we first randomly sample a node from G and create the corresponding root node r for the WRR tree (Lines 1-2). Line 3 initializes m_r and $reach(r \xrightarrow{\alpha} r)$. For each incoming edge $\langle v, w \rangle$ of w, we decide with a probability $p(v, w)$ whether it should be added to the WRR tree (Line 5). If the decision is to add the edge, all of the in-link neighbours of v are placed in a queue for subsequent processing. For the nodes that are added to the tree, we compute their $reach(v \xrightarrow{\alpha} r)$ in Lines 7-9. The algorithm terminates when all $(\alpha + 1)$-level nodes have been processed.

Algorithm 3.2: WRRGenerate(G, α)

```
1  Randomly choose a node r uniformly and start BFS.
2  Create tree root r.
3  Initialize m_r = [1, 0, ··· , 0] and reach(r --α--> r) = 1 .
4  while breath-first traversal is within α levels of r do
5      Flip a coin with probability p(v, w) for each in-link ⟨v, w⟩.
6      if decision is YES then
7          Create node v or a copy of v if v has been visited before
8          Compute m_v[j] for each j ∈ [0, 1, ··· , α] using Equation 3.1.
9          Compute reach(v --α--> r) using Equation 3.2.
10         Place v's in-link neighbours in the processing queue.
11     end
12     else
13         Continue.
14     end
15 end
16 Return the WRR tree.
```

Note that each node in G is equally likely to be sampled by WRRGenerate. However, given a targeted location γ, a node v whose location is far away from γ should be given less consideration. As such, we should sample nodes that are close to γ as the target nodes. Further, we realize that if the expected contribution of a node in influencing the root of a WRR tree is smaller than some threshold η, then we will abandon this node and terminate the tree construction.

We design a more efficient algorithm WRRGenerate$^+$ by focusing on nodes that are closer to the target location. We first sample a node r with probability $\frac{f(\gamma,l_r)}{\sum_{v\in V} f(\gamma,l_v)}$ where nodes with higher $f(\gamma, l_r)$ value possesses a higher chance of being selected. Then we initialize m_r and start the BFS from the sampled node. For any visited node v, we create a copy of this node, compute m_v and $reach(v \xrightarrow{\alpha} r)$ as done in WRRGenerate algorithm.

Note that in Algorithm WRRGenerate, no matter how unlikely for the nodes to *register* the root, we will always perform a breadth-first traversal until α levels. However, in Algorithm WRRGenerate$^+$, we stop the traversal as soon as we realize the probability of some nodes *registering* the target node is smaller than a pre-defined threshold η. We realize that for two nodes v and w where v is an ancestor of w in T_r, it is impossible for w to have a higher probability of *registering* r than v. Hence, Algorithm WRRGenerate$^+$ will check whether $f(\gamma, l_r) \cdot reach(v^x \xrightarrow{\alpha} r) < \eta$. If so, that means all v's descendant nodes' probabilities of *registering* r cannot be larger than η. As such, we terminate the traversal and continue with the other nodes in the processing queue.

3.2.2 Greedy Selection

After generating a pool of WRR trees $\mathcal{T}$, the next phase is to find k nodes that cover the largest number of WRR trees. Let $T_r \in \mathcal{T}$ be a WRR tree with root node r, and $\psi(S, T_r)$ be the probability of a seed set S influencing r. For each node v in a WRR tree T_r, we maintain a value $weight(v, S, T_r)$ to indicate the contribution by v in influencing the root r where

$$weight(v, S, T_r) = \begin{cases} reach(v \xrightarrow{\alpha} r) & S = \emptyset \\ \psi((S \cup \{v\}), T_r) - \psi(S, T_r) & \text{otherwise} \end{cases} \tag{3.3}$$

Recall that $f(\gamma, l_r)$ gives the probability that the root node of T_r will register for an event at location γ. At each iteration, we select the node with the highest

$$\sum_{T_r \in \mathcal{T}} weight(v, S, T_r) \cdot f(\gamma, l_r)$$

to put into S. We repeat the process k times to get our seed set S.

This approach is simple but inefficient, as we need to compute the probabilities of $S \cup \{v\}$ influencing r, and S influencing r whenever we update the weight of a node v in T_r.

Careful analysis reveals that there are two cases to consider when we update the weight of a node v that does not correspond to any node in S. Let $F_S \subset S$ be the set of nodes that have no ancestors corresponding to nodes in S.

- Case 1. v is a descendant of $u \in F_S$.
  The probability of r getting influenced when both u and v are selected is the same as the probability of r getting influenced when u is selected only. This is because any influence that v can exert on r must go through u. Once u is selected, the additional contribution of v on r is 0. In other words, $weight(v, S, T_r) = 0$.

- Case 2. v is not a descendant of any node in F_S.
  Probability of r getting influenced given S
  $= 1-$ the probability that none of the nodes in S influence r
  $= 1 - \prod_{w \in F_S}(1 - reach(w \xrightarrow{\alpha} r))$

  Probability of r getting influenced given $S \cup \{v\}$
  $= 1 - \prod_{w \in F'_S}(1 - reach(w \xrightarrow{\alpha} r)))$
  where

$$F'_S = \begin{cases} F_S \cup \{v\} - \{u\} & v \text{ is an ancestor of } u \in F_S \\ F_S \cup \{v\} & \text{Otherwise} \end{cases}$$

  Then we have

$$weight(v, S, T_r) = \prod_{w \in F'_S}(1 - reach(w \xrightarrow{\alpha} r))) - \prod_{w \in F_S}(1 - reach(w \xrightarrow{\alpha} r)) \tag{3.4}$$

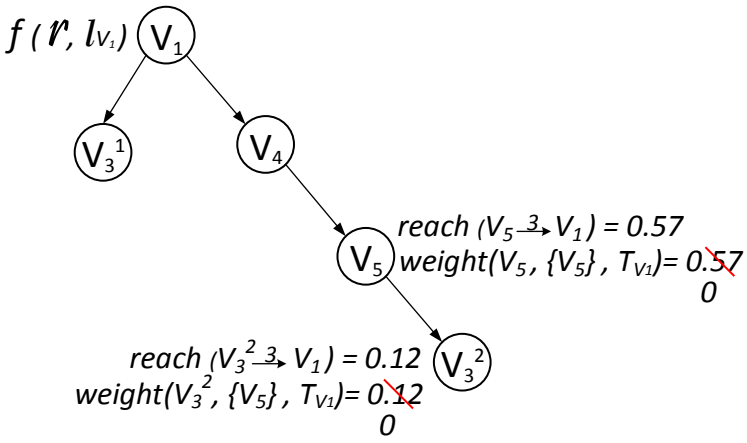
Consider the example WRR tree in Figure 3.2. Suppose $S = \{v_5\}$. Since v_3^2 is a descendant of v_5 (see Figure 3.2a), we set
$weight(v_3^2, \{v_5\}, T_{v_1}) = 0$.

The nodes v_1 and v_4 are ancestors of v_5 (see Figure 3.2b), their weights are updated as follows:

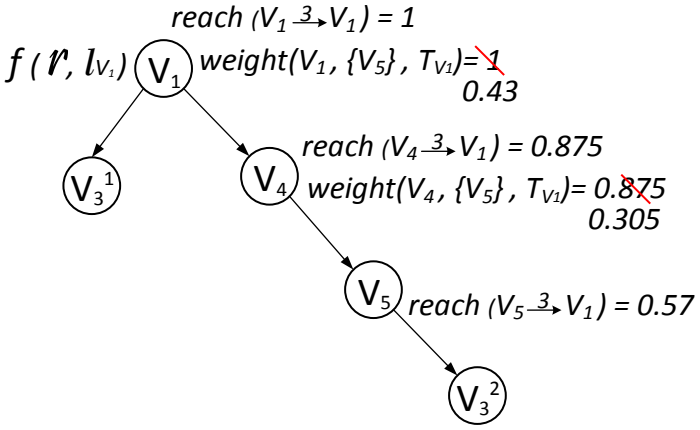
$$\begin{aligned} weight(v_1, \{v_5\}, T_{v_1}) &= (1 - 0.57) - (1 - 1) \\ &= 0.43 \\ weight(v_4, \{v_5\}, T_{v_1}) &= (1 - 0.57) - (1 - 0.875) \\ &= 0.305 \end{aligned}$$

Finally, v_3^1 is on a different path (see Figure 3.2c) and its weight is updated as follows:

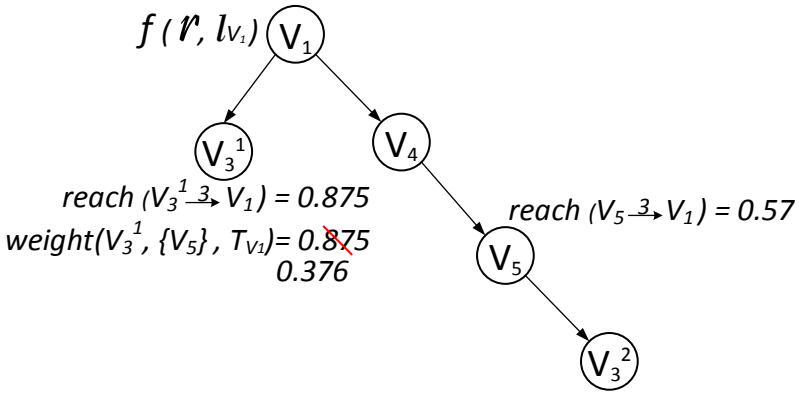
$$\begin{aligned} weight(v_3^1, \{v_5\}, T_{v_1}) &= (1-0.57) - (1-0.57) \cdot (1-0.875) \\ &= 0.376. \end{aligned}$$



(a) Updating Descendants



(b) Updating Ancestors



(c) Updating Other Paths

Figure 3.2: Illustration of Updating Weights in a WRR Tree.

Algorithm 3.3 gives the details. At each iteration, we select a node u with the highest $\sum_{T_r \in \mathcal{T}} weight(u, S, T_r) \cdot f(\gamma, l_r)$ value (Line 3). Then for each WRR tree T_r that involves the newly selected node u, we update the weights of the other nodes in Lines 4-15. For any node v in T_r, we first check whether the node is a copy of u. Then for each of the remaining nodes, we see whether it is a descendant of u (Lines 8-9). If not, we check whether it is an ancestor of u and update $weight(v, S, T_r)$ accordingly (Line 11-15). After updating the weights, we select the next node u' with the highest $\sum_{T_r \in \mathcal{T}} weight(u', S, T_r) \cdot f(\gamma, l_r)$ among all WRR trees. We repeat the process for k times and return the final set S as output.

Algorithm 3.3: GreedySelect($\mathcal{T}$, k, $f()$)

```
1  Initiate S = ∅.
2  for j = 1 to k do
3      Select u with highest probability of registering the roots
       Σ_{T_r∈T} weight(u, S, T_r) · f(γ, l_r).
4      foreach WRR tree involving u do
5          Set weight of any node copy corresponding to u to 0.
6          Identify the users in S with no ancestors in S as F_S.
7          foreach node v in the tree T_r do
8              if v is a descendant of any node in F_S then
9                  Set weight(v, S, T_r) = 0.
10             else
11                 if v is an anscestor of node u ∈ F_S then
12                     F'_S = F_S ∪ {v} − {u}.
13                 else
14                     F'_S = F_S ∪ {v}.
15                 end
16                 Compute weight(v, S, T_r) with Equation 3.4.
17         end
18     end
19 end
20 Return S.
```

3.2.3 Estimation of θ

In this section, we provide an estimate of θ, the number of WRR trees to be sampled, so that the k nodes returned by Algorithm GreedySelect is guaranteed to be within $(1-1/e-\varepsilon)$ of the optimal solution.

Given a seed set S and a node v, Borgs et al. [BBCL14] have shown that the probability that S overlaps with a random RR set equals to the probability of S influencing v in the traditional IC model. Based on this result, we establish the following lemma.

Lemma 1. *Let T_r be a WRR tree and S be a set of selected nodes. Suppose p is the probability that a node in S correspond to some node in T_r, and $\psi(S, T_r)$ is the probability of S influencing r. The probability that S successfully influencing r to be registered in the original graph G under IC-L model is $p \cdot \psi(S, T_r) \cdot f(\gamma, l_r)$.*

Proof. Let R_r be the RR set generated for r on the same graph instance g by restricting the depth of BFS to α levels. According to Definition 3, the set of nodes in R_r correspond to copies of the same set of nodes in T_r. Based on the result in [BBCL14], p equals the probability of S influencing r via some path(s) in the original graph under IC model. Since the IC-L model reduces to the traditional IC model when all nodes have login probabilities 1, p is equal to the probability of S influencing r via the same path(s) under IC-L model if all nodes have login probability 1. Since T_r keeps track of these path(s) via which S influences r, so $\psi(S, T_r)$ is the probability that S influences r via these paths with different login probabilities. Taking into account the location of r in registering for an event at location γ, we have $p \cdot \psi(S, T_r) \cdot f(\gamma, l_r)$ which gives the probability of S influencing r to be registered in the original graph under IC-L model. □

Next, we define $\Psi(S, \mathcal{T})$ which gives the total expected number of root nodes in $\mathcal{T}$ that are *registered* if S is the initial set of influenced nodes as follows:

$$\Psi(S, \mathcal{T}) = \sum_{T_r \in \mathcal{T}} \psi(S, T_r) \cdot f(\gamma, l_r)$$

The function $\Psi(S, \mathcal{T})$ is monotone and submodular, indicating the greedy algorithm can produce a $(1 - 1/e)$-approximate solution [Vaz01].

Theorem 1. *$\Psi(S, \mathcal{T})$ is monotone and submodular.*

Proof. Clearly, given S, adding in other nodes into the S can never decrease $\psi(S, \mathcal{T})$. In addition, $f(\gamma, l_r)$ is always greater than or equal to 0. Hence, for $x \notin S$, we always have

$$\sum_{T_r \in \mathcal{T}} \psi(S \cup \{x\}, T_r) \cdot f(\gamma, l_r) \geq \sum_{T_r \in \mathcal{T}} \psi(S, T_r) \cdot f(\gamma, l_r)$$

By definition of Ψ, we have

$$\Psi(S \cup \{x\}, \mathcal{T}) \geq \Psi(S, \mathcal{T})$$

Thus Ψ is monotone.

Next, Ψ is submodular if for any two given seed sets S_1 and S_2 where $S_1 \subset S_2$, and a node $x \notin S_2$, we have $\Delta p_1 \geq \Delta p_2$ where $\Delta p_1 = \Psi(S_1 \cup \{x\}, \mathcal{T}) - \Psi(S_1, \mathcal{T})$ and $\Delta p_2 = \Psi(S_2 \cup \{x\}, \mathcal{T}) - \Psi(S_2, \mathcal{T})$.

From the definition of Ψ, we have

$$\begin{aligned} & \Psi(S_1 \cup \{x\}, \mathcal{T}) - \Psi(S_1, \mathcal{T}) \\ = & \sum_{T_r \in \mathcal{T}} \psi(S_1 \cup \{x\}, T_r) \cdot f(\gamma, l_r) - \sum_{T_r \in \mathcal{T}} \psi(S_1, T_r) \cdot f(\gamma, l_r) \\ = & \sum_{T_r \in \mathcal{T}} (\psi(S_1 \cup \{x\}, T_r) - \psi(S_1, T_r)) \cdot f(\gamma, l_r) \end{aligned}$$

Similarly,

$$\begin{aligned} & \Psi(S_2 \cup \{x\}, \mathcal{T}) - \Psi(S_2, \mathcal{T}) \\ = & \sum_{T_r \in \mathcal{T}} (\psi(S_2 \cup \{x\}, T_r) - \psi(S_2, T_r)) \cdot f(\gamma, l_r) \end{aligned}$$

We prove that Ψ is submodular by contradiction. Suppose $\Delta p_1 < \Delta p_2$. Then for a $T_r \in \mathcal{T}$ we have

$$\psi(S_1 \cup \{x\}, T_r) - \psi(S_1, T_r) < \psi(S_2 \cup \{x\}, T_r) - \psi(S_2, T_r)$$

that is,

$$weight(x, S_1, T_r) < weight(x, S_2, T_r)$$

Let $S1 = \emptyset$ and $S_2 = \{u\}$. Given a node $x \neq u$,

$$weight(x, S_1, T_r) = reach(x \xrightarrow{\alpha} r)$$

Further, we have
$weight(x, S_2, T_r)$

$$= \begin{cases} 0 & u \text{ is an ancestor of } x \\ reach(x \xrightarrow{\alpha} r) - reach(u \xrightarrow{\alpha} r) & u \text{ is a descendant of } x \\ (1 - reach(u \xrightarrow{\alpha} r)) \cdot reach(x \xrightarrow{\alpha} r) & \text{otherwise} \end{cases}$$

Since $weight(x, S_2, T_r)$ is always less than $reach(x \xrightarrow{\alpha} r)$, and $weight(x, S_1, T_r) = reach(x \xrightarrow{\alpha} r)$, which contradicts our assumption that $\Delta p_1 < \Delta p_2$. Thus Ψ is submodular. □

Recall that $\Phi(S)$ is defined as the number of users who register as a result of S's influence under the IC-L model. Let n be the number of nodes in the original graph G, *i.e.* $n = |V|$. We can compute the probability of S influencing a node in G to be registered as $\frac{\Phi(S)}{n}$.

Let $\mathcal{T}_S \subset \mathcal{T}$ be the subset of WRR trees that have nodes in S. Then the probability of S influencing the root of a WRR tree $T_r \in \mathcal{T}_S$ to register is given by

$$\frac{\Psi(S,\mathcal{T})}{|\mathcal{T}_S|} = \frac{\sum_{T_r \in \mathcal{T}_S} \psi(S,T_r) \cdot f(\gamma, l_r)}{|\mathcal{T}_S|}$$

Since $\frac{|\mathcal{T}_S|}{|\mathcal{T}|}$ gives the probability that S has some node corresponding to a node in a WRR tree, based on Lemma 1, we have

$$\frac{|\mathcal{T}_S|}{|\mathcal{T}|} \cdot \frac{\Psi(S,\mathcal{T})}{|\mathcal{T}_S|} = \frac{\Phi(S)}{n}$$

Hence we have

$$\frac{n}{|\mathcal{T}|} \cdot \Psi(S,\mathcal{T}) = \Phi(S) \tag{3.5}$$

Let M be the maximum number of registered nodes by any size-k node set in G and S^M is the set that maximizes $\Phi(S)$, *i.e.*$\Phi(S^M) = M$. Suppose S^* is the set that can *register* the largest number of tree roots in $\mathcal{T}$, that is, $S^* = \text{argmax}_S \Psi(S,\mathcal{T})$.

Based on the results established in [TSX15], we have the following lemmas:

Lemma 2. *If $\theta > \theta_1$ where $\theta_1 = \frac{2n \cdot log(1/\delta_1)}{M \cdot \varepsilon_1^2}$, then*

$$(n/\theta) \cdot \Psi(S^M,\mathcal{T}) \geq (1-\varepsilon_1) \cdot M \tag{3.6}$$

holds with a probability of at least $(1-\delta_1)$, $\delta_1 \in (0,1)$, $\varepsilon_1 > 0$.

Lemma 3. *If $\theta > \theta_2$ where $\theta_2 = \frac{(2-2/e) \cdot n \cdot log(\binom{n}{k}/\delta_2)}{M \cdot (\varepsilon - (1-1/e)\cdot\varepsilon_1)^2}$, then*

$$\frac{n}{\theta} \cdot \Psi(S,\mathcal{T}) - \Phi(S) < \varepsilon_2 \cdot M \tag{3.7}$$

holds with a probability of at least $(1-\delta_2)$, $\delta_2 \in (0,1)$, $\varepsilon_1 < \varepsilon$ and $\varepsilon_2 = \varepsilon - (1-1/e)\cdot\varepsilon_1$.

With the above results, we derive a bound for the approximate solution obtained by Target-IM.

Theorem 2. *Given $\varepsilon_1 < \varepsilon$ and δ_1, δ_2 in$(0,1)$ with $\delta_1 + \delta_2 \leq \frac{1}{n}$, we set $\theta = max\{\theta_1, \theta_2\}$ to ensure that Target-IM returns a $(1-1/e-\varepsilon)$-approximate solution with at least $1-\frac{1}{n}$ probability, where θ_1 and θ_2 are determined from Lemma 2 and Lemma 3 respectively.*

Proof. Let S be the set of nodes returned by Target-IM, $S^* = \text{argmax}_S \Psi(S,\mathcal{T})$ and $S^M =$

$\text{argmax}_S \Phi(S)$.

According to Equation 3.7, we have

$$\Phi(S) > \frac{n}{\theta} \cdot \Psi(S, \mathcal{T}) - \varepsilon_2 \cdot M.$$

Since $\Psi(S, \mathcal{T})$ is a $(1 - 1/e)$-approximate solution for $\Psi(S^*, \mathcal{T})$ according to Theorem 1, we have

$$\Phi(S) > (1 - 1/e) \cdot \frac{n}{\theta} \cdot \Psi(S^*, \mathcal{T}) - \varepsilon_2 \cdot M.$$

Since S^* maximizes $\Psi(S^*, \mathcal{T})$, we know $\Psi(S^*, \mathcal{T}) \geq \Psi(S^M, \mathcal{T})$. Then we can have

$$\Phi(S) > (1 - 1/e) \cdot \frac{n}{\theta} \cdot \Psi(S^M, \mathcal{T}) - \varepsilon_2 \cdot M.$$

According to Equation 3.6, we know $\frac{n}{\theta} \cdot \Psi(S^M, \mathcal{T}) \geq (1 - \varepsilon_1) \cdot M$. By replacing $\frac{n}{\theta} \cdot \Psi(S^M, \mathcal{T})$ with $(1 - \varepsilon_1) \cdot M$, we have

$$\Phi(S) > (1 - 1/e) \cdot (1 - \varepsilon_1) \cdot M - \varepsilon_2 \cdot M.$$

Since $\varepsilon_2 = \varepsilon - (1 - 1/e) \cdot \varepsilon_1$ according to Lemma 3, we have

$$\begin{aligned} \Phi(S) &> (1 - 1/e) \cdot (1 - \varepsilon_1) \cdot M - (\varepsilon - (1 - 1/e) \cdot \varepsilon_1) \cdot M \\ &= ((1 - 1/e) \cdot (1 - \varepsilon_1) - \varepsilon + (1 - 1/e) \cdot \varepsilon_1) \cdot M \\ &= ((1 - 1/e) \cdot (1 - \varepsilon_1 + \varepsilon_1) - \varepsilon) \cdot M \\ &= (1 - 1/e - \varepsilon) \cdot M \end{aligned}$$

Thus we establish the bound for Target-IM. □

Theorem 2 allows us to control the result quality by setting an appropriate value for ε.

3.2.4 Time Complexity of Target-IM

Given an input graph $G = (V, E)$, the time complexity for generating θ WRR trees is $O(k(|V| + |E|)\log|V|/\varepsilon^2)$ based on the results in [TSX15]. This is equivalent to the total number of edge traversed by the breath-first search. The second phase in Target-IM is to greedily select k nodes by calling Algorithm 3.3, GreedySelect. Since in the generation of WRR trees, each edge traversal will result in the creation of a node in the WRR tree, we have $O(k(|V| + |E|)\log|V|/\varepsilon^2)$ number of nodes in $\mathcal{T}$. In the worst case, for each iteration of Algorithm 3.3, we have to update the weights of all nodes in $\mathcal{T}$. This results in a time complexity of $O(k^2(|V| + |E|)\log|V|/\varepsilon^2)$. Hence the overall time complexity of Target-IM algorithm is $O(k^2(|V| + |E|)\log|V|/\varepsilon^2)$.

3.3 Experiments

In this section, we present the results of experiments to evaluate the performance of the proposed methods on several real-world datasets. All the experiments are run on a linux machine with 2 Xeon E5440 2.83 GHz CPU and 16G RAM.

Datasets. We use four location-based social network datasets: GOWALLA, TWITTER, WEIBO and FOURSQUARE. We assume that the most frequent check-in location of a user u is his location l_u. Table 3.1 gives the total number of nodes and edges, as well as the average number of neighbours per node of these datasets.

Table 3.1: Characteristics of Datasets

Dataset	#Nodes	#Edges	Ave # neighbours per node
GOWALLA	26,316	106,271	8.07
TWITTER	554,372	2,402,720	8.58
WEIBO	1,020,730	16,490,916	32.3
FOURSQUARE	4,899,219	28,484,755	11.6

Methods. We compare the performance of the following methods in our experiments.

- Target-IM. This is our proposed method which computes the k nodes that maximizes the number of registered users by taking into account both deadline and user location.
- Target-IM$^+$. This method calls Algorithm WRRGenerate$^+$ instead of WRRGenerate to speed up the runtime by omitting nodes that are unlikely to participate given the event location.
- MIA-L [CLZ12]. This method computes the incremental influence spread within deadline α of a given node under the IC-L model using *maximum in-arborescence*, and does not take into consideration location information.
- Expansion [LCF+14]. This is the state-of-the-art location-aware influence maximization method. Note that this method does not consider deadline.
- IMM [TSX15]. This is the state-of-the-art approach for the traditional influence maximization problem. IMM can be considered as a special case of Target-IM by setting $\alpha = \infty$ and $\forall v \in V$, $f(\gamma, l_v) = 1$.

Parameter Setting. We randomly generate the coordinates for the target location γ. The location function $f(\gamma, l_v)$ is set according to [LCF+14] where $f(\gamma, l_v) = 1$ if the distance between l_v and γ is within a predefined threshold, and 0 otherwise. This distance threshold is controlled by the number of nodes within a circular region centred at γ. Similar to

[LCF+14], we set the number of nodes for GOWALLA to be 10^4, TWITTER and WEIBO to be 10^5, and FOURSQUARE to be 10^6.

We set the login probability of each node by randomly choosing a real number from the interval $[0, 1]$. Further, the influence probability $p(u, v)$ for each edge $\langle u, v\rangle$ is given by $1/inDegree(v)$ where $inDegree(v)$ is the number of incoming edges to node v as in [KKT03, TXS14]. For Target-IM$^+$ and MIA-L, nodes influencing the target with a probability less than η will not be considered. A larger η can result in more registered users despite longer running time. Based on [CLZ12], we set $\eta = 1/320$.

For Target-IM and Target-IM$^+$, the parameter ε controls the number of WRR trees generated. A smaller ε value indicates a tighter approximation bound but longer running time. As in [TSX15], we fix $\varepsilon = 0.5$ in all our experiments as it provides higher efficiency.

3.3.1 Experiments with Deadline and Location

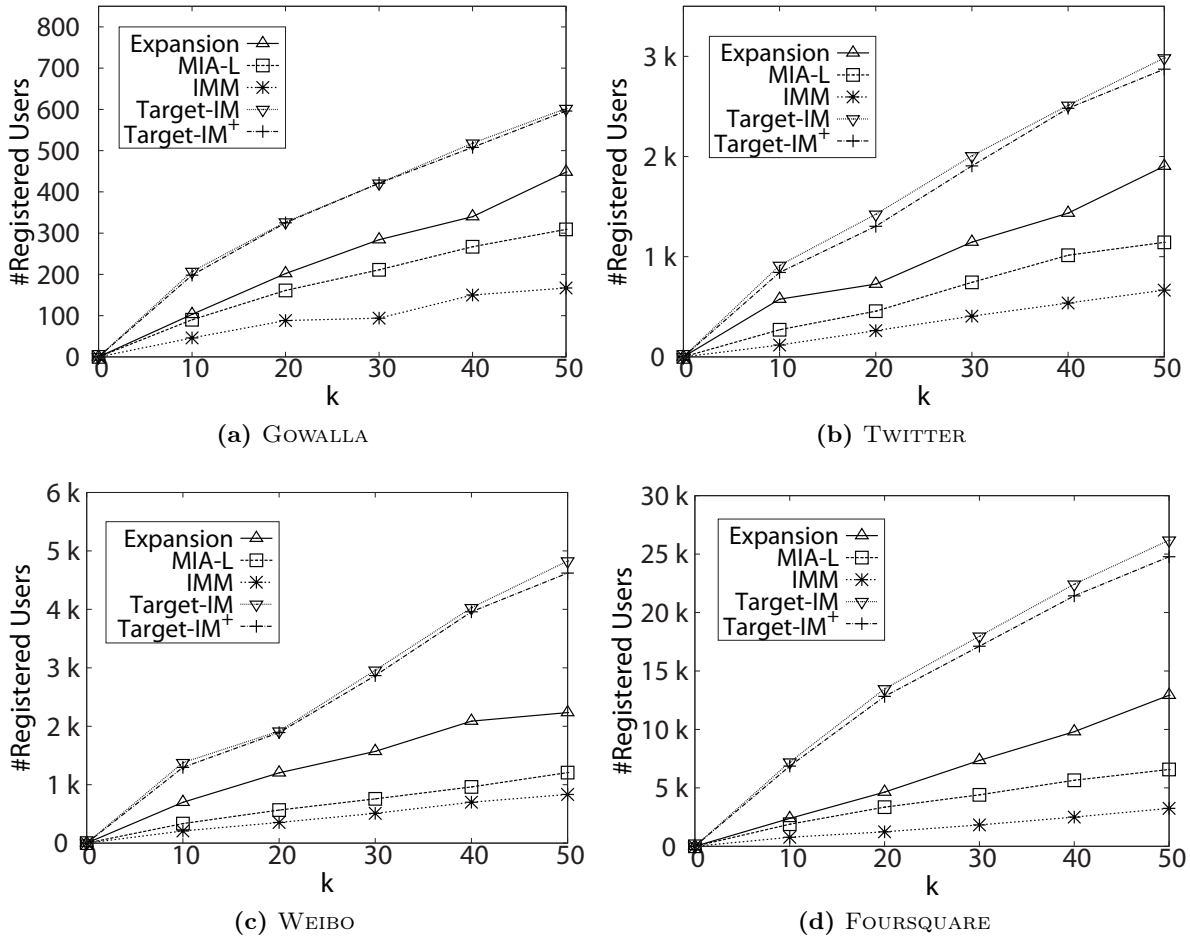
In this set of experiments, we evaluate the performance of the various methods by running each method five times, each time with a random target location. We record the number of registered users as a result of the influence of the k users returned by these methods, and report the average results of the five runs.

First, we set the deadline $\alpha = 10$ and vary k from 10 to 50. Figure 3.3 shows the results.

We observe that Target-IM and Target-IM$^+$ give the best performance as they take into account both location and deadline (+30% in GOWALLA and +80% in the other three datasets). Further, the rate of increase is the steepest for Target-IM and Target-IM$^+$ as k increases.

Expansion finds the set of nodes that has the highest influence within the region without considering deadline. This would include many nodes that are influenced after the deadline α, which are not the registered users in our targeted influence maximization problem. MIA-L computes the influence probability with login events and deadline. However, it does not consider the user location and may find nodes that are unlikely to be registered.

IMM has the poorest performance since it finds nodes generally with large influence spread. However, such nodes may not be the registered users when we restrict the effective nodes to a specific region and limit the propagation time.

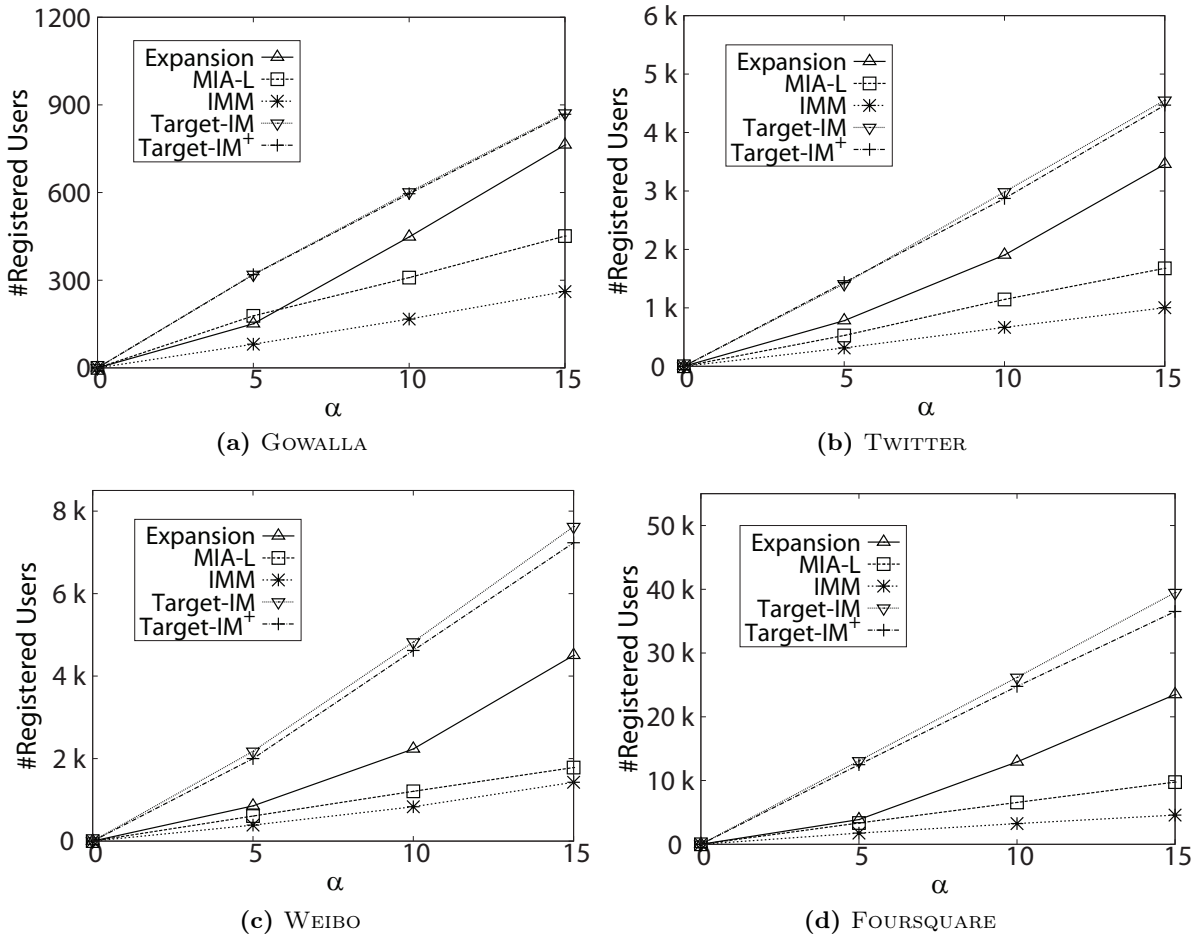


(a) Gowalla **(b)** Twitter

(c) Weibo **(d)** Foursquare

Figure 3.3: Effect of k ($\alpha = 10$).

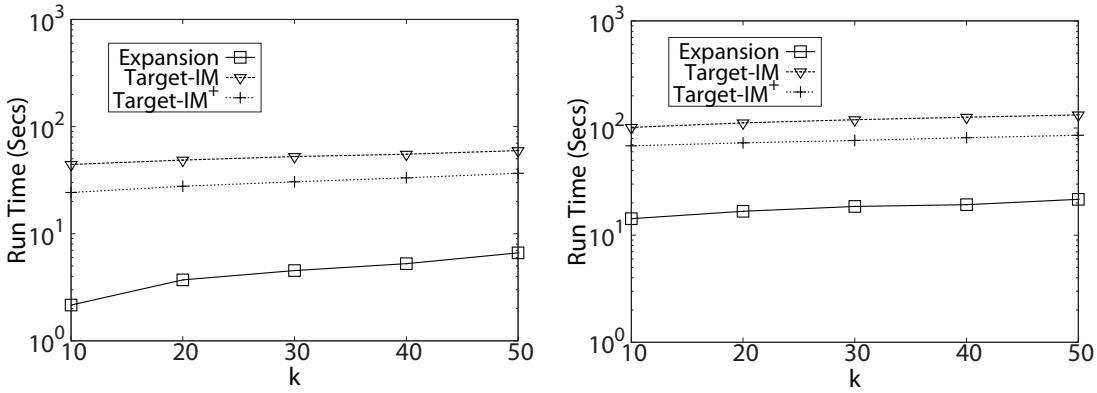
Next, we set $k = 50$ and vary the deadline α from 5 to 15. Figure 3.4 shows the results. We observe that as α increases from 5 to 15, the number of registered users increases for all of the methods. However, Target-IM and Target-IM$^+$ consistently outperform the other baseline algorithms. Expansion shows the highest rate of increase in terms of the number of *registered* users when α increases. This is because Expansion finds the nodes that can *register* a large number of users if there is no deadline for the event. As α increases, the results given by Expansion slowly converges to the optimal result. When α is very small, Expansion's performance in the number of registered users is only slightly better than IMM, which considers neither location nor deadline.



(a) Gowalla **(b)** Twitter

(c) Weibo **(d)** Foursquare

Figure 3.4: Effect of α ($k = 50$).



(a) Weibo **(b)** Foursquare

Figure 3.5: Runtime when $login(v) = 1$ for all v and $\alpha = \infty$.

3.3.2 Experiments with Location only

In this set of experiments, we compare the performance of Target-IM and Target-IM$^+$ with Expansion. We set the login probability of all nodes to be 1 and deadline $\alpha = \infty$ while $f(\gamma, l_v)$ is set according to the distance between l_v and γ. In other words, the targeted influence maximization problem reduces to the location-aware influence maximization problem [LCF+14].

Figure 3.5 shows the runtime when we vary k from 10 to 50 on the two larger datasets Weibo and Foursquare. We observe that Expansion has the fastest execution time as it utilizes heuristics to avoid paths with low influence probability. On the other hand, Target-IM and Target-IM$^+$ consider all possible paths from u to v to estimate each node's influence more accurately, leading to a higher runtime. However, Expansion cannot guarantee any bound for the returned solution, while Target-IM has a guaranteed bound.

Figure 3.6 shows the number of registered users on all four datasets. All three methods give comparable results, although Target-IM and Target-IM$^+$ win Expansion by a small margin. Expansion considers that the best chance for v to be influenced by u through the path from u to v with the highest influence probability and ignores all other paths from u to v. As a result, Expansion may under-estimate the actual influence of a node.

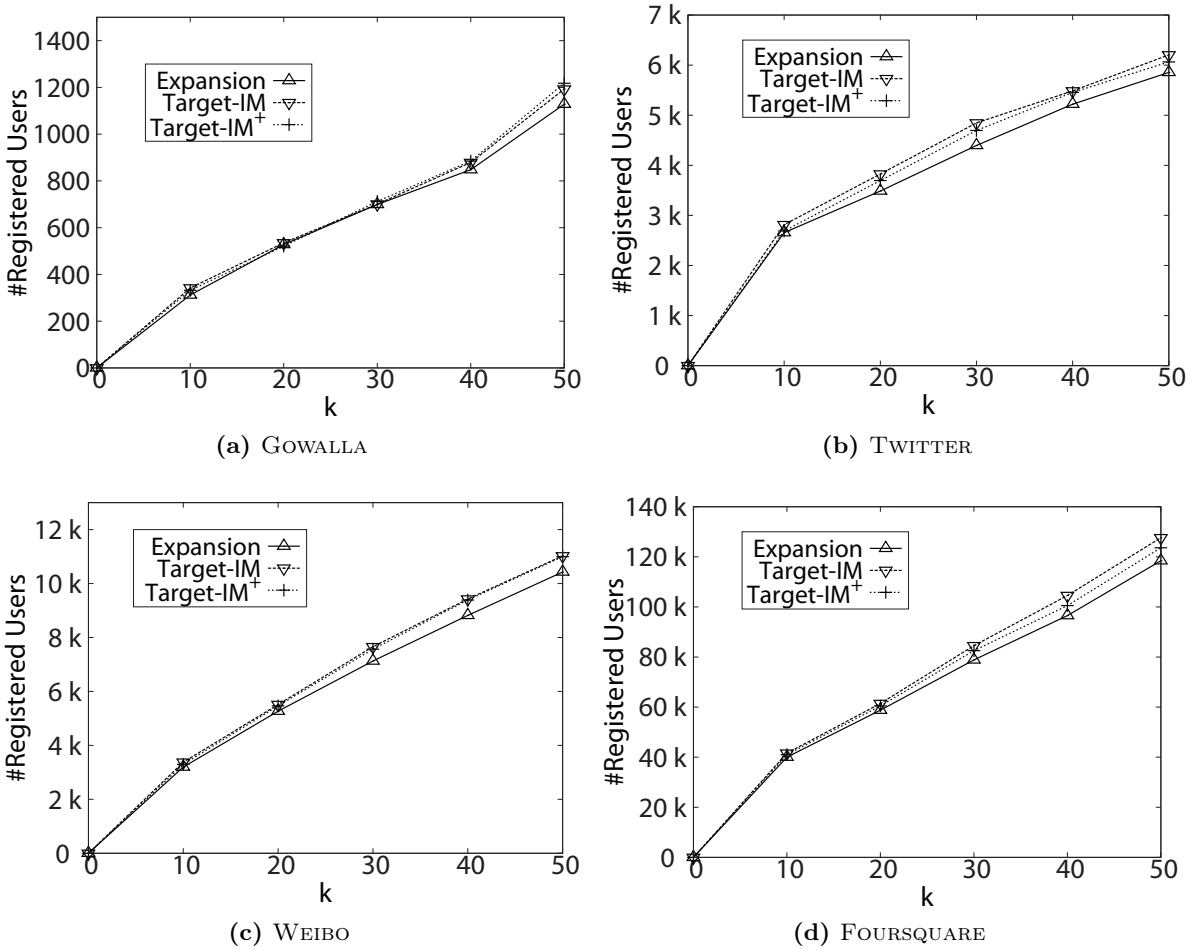
3.3.3 Experiments with Deadline only

Finally, we compare Target-IM and Target-IM$^+$ with MIA-L in terms of both effectiveness and efficiency without considering the deadline. We set $f(\gamma, l_v) = 1$ for all v, in other words, all of the users will register for an event regardless of its location as long as he is influenced. This setting reduces the targeted influence maximization problem to the deadline-aware influence maximization problem proposed in [CLZ12].

Figure 3.7 shows the runtime on the two larger datasets WEIBO and FOURSQUARE when we vary α from 5 to 15 . We see Target-IM and Target-IM$^+$ are clearly faster than MIA-L. This is because MIA-L utilizes the maximum influence in-arborecence structure to record the high influence paths. When there are multiple parent nodes pointing to the same child node in a maximum influence in-arborecence, the MIA-L algorithm needs to enumerate all possible orders of these parent nodes since different ordering of login events will result in different influence spread for each parent node. When there are many nodes that directly point to the same child node, MIA-L will incur a large amount of computation to estimate each parent node's influence on this child node.

Note that Target-IM$^+$ runs faster than Target-IM because Target-IM$^+$ avoids the computation for a node once we realize its probability of registering the root is below some threshold. When α increases, Target-IM still needs to traverse until α levels to construct each WRR tree whereas for Target-IM$^+$, we stop the breadth-first traversal as soon as we reach a node with low influence probability. Hence when α increases, Target-IM's running time increases more significantly than Target-IM$^+$'s as shown in Figure 3.7.

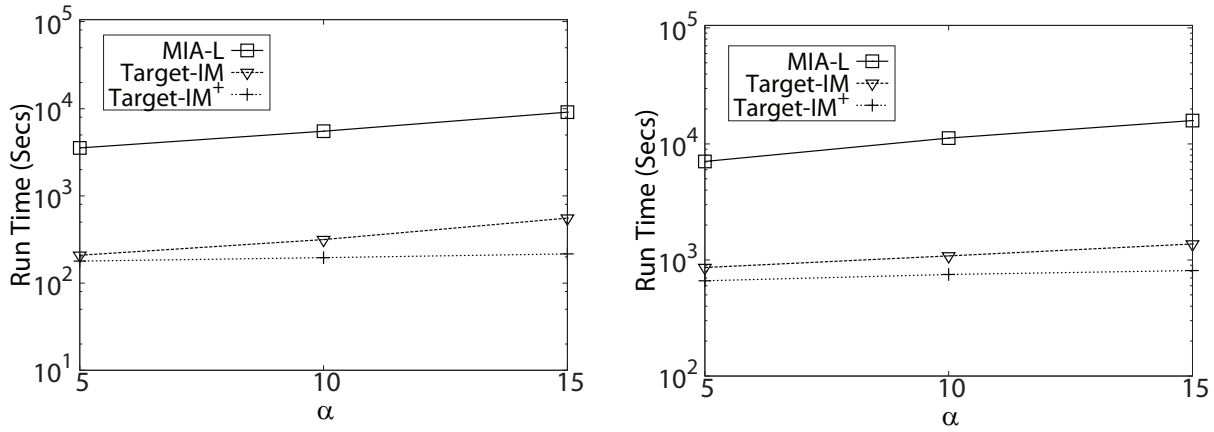
Figure 3.8 shows the number of registered users on all the four datasets. We observe that even though MIA-L considers deadline information, its performance is not as good as our proposed methods. This is because the path with the highest influence probability may not be the optimal path that u influences v within deadline because the nodes on the path may have low login probabilities. As a result, the set of nodes returned by MIA-L algorithm may not influence the largest number of users within deadline as compared to our proposed Target-IM and Target-IM$^+$.



(a) Gowalla **(b)** Twitter

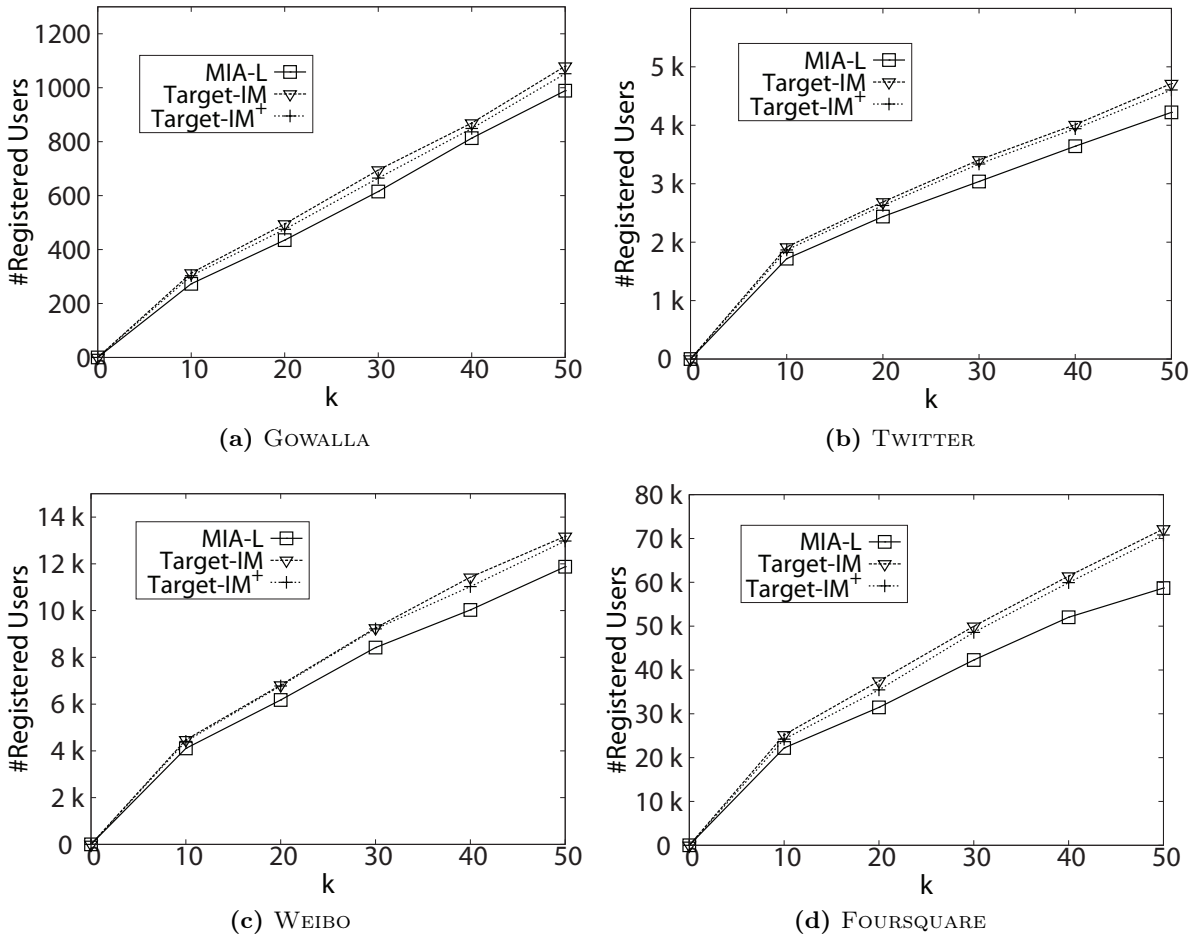
(c) Weibo **(d)** Foursquare

Figure 3.6: Performance when $login(v) = 1$ for all v and $\alpha = \infty$.



(a) Weibo **(b)** Foursquare

Figure 3.7: Runtime when $f(\gamma, l_v) = 1$ for all v ($k = 50$).



(a) Gowalla **(b)** Twitter

(c) Weibo **(d)** Foursquare

Figure 3.8: Performance when $f(\gamma, l_v) = 1$ for all v ($\alpha = 10$).

3.4 Summary

In this chapter, we have examined the problem of targeted influence maximization in social networks, taking into account the temporal and geographical constraints. We introduced the notion of WRR trees and designed an algorithm called Target-IM that generates a set of WRR trees. We provided a way to estimate the number of WRR trees required in order to return a solution with $(1 - 1/e - \varepsilon)$ approximation bound. With this, we greedily selects k nodes that covers the largest number of WRR trees. We further improved the WRR tree generation algorithm by omitting nodes that are unlikely to influence the target region. Extensive experimental studies have been conducted on four datasets to demonstrate the effectiveness and efficiency of Target-IM and Target-IM$^+$.

Chapter 4

Identifying Brokers in Dynamic Social Networks

The users that are influenced in the *Targeted Influence Maximization* problem are within a specific region and are often likely to be within the same community. Sometimes we wish to propagate some information such that many communities in the network become aware of the information rather than influencing all the users in the same community. In this chapter, we navigate to address the second challenge in spreading information, *i.e.*finding the nodes that play significant roles in diffusing information to remote users. We introduce our concept of *brokers* and present our study on finding brokers in dynamic social networks [SHL15a].

4.1 Problem Definition and Analysis

We define a broker as one whose removal would result in the greatest increase in the pairwise distance among the remaining users. The intuition behind this definition is to capture users who are situated between otherwise disconnected or distant users. Formally, we have

Definition 4. ***Top-k Brokers.*** *Let $G = (V, E)$ be a directed graph where each node $v \in V$ denotes a user, and each edge $\langle u, v\rangle \in E$ denotes that user u follows user v. The top-k brokers are the set of users $H \subset V$, $|H| = k$, such that the utility function $\mathcal{D}$ is maximized:*

$$\mathcal{D} = \sum_{u,v \in V-H} \frac{1}{d(u,v)} - \sum_{u,v \in V-H} \frac{1}{d'(u,v)}, \tag{4.1}$$

where $d(u, v)$ and $d'(u, v)$ denote the distance between u and v in G and $G \backslash H$ respectively. $G \backslash H$ is the resultant subgraph of G having the nodes in H and their incident edges removed.

Next, we prove that the problem of finding top-k brokers is NP-hard.

Theorem 3. *Identifying the top-k brokers in a directed graph is NP-hard.*

Proof. The decision version of the top-k broker problem is stated as follow: Is there a subset of nodes H where $|H| = k$ such that the utility function $\mathcal{D}$ in Equation 4.1 is greater than a given value s? To prove that the top-k broker problem is NP-hard, we first show that

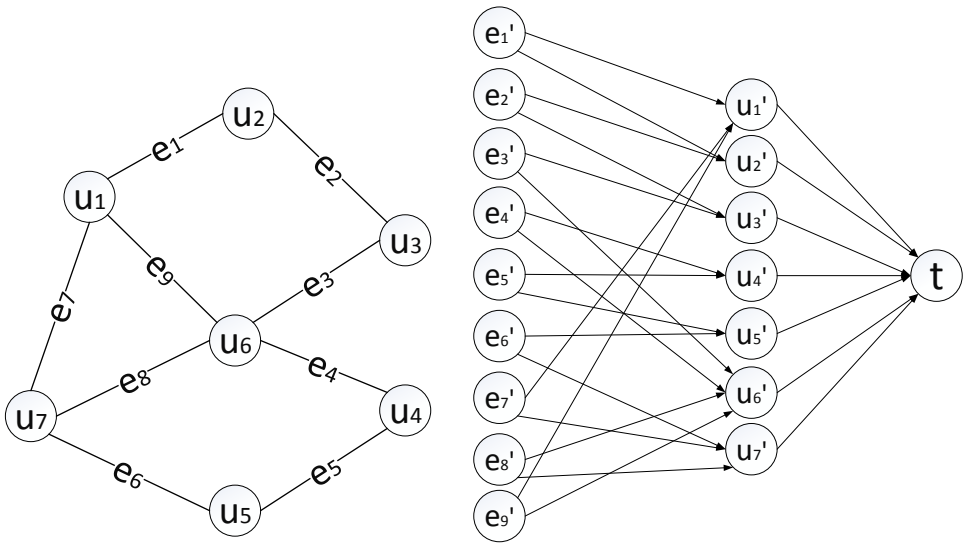
the decision version of top-k broker can be reduced from its function version in polynomial time, and then we prove the decision version of top-k broker problem is NP-hard.

Suppose we have a graph with n nodes, the maximum distance between any pair of nodes is $(n-1)$ and there can be at most n^2 pairs of nodes. This implies that there are at most $n^2(n-1)$ possible values for $\mathcal{D}$. If we can solve the decision version in polynomial time, then the solution to the top-k broker problem is the set of nodes H that gives the largest s value which can be found in polynomial time.

Next, we prove that the top-k broker decision problem is NP-hard by showing that it can be reduced from the known NP-hard *k-densest subgraph* problem, namely, determining if there exists a k-node subgraph with at least p edges in a given graph [AHI02].

Given an instance $I = \{G = (V, E), k, p\}$ of the k-densest subgraph problem, let $n = |V|$, and $m = |E|$. We can construct a directed graph $G' = (V', E')$ of $(n + m + 1)$ nodes and $(n + 2m)$ edges as follows. We first create a terminal node t in G'. For each node u in G, we create a corresponding node u' in G', and connect u' to t. For each edge $e(u, v)$ in G, let u' and v' in G' be the corresponding nodes of u and v respectively. We create a node e' in G', and connect e' to u' and v'. Let $Y_1 = \{e'_1, \cdots, e'_m\}$ and $Y_2 = \{u'_1, \cdots, u'_n\}$. Then $V' = Y_1 \cup Y_2 \cup \{t\}$.

Figure 4.1 shows an example graph G and the corresponding constructed graph G'. It is clear that the construction of G' can be done in polynomial time.



(a) Original Graph G **(b)** Converted Graph G'

Figure 4.1: Construction of G' from G.

Next, we prove that the k-densest subgraph instance I is satisfiable, *if and only if* there exists a subset $H \subseteq Y_2$ and $|H| = k$ such that $\mathcal{D} \geq \frac{p}{2}$ in G'.

For the *only if* direction, suppose the k-densest subgraph instance I is satisfiable, that is, we have a subgraph with k nodes and at least p edges in G. Let $H \subseteq Y_2$ be the set of nodes in G' that correspond to these k nodes. When we remove H from G', there are at least $p \in Y_1$ nodes in G' that become isolated. These isolated nodes correspond to the p edges in the k-densest subgraph solution. Recall the utility function in Equation 4.1 which computes the difference in the sum of the inverse pairwise distance between G' and $G' \backslash H$. From the graph G' in Figure 5.1(b), we note that

$$d(u,v) = \begin{cases} 1 & u \in Y_1 \text{ and } v \in Y_2 \\ 1 & u \in Y_2 \text{ and } v = t \\ 2 & u \in Y_1 \text{ and } v = t \end{cases}.$$

Clearly, the p isolated nodes will cause a difference in the pairwise distances between the remaining nodes in G' and $G' \backslash H$. This is because these p nodes are able to reach the terminal node t in G' in 2 hops, but they can no longer reach t in $G' \backslash H$. Hence,

$$\begin{aligned} \mathcal{D} &= \sum_{u,v \in V' - H} \frac{1}{d(u,v)} - \sum_{u,v \in V' - H} \frac{1}{d'(u,v)} \\ &= \sum_{u \in \{isolated\ nodes\}} \frac{1}{d(u,t)} \\ &\geq \frac{p}{2}. \end{aligned}$$

For the *if* direction, suppose the top-k broker decision instance is satisfiable, that is, there exists $H \subset V'$ such that $\mathcal{D} \geq \frac{p}{2}$. As mentioned above, the utility function is affected when nodes that are previously reachable in G' are no longer reachable in $G' \backslash H$. This occurs when nodes in Y_2 are removed, leading to isolated nodes in Y_1. In this case, the number of nodes removed from Y_2 is $|H \cap Y_2| \leq k$. Since $\mathcal{D} \geq \frac{p}{2}$, we have at least p isolated nodes in Y_1. These p nodes are edges in G whose end points correspond to the nodes in $H \cap Y_2$. Thus, we have found a subgraph with at most k nodes and at least p edges. In other words, the k-densest subgraph instance I is satisfiable. □

Given that the problem of detecting top-k brokers is NP-hard, we propose a solution that is based on the weak tie theory [Gra83].

4.2 Proposed Approach

Granovetter [Gra83] first introduced the concept of weak ties in social networks where the relationship between two users is weak when their number of overlapping friends is small. Bakshy et al. [BRMA12] showed that weak ties are important in the diffusion of novel information between remote users in Facebook while Burt [Bur07] found that users with many weak ties are more likely to be placed in bridging positions.

We carry out preliminary experiments on the real world DBLP dataset to investigate the correlation between weak ties and brokers. Here, the authors are the nodes while co-authorships form the edges. The tie strength of an edge $\langle author1, author2\rangle$ is given by the fraction of the number of common co-authors between $author1$ and $author2$ to the total number of co-authors they have. An edge is a weak tie if the tie strength is below some threshold. We consider authors who have served as PC members in only one research area as opinion leaders, and those who have served in multiple research areas as brokers.

We compute the number of weak ties that are incident to each author and sort the authors according to the number of weak ties. Figure 4.2 shows the percentage of opinion leaders and brokers with respect to the different ranges of weak ties. We observe that as the number of weak ties increases, the proportion of brokers increases significantly. This motivates us to develop an algorithm to find brokers based on the their number of incident weak ties.

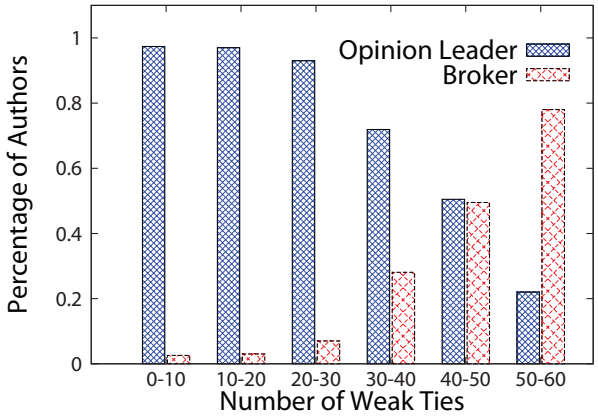


Figure 4.2: Percentage of opinion leaders and brokers versus weak ties in DBLP.

Although the experiment results show that a broker is correlated with the number of weak ties, simply counting the number of weak ties is not a good indicator of a node being a broker. Consider Figure 4.3 where a bold line denotes that two users have many overlapping friends, i.e. they have a *strong tie*. u_1 has a total of 4 weak ties while u_2 has 5. We observe that even though u_2 has more incident weak ties than u_1, u_1 is connected to more separate groups and is more likely to be a broker. A closer observation reveals that many of u_2's neighbours are, in fact, friends of each other's friends. Thus we need to take into account the connectedness among friends' friends.

We first define the tie strength in a social network as follows:

Definition 5. ***Tie Strength.*** *Let $G = (V, E)$ be a social network. For any edge $\langle u, v\rangle \in E$, u is a follower of v and v is a followee of u. Let F_u denote the followees of u inclusive of u, and F_v denote the followees of v inclusive of v. Then the tie strength of the edge $\langle u, v\rangle \in E$ is given by*

$$S_{uv} = \frac{\mid F_u \cap F_v \mid}{\mid F_u \cup F_v \mid}.$$

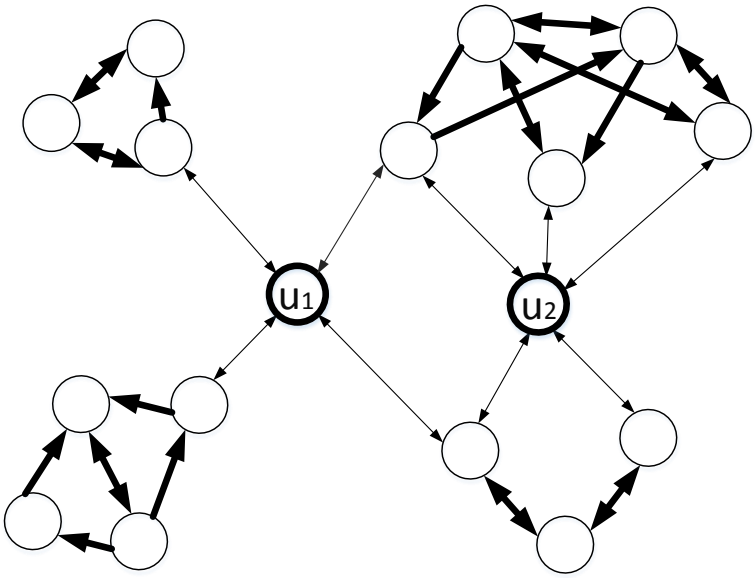


Figure 4.3: Example of two users with weak ties.

This implies that if the majority of u and v's followees are the same, S_{uv} will be close to 1. The edge from u to v is a weak tie if S_{uv} is below some threshold τ. Otherwise, it is a strong tie. Note that $S_{uv} = 0$ if u is not a follower of v.

A path from u to v is a sequence of directed edges that starts from u and ends at v. The strength of a path (*path strength*) is defined as the minimum tie strength among all the edges in the path. Now we define a strongly connected group as follows:

Definition 6. ***Strongly Connected Group.*** *Given a social network $G = (V, E)$ and a tie threshold τ, a strongly connected group is a maximal subgraph of G such that for all pairs of nodes (u, v) in the subgraph, there exists a path from u to v with path strength greater than τ.*

Note that our strongly connected groups are in fact strongly connected components with the additional constraint on the minimum path strength. We denote the set of strongly connected groups as C. Based on Definition 6, when two nodes u and v are in the same strongly connected group, there exists a path from u to v with path strength greater than τ and vice versa. This implies that all the nodes in the same strongly connected group form a cycle. Hence, the order of processing is irrelevant as we will still find the same set of strongly connected groups.

We define the *closeness* of a strongly connected group c to a node u as follows: Let N be the number of users in c whose tie strength to u is non-zero.

$$closeness(c, u) = \frac{\sum_{v \in c} S_{vu} - N \times \tau}{|c|}. \tag{4.2}$$

If $closeness(c, u) < 0$, this implies that u does not share many common followees with the users in c. In other words, u is more likely to be a broker connecting c to the rest of the network.

With this, we can assess the potential of a node u being a broker as follows:

$$score(u) = \begin{cases} \sum_{c \in \mathcal{C}} (-closeness(c, u)) & \mathcal{C} \neq \emptyset \\ 0 & otherwise \end{cases}, \tag{4.3}$$

where $\mathcal{C} \subset C$ denotes the set of strongly connected groups whose closeness with u is below 0. A large value of $score(u)$ indicates that u is connected to many largely non-overlapping groups. Hence, u is in a bridging position among these groups and plays an important role in bringing the users of these groups close to each other.

Given a social network $G = (V, E)$ and threshold τ, our proposed approach to find the top-k brokers in G consists of the following main steps:

1. For each directed edge $\langle u, v \rangle \in E$, compute tie strength S_{uv};
2. Find the set of strongly connected groups C in G;
3. For each node $u \in V$, compute its $score(u)$;
4. Return the top-k nodes with the highest scores.

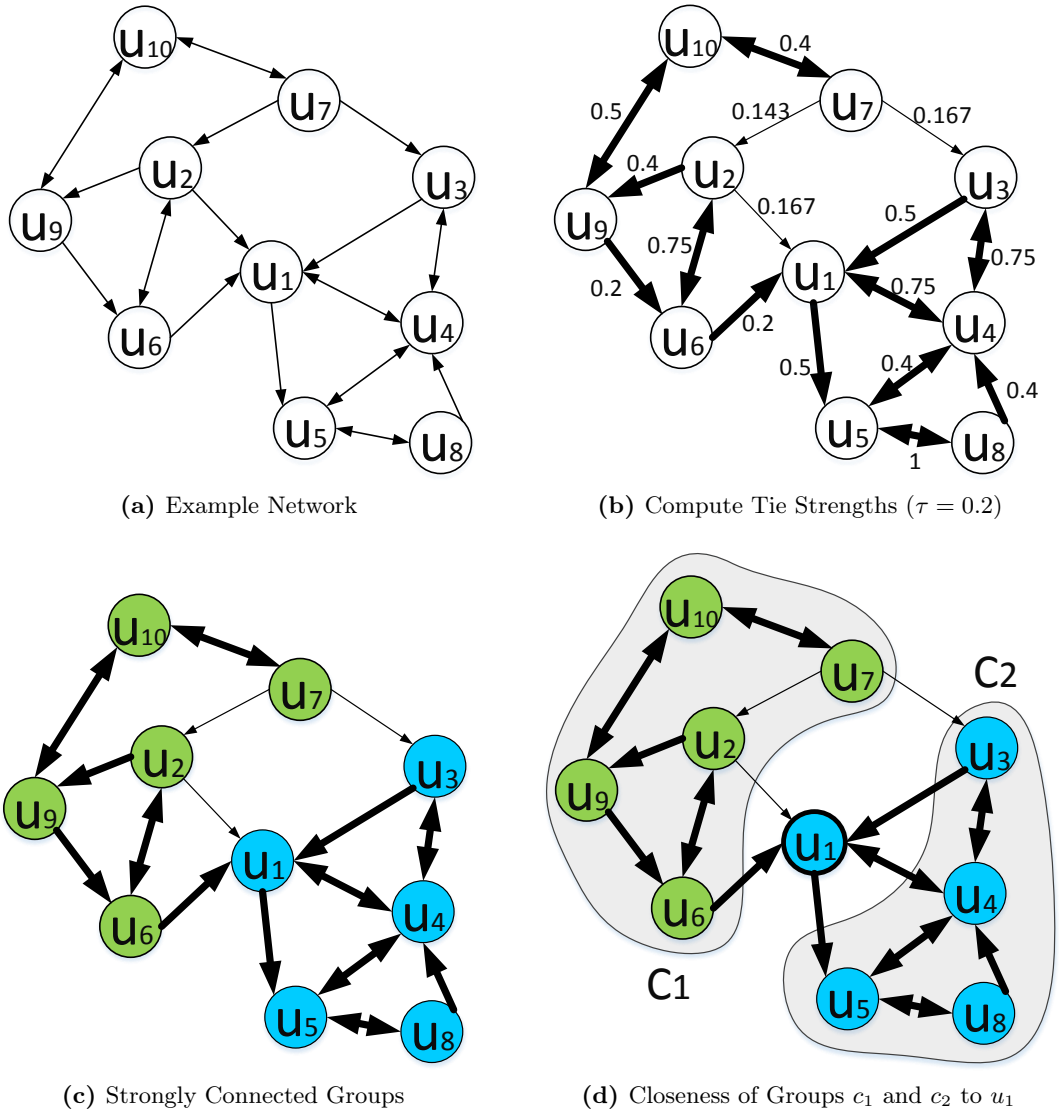
Step 2 utilizes the Tarjan's method [Tar72], which is designed for detecting strongly connected *components*, to perform a depth-first traversal of the strong ties in G to obtain the set of strongly connected *groups*.

Figure 4.4 illustrates the proposed approach using a small network with 10 nodes. We first compute the tie strengths for all the edges. For example, the tie strength for edge $\langle u_1, u_5 \rangle$ is 0.5 since we have $F_{u_1} = \{u_1, u_4, u_5\}$ while $F_{u_5} = \{u_4, u_5, u_8\}$, with u_4 and u_5 being the common followees out of 4. Figure 4.4b shows the weak and strong (bold arrows) ties in the example network when $\tau = 0.2$.

Next, we find all the strongly connected groups by applying Tarjan's algorithm [Tar72] to traverse only the strong ties. Figure 4.4c shows two strongly connected groups obtained (marked in different colors). With this, we can compute the scores of all the nodes in the network. For example, node u_1 is connected to two strongly connected groups, c_1 and c_2 (see Figure 4.4d). We use Equation 4.2 to compute the closeness of each group to u_1 as follows:

$$\begin{aligned} closeness(c_1, u_1) &= \frac{(S_{u_2u_1} + S_{u_6u_1} - 2 \times \tau)}{2} = -0.033, \\ closeness(c_2, u_1) &= \frac{(S_{u_3u_1} + S_{u_4u_1} - 2 \times \tau)}{2} = 1.1. \end{aligned}$$

Based on the above, we conclude that c_1 is not close to u_1 since the closeness is less than 0, while c_2 is close to u_1. Finally, we have score$(u_1) = 0.033$.



(a) Example Network

(b) Compute Tie Strengths ($\tau = 0.2$)

(c) Strongly Connected Groups

(d) Closeness of Groups c_1 and c_2 to u_1

Figure 4.4: Illustration of WeakTie algorithm (best viewed in color).

4.2.1 Incremental Methods

Real world social networks are highly dynamic. In an evolving social network, there are many users joining (add node) and leaving (remove node) each day. Existing users may start new relationships (add edge) or they may break off their existing relationships (remove edge). These changes in relationships can all potentially affect the set of top-k brokers. Figure

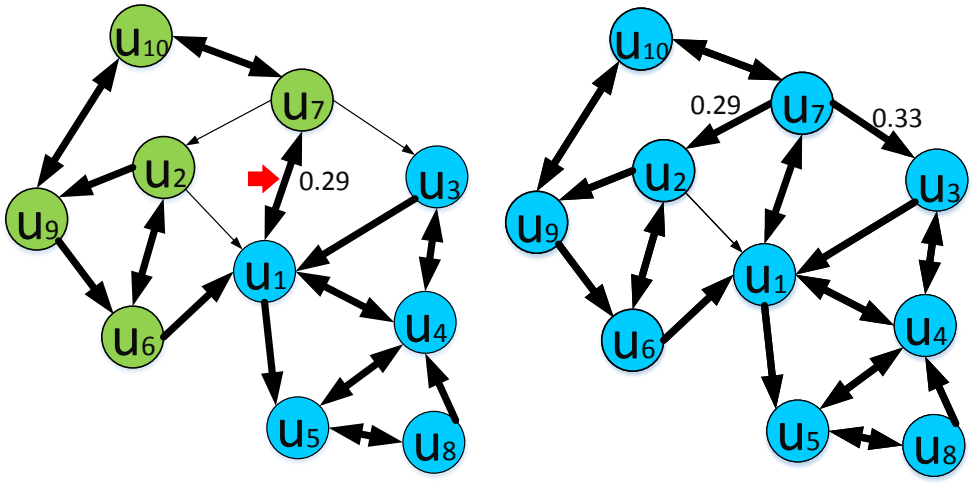


Figure 4.5: Effect of Adding Edges (best viewed in color).

4.5 shows the changes in the network after u_1 and u_7 start following each other. Note that when edges $\langle u_1, u_7 \rangle$ and $\langle u_7, u_1 \rangle$ are added, their tie strength is 0.33 and hence they are strongly connected to each other. Additionally, the tie strengths between their common followers increase and the edges $\langle u_7, u_2 \rangle$ and $\langle u_7, u_3 \rangle$ now become strong ties (see Figure 4.5b). Therefore, the entire network forms a single strongly connected group.

Recomputing the set of new top-k brokers when each update occurs is computationally expensive. Each update requires reapplying the Tarjan's algorithm on the entire graph to obtain the strongly connected groups. This has a complexity of $O(|V| + |E|)$ and is clearly not a practical solution for online social networks where the volume of updates is high. This motivates us to propose incremental algorithms to handle the dynamic nature of social networks.

Algorithm WeakTie-Local

Updates in social networks are handled as either addition of new edges or removal of existing edges. This is because adding a node can be modelled as the addition of a list of edges between the new node and existing nodes; while removing an existing node is modelled as the removal of the edges associated with the removed node. Empirical investigation reveals

changes to the strongly connected groups are often restricted to the 2-hop neighborhood of the affected nodes. This is because it is rarely the case that two users are connected by a chain of close friends and they do not know each other [Kat73]. We design WeakTie-Local algorithm that utilizes the 2-hop neighborhood of an affected node to identify brokers.

Algorithm 4.1 gives the details. Suppose an edge $\langle u, v\rangle$ is added, Algorithm WeakTie-Local first creates the respective nodes if they do not exist and updates the tie strength S_{uv}. Since F_u has changed because of the addition/deletion of $\langle u, v\rangle$, for any neighbor w of u, we update S_{uw} or S_{wu} according to Definition 5 (Lines 14-15).

Algorithm 4.1: WeakTie-Local

input : 1. Social network $G = (V, E)$
2. Set of edges to be added E^+
3. Set of edges to be removed E^-
4. Tie strength threshold τ

output : Top-k Brokerss

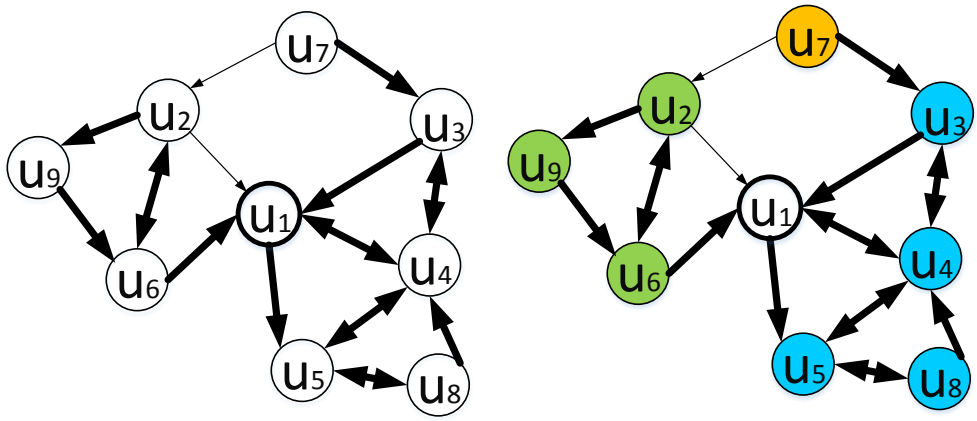
1 **foreach** $\langle u, v\rangle \in E^+ \cup E^-$ **do**
2 **if** $\langle u, v\rangle \in E^+$ **then**
3 Create node u if $u \notin V$;
4 Create node v if $v \notin V$;
5 Compute S_{uv};
6 **end**
7 **foreach** *neighbor* w *of* u **do**
8 Compute S_{wu} or S_{uw};
9 **end**
10 Extract u, v's 2-hop neighborhood G_u, G_v;
11 $score(u) = UpdateScore(u, G_u)$;
12 $score(v) = UpdateScore(v, G_v)$;
13 **foreach** *neighbor* w *of* u **do**
14 **if** w *is neighbor of* v **then**
15 Extract w's 2-hop neighborhood G_w;
16 $score(w) = UpdateScore(w, G_w)$;
17 **else**
18 Compute $closeness(c, w)$for $u \in c \wedge c \in C_w$ with Equation 4.2;
19 Compute $score(w)$ with Equation 5.3;
20 **end**
21 **end**
22 Return top-k nodes with highest scores.

Function Updatescore(u, G)

```
1  Call Tarjan's algorithm to find the set of strongly connected groups C_u in G.
2  score(u) = 0;
3  foreach c ∈ C_u do
4  |   Compute closeness(c, u) with Equation 4.2;
5  |   if closeness(c, u) < 0 then
6  |   |   score(u) = score(u) − closeness(c, u);
7  |   end
8  end
9  Return score(u);
```

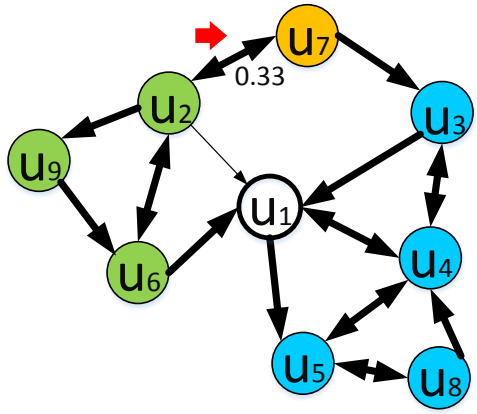
Let G_u be the 2-hop neighborhood of a node u and C_u be the set of strongly connected groups of nodes in G_u. From henceforth, we call C_u the local groups of u. We extract the 2-hop neighborhoods G_u and G_v. The scores of u and v are recomputed by calling the function *UpdateScore()* (Lines 1-8). For u's neighbor w, we need to recompute the $closeness(c, w)$ for $u \in c \wedge c \in C_w$ and update $score(w)$ (Lines 24-25). Further, adding the edge $\langle u, v \rangle$ may affect the local groups of the *common neighbors* of u and v. Hence, we also update the scores of these nodes (Lines 20-22). Similarly, when an edge is deleted, the corresponding 2-hop neighborhoods are extracted, and we recompute the scores of the affected nodes.

Figure 4.6 illustrates the WeakTie-Local algorithm. Figure 4.6a is the 2-hop neighborhood of node u_1 and the tie strengths of the edges have been computed. We find all the local groups by applying Tarjan's algorithm [Tar72] on the nodes in G_{u_1} considering the ties among them. Figure 4.6b shows the 3 local groups for u_1 and we compute the score of u_1. When the edge $\langle u_2, u_7 \rangle$ is added (Figure 4.6c), u_2 has gained a new followee. This changes the tie strengths $S_{u_7u_2}$, $S_{u_2u_1}$, $S_{u_2u_6}$ and $S_{u_2u_9}$ and edge $\langle u_7, u_2 \rangle$ now becomes a strong tie (Figure 4.6d). We then apply the Tarjan's algorithm to obtain the new set of local groups of u_1 and compute the updated score of u_1.

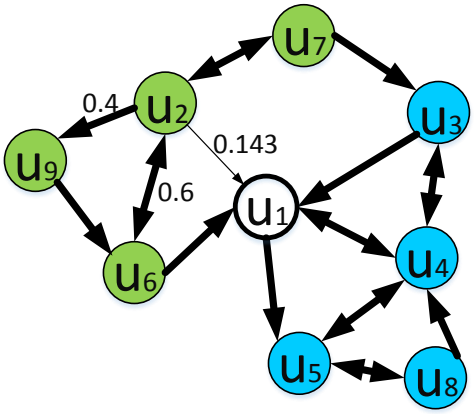


(a) 2-hop Neighborhood of u_1

(b) Local Groups of u_1



(c) Add Edge $\langle u_2, u_7 \rangle$



(d) New Local Groups of u_1

Figure 4.6: Illustration of WeakTie-Local algorithm (best viewed in color).

Algorithm WeakTie-Bi

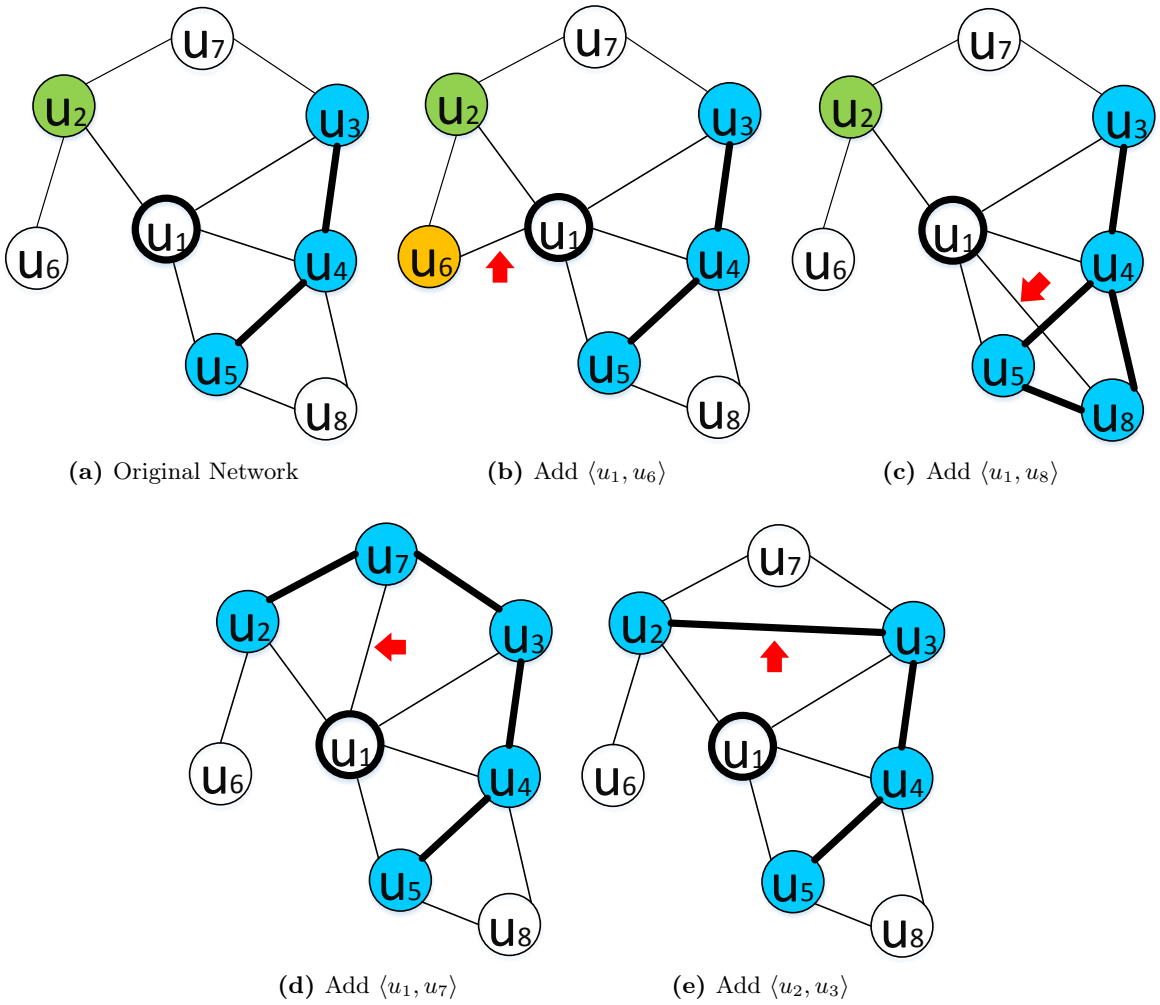
For bidirectional social networks, we do not need to perform Tarjan's algorithm to find the new set of strongly connected groups whenever updates come. In most cases, as we will show soon, updating the score of an affected node takes linear time to the number of neighbors. The only case when we need to traverse the neighborhood is when the deleted edge causes a group to split into smaller groups. We present WeakTie-Bi algorithm to carefully exam the different cases in order to further accelerate broker detection in bidirectional networks.

Algorithm 4.2 gives the details. Lines 2-8 process the insertion of new edges. Given an edge $\langle p, q \rangle$, if either p or q does not exist, we create a new node (Line 3). Lines 4-7 update the friend lists of p and q as well as their tie strengths. Line 8 calls Algorithm 4.3 to handle the various cases for the addition of edges. Lines 10-13 process the deletion of edges. The updating of friend lists is done in Line 10 and the tie strengths are recomputed in lines 11 and 12. Line 13 calls Algorithm 4.4 to handle cases for the deletion of edges. The scores of

the affected nodes are updated in lines 14-18. Finally, we return the top k nodes with the highest scores.

Algorithm 4.2: WeakTie-Bi

```
    input  : 1. Social network G = (V, E)
             2. Set of edges to be added E+
             3. Set of edges to be removed E−
             4. Tie strength threshold τ
    output : Top-k Brokers
 1  foreach ⟨p, q⟩ ∈ E+ ∪ E− do
 2      if ⟨p, q⟩ ∈ E+ then
 3          Create nodes p and q if p, q ∉ V;
 4          F_p = {q} ∪ F_p; F_q = {p} ∪ F_q;
 5          Compute S_pq;
 6          foreach w ∈ F_p ∪ F_q do
 7              Compute S_pw and S_qw;
 8          end
 9          Call Algorithm AddCases (p, q);
10      end
11      else
12          F_p = F_p − {q}; F_q = F_q − {p};
13          foreach w ∈ F_p ∪ F_q do
14              Compute S_pw and S_qw;
15          end
16          Call Algorithm RemoveCases(p, q);
17      end
18      Update closeness between p and q and their local groups;
19      Compute score(p) and score(q);
20      foreach w ∈ F_p ∪ F_q do
21          Update closeness between w and its local groups;
22          Compute score(w);
23      end
24  end
25  Return top-k nodes with highest scores.
```



(a) Original Network **(b)** Add $\langle u_1, u_6 \rangle$ **(c)** Add $\langle u_1, u_8 \rangle$

(d) Add $\langle u_1, u_7 \rangle$ **(e)** Add $\langle u_2, u_3 \rangle$

Figure 4.7: Illustration of cases A1, A2 and A3 (best viewed in color). The new yellow group is formed in (b). The blue group is strengthened in (c). Blue group merge with green group in (d) and (e).

We enumerate the cases that may arise due to the insertion of an edge. The cases are illustrated using the network in Figure 4.7a.

A1. New group formed.

This corresponds to the case when the edge $\langle u_1, u_6 \rangle$ is added. Although u_6 was already a friend of u_2 prior to becoming friend of u_1, the tie between u_6 and u_2 is weak. Hence, a new group (colored in yellow) is formed (see Figure 4.7b). Lines 11-13 in Algorithm 4.3 deal with this case.

A2. Strengthen group.

Here, $\langle u_1, u_8 \rangle$ has been added and u_8 is strongly connected with multiple friends of u_1 where all these friends are in the same group in blue (see Figure 4.7c). In this case, we

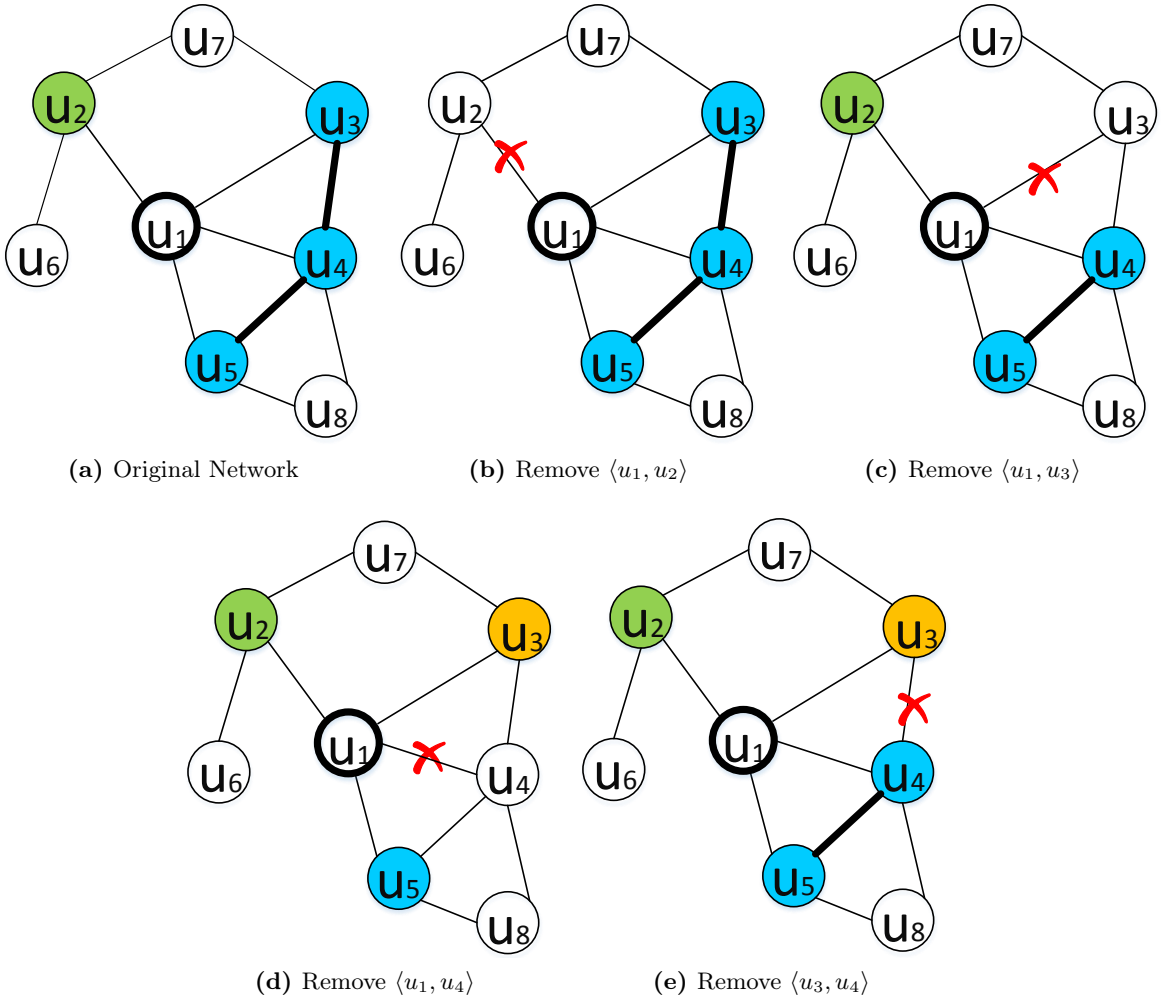
simply include u_8 into the blue group and update the corresponding *closeness*. Lines 14 and 15 in Algorithm 4.3 handle this case.

A3. Merge groups.

This case corresponds to the addition of edge $\langle u_1, u_7 \rangle$ or $\langle u_2, u_3 \rangle$ (see Figure 4.7d and 4.7e). After adding the edge, u_1's local green group is now merged with the blue group to form the resultant group c_{new}. We compute $closeness(c_{new}, u_1)$. This case is handled in lines 6-10 and 16-19 of Algorithm 4.3.

Algorithm 4.3: AddCases (e)

```
   input  : Edge e = (m,n)
   output : Updated local groups
1  Let C' = ∅;
2  foreach w ∈ F_m ∩ F_n do
3      Let c_w denote the local group that contains w;
4      if S_nw > τ then
5          C' = C' ∪ {c_w}
6      end
7      if S_mn > τ then
8          Let c_1, c_2 ∈ C_w be the local groups of m and n respectively;
9          if c_1 ≠ c_2 then
10             c_new = c_1 ∪ c_2;
11             C_w = (C_w − {c_1, c_2}) ∪ {c_new};
12         end
13     end
14 end
15 if |C'| = 0 then
16     Create new group c_new = {n};
17     C_m = C_m ∪ {c_new};
18 end
19 if |C'| = 1 and C = {c} then
20     c = {n} ∪ c;
21 end
22 if |C'| > 1 then
23     Merge all the groups in C' to form c_new;
24     c_new = c_new ∪ {n};
25     C_m = (C_m − C') ∪ {c_new}.
26 end
```



(a) Original Network **(b)** Remove $\langle u_1, u_2 \rangle$ **(c)** Remove $\langle u_1, u_3 \rangle$

(d) Remove $\langle u_1, u_4 \rangle$ **(e)** Remove $\langle u_3, u_4 \rangle$

Figure 4.8: Illustration of cases D1, D2 and D3 (best viewed in color). The green group is removed in (b). The blue group is weakened in (c). The blue group split into blue and yellow groups in (d) and (e).

Similarly, we enumerate the cases when an edge is removed.

D1. Remove group.

In this case, the edge removed is $\langle u_1, u_2 \rangle$. u_2 has no other overlapping friends with u_1 and hence the strongly connected group in green consisting of only u_2 is dropped (see Figure 4.8b). Lines 10 and 11 in Algorithm 4.4 handle this case.

D2. Weaken group.

This case corresponds to the removal of edge $\langle u_1, u_3 \rangle$. Here, u_3 has only one common neighbor with u_1. After the edge has been removed, the remaining users in the blue group are still strongly connected to each other (see Figure 4.8c). Lines 12-14 in Algorithm 4.4 deal with this case.

D3. Split group.

This case corresponds to the removal of $\langle u_1, u_4 \rangle$ or $\langle u_3, u_4 \rangle$. If $\langle u_1, u_4 \rangle$ is removed, u_3 and u_5 are no longer strongly connected, or if $\langle u_3, u_4 \rangle$ is removed, u_3 needs to form an individual group (see Figure 4.8d and 4.8e). As a result, two groups are formed and we compute the closeness of the two new groups. Lines 5-9 and 15-19 in Algorithm 4.4 handle this case.

Algorithm 4.4: RemoveCases(e)

```
   input  : Edge e = (m,n)
   output : Updated local groups
1  Let N' = ∅;
2  foreach w ∈ F_m ∩ F_n do
3      if S_nw > τ then
4          N' = N' ∪ {w}
5      end
6      if S_mn > τ then
7          Let c_1, c_2 ∈ C_w be the local groups of m and n respectively;
8          if c_1 = c_2 then
9              Regroup nodes in c to form new groups C_new;
10             C_w = (C_w − {c}) ∪ C_new;
11         end
12     end
13 end
14 if |N'| = 0 then
15     C_m = C_m − {n};
16 end
17 if |N'| = 1 then
18     Get n's local group c ∈ C_m;
19     c = c − {n};
20 end
21 if |N'| > 1 then
22     Get n's local group c;
23     c = c − {n};
24     Regroup nodes in c to form new groups C_new;
25     C_m = (C_m − {c}) ∪ C_new.
26 end
```

We now analyze the efficiency of WeakTie-Local and WeakTie-Bi algorithms. Let d denote the degree of a node, where $0 \leq d \leq |V|$. Given that real world social networks are sparse, we have $d \ll |V|$ for most of the nodes. In WeakTie-Local, when an edge $\langle u, v \rangle$ is added or deleted, it takes $O(d^2)$ to find the local groups again. Hence, the time needed to recompute the scores for u and v is $O(d^2)$. For a neighbor w of u only, it needs to recompute the *closeness* between each group and update the *score*, and this takes $O(|C_w|)$. For the common neighbors of u and v, WeakTie-Local needs to run the strongly connected group detection procedure in the neighborhood whose time complexity is $O(d^2)$ for each node. We have at most d such nodes, hence the time complexity for WeakTie-Local is $O(d^3)$. In WeakTie-Bi, when (u, v) is added, updating the scores for u and v takes $O(|C_u|)$ and $O(|C_v|)$ respectively since we need to recompute the *closeness* with each local group. Similarly for the a friend w of u or v, each friend takes $O(|C_w|)$ time to update its score. This requires a time complexity of $O(d|C_w|)$ as we have d friends. When (u, v) is deleted, both cases D1 and D2 require $O(d|C|)$ complexity. For case D3, the complexity is $O(d^3)$ as we have d nodes to apply depth-first search to find local groups again in their neighborhood.

4.3 Experiments

In this section we evaluate the effectiveness and efficiency of the weak tie based algorithms in identifying brokers. We also study how brokers can help information diffusion tasks. All experiments are run on a linux machine with 2 Xeon E5440 2.83 GHz CPU and 16G RAM.

Datasets. We use the four social network datasets described in Section 3.3 and the details of these datasets are given in Table 3.1. Brokers can also be the researchers that enable the fusion of ideas from different research areas. In this section, we also have a DBLP[1] dataset to illustrate the effectiveness of identified brokers. The DBLP dataset has 815,946 authors which form the nodes in the network. We extract a subset of authors from 4 research areas in computer science, namely Databases (DB), Information Retrieval (IR), Artificial Intelligence (AI) and Networks and Communication (NC). The resulting dataset has 219,815 authors and 1,089,793 co-authorships. The longest distance between any two authors is 20 and the average distance is 7.06.

4.3.1 Effectiveness Experiments

In this set of experiments, we evaluate the quality of the solutions given by the following methods:

- WeakTie: This method uses the entire graph to find the strongly connected groups.

[1] http://aminer.org/billboard/structural-hole.html

- WeakTie-Local: Based on Algorithm WeakTie, this method utilizes the 2-hop neighborhood of a node to find local groups.
- WeakTie-Bi: This method is based on Algorithm WeakTie-Local and is optimized for undirected graphs.
- PageRank♯: In this method, each node's follower' PageRank scores [PBMW98] are summed up, and the k nodes with the highest scores are returned.
- Betweenness [HC13]: This returns the top k nodes that has the largest number of shortest paths passing through them.

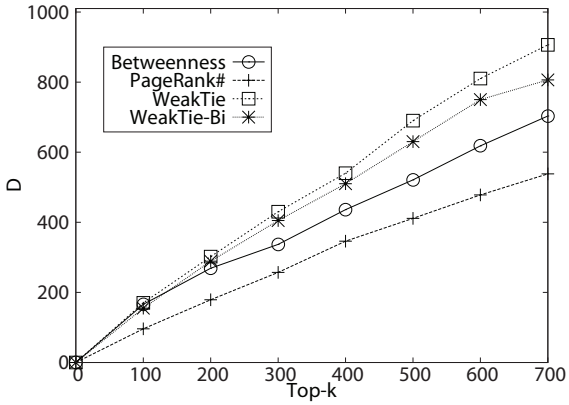
We run WeakTie, PageRank♯ and Betweenness on both DBLP dataset and social network datasets. Since DBLP and Gowalla are bidirectional networks while Twitter, Weibo and Foursquare are unidirectional networks, we apply WeakTie-Bi on DBLP and Gowalla and WeakTie-Local on Twitter, Weibo and Foursquare. We vary k from 100 to 700 and compute the utility function $\mathcal{D}$ values obtained (see Equation 4.1).

Figure 4.9 shows that WeakTie is able to obtain the highest $\mathcal{D}$ values for all k. WeakTie-Bi and WeakTie-Local suffer a slight decrease in quality as they only consider the local views of the affected nodes, and may not obtain the true picture of the connectedness among their friends. We observe that the Betweenness method performs well when k is small. This is because when we remove the nodes that have many shortest paths passing through them, the average distance between nodes will increase. However, as k increases, many of the nodes have alternate paths and the performance of the Betweennesss method decreases. Note that PageRank♯ performs worse in unidirectional networks Twitter, Weibo and Foursquare than in bi-directional networks Gowalla and DBLP. This is because, in unidirectional networks, a user with many followers tends to have high PageRank♯ score. However, this user may not have any followee. As such, he/she ends up at the end of a path, and will not contribute much to the reduction of the average distance among other users.

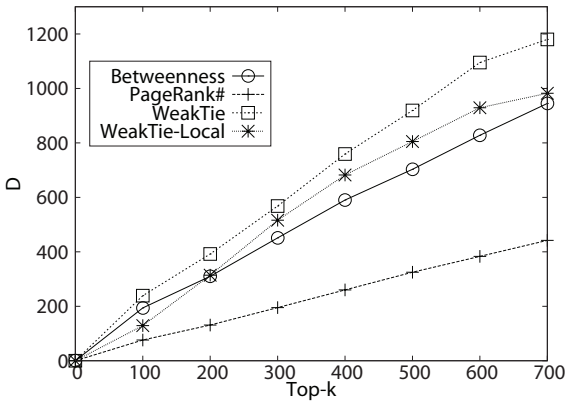
4.3.2 Sensitivity Experiments

Next, we examine the effect of τ on the performance of the proposed methods. We set $k = 500$. Figure 4.10 shows the results of varying τ on DBLP and Twitter. The results on Gowalla, Weibo and Foursquare are omitted because the observation for τ is similar to that in Twitter. We see that the performance of the proposed algorithms is dependent on τ. For the Twitter dataset, both methods perform best when $\tau = 0.125$. For the DBLP dataset, the best performance is achieved when τ is around 0.5. Clearly, τ is dependent on the overall connectedness of the users in the social network.

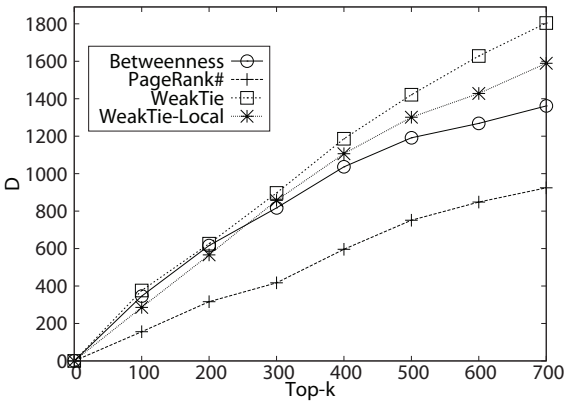
For Twitter-like networks, users tend to follow a large set of users and the proportion of overlapping friends tends to be small. As a result, a small value of τ for the social network



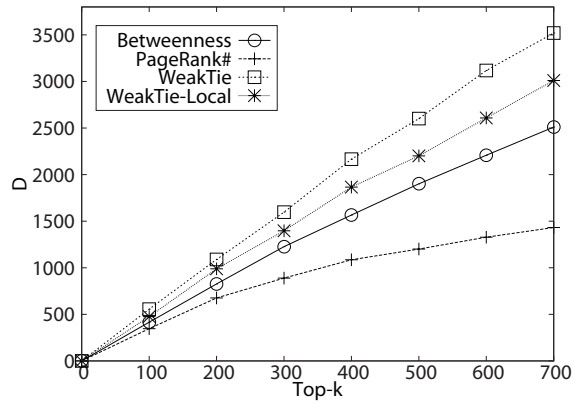
(a) Gowalla



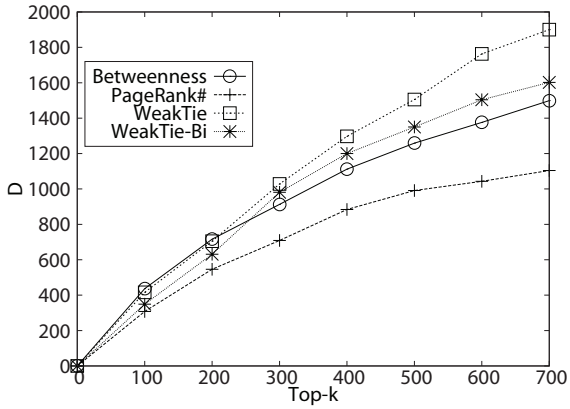
(b) Twitter



(c) Weibo

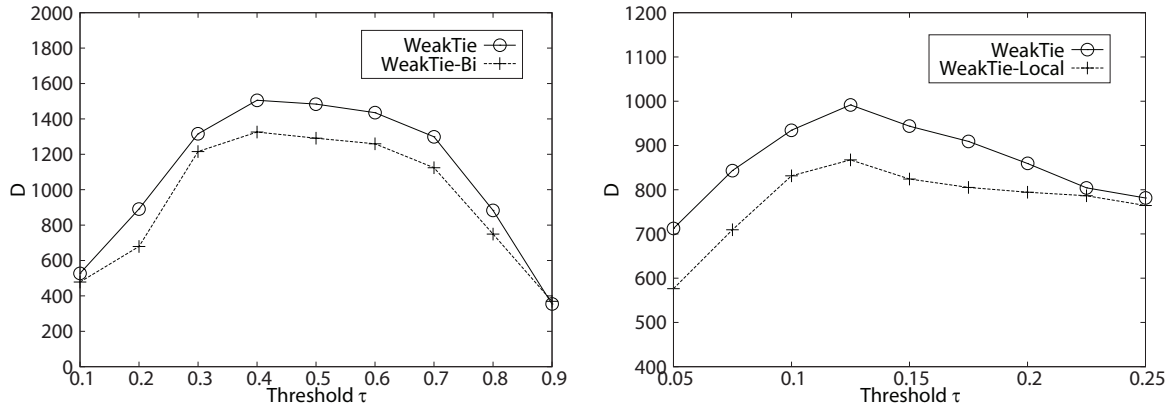


(d) Foursquare



(e) DBLP

Figure 4.9: Quality of returned solutions.



(a) DBLP **(b)** Twitter

Figure 4.10: Effect of τ.

dataset is preferred. On the other hand, in the DBLP dataset, each paper has only a small number of co-authors resulting in relatively larger tie strengths, and hence a larger value of τ is appropriate. If τ is too large/small, almost all edges will be characterized as weak/strong ties, our proposed algorithms will not work well as they would not be able to distinguish between weak and strong ties. In all our experiments, we set τ as the *average tie strength* in the entire social network, which is 0.206 for Gowalla, 0.172 for Twitter, 0.161 for Weibo, 0.104 for Foursquare and 0.486 for DBLP.

4.3.3 Scalability

We also compare the scalability of WeakTie, and incremental algorithms WeakTie-Local and WeakTie-Bi. We use the largest Foursquare dataset for this experiment. Since WeakTie-Bi requires undirected graph, we convert each directed edge (u, v) in the Foursquare dataset to an undirected edge by adding the edge (v, u) if it does not exist. We start with an initial size of 500k nodes and increase the dataset size by 500k nodes as well as their associated edges at each time step.

Figure 4.11 shows that WeakTie is the slowest since WeakTie has to re-run the algorithm on the entire graph when updates arrive. On the other hand, both WeakTie-Local and WeakTie-Bi only look at the local view of the affected nodes to handle the new updates, hence they run much faster than WeakTie as the dataset size increases. Furthermore, WeakTie-Bi eliminates the need to call Tarjan's algorithm in most cases and achieves the best runtime performance.

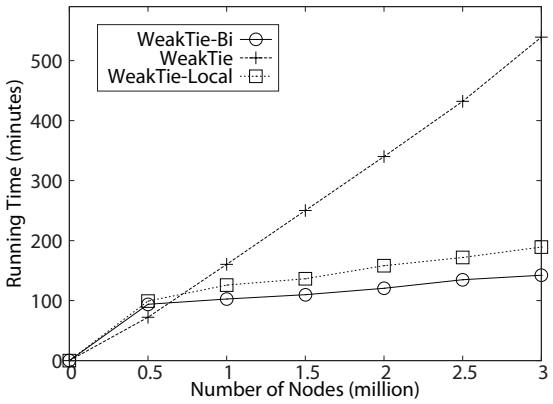


Figure 4.11: Scalability.

4.3.4 Applications of Brokers

Structural Hole Spanner Detection

In this set of experiments, we demonstrate the usefulness of brokers in diffusing information across communities. Lou and Tang [LT13] define structural hole spanners as users who have connections with different communities, and propose two models, MaxD and HIS to find the structural hole spanners. Given a set of communities $\mathbf{C}$, MaxD iteratively finds a node such that removing this node leads to the largest reduction in the minimum cut of the communities in $\mathbf{C}$. The first k nodes that are removed are the top-k spanners. In the HIS model, each user is assumed to have an importance score in each community. The structural hole score is the minimum value of a node's importance score in the different communities. The k nodes with highest structural hole scores are the spanners.

We adapt our WeakTie methods to detect the top-k spanners by finding the nodes that have connections to different communities with the highest WeakTie scores. Note that the spanners are a subset of the brokers we identify.

In the DBLP dataset, authors who have served as program committee members in conferences of different areas are considered as spanners [LT13]. Figure 4.12 shows the precision of the different approaches on this dataset. We observe that WeakTie significantly outperforms MaxD (+35%). Betweenness performs poorly, indicating spanners do not necessarily carry large numbers of shortest paths. HIS and PageRank♯ perform similarly as they both look for nodes that are incident with authority nodes but spanner authors do not merely establish cooperation with opinion leaders in different areas. WeakTie-Local performs slightly worse than WeakTie since it utilizes only the strongly connected groups in the local view.

For the social network datasets, since we do not have the ground truth, we examine the information diffusion paths instead. We demonstrate the results using Twitter and Foursquare datasets here. We divide Twitter dataset into 4 communities and Foursquare

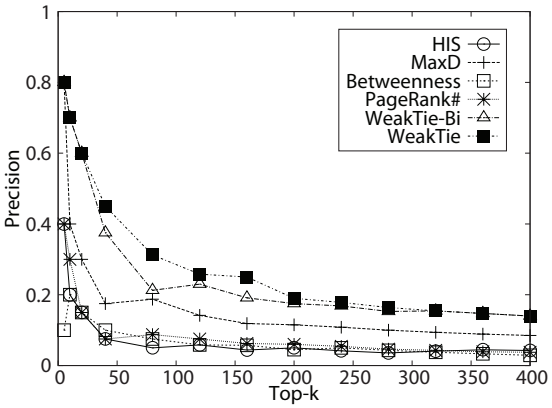
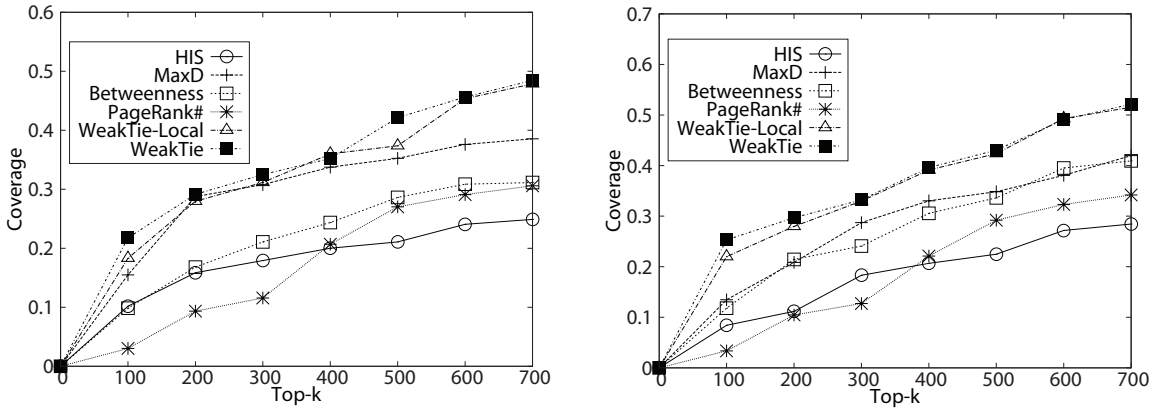


Figure 4.12: Precision of detected spanners in DBLP.



(a) Twitter **(b)** Foursquare

Figure 4.13: Coverage of detected spanners in Twitter and Foursquare.

dataset into 8 communities using Girvan-Newman's community detection method [New04]. Let P be the set of paths that involve at least two users in different communities. Suppose S is the set of spanners found by a method. Let $P_S \subseteq P$ be the set of paths involving users in S. We define the coverage of a method as:

$$coverage = |P_S|/|P|.$$

A high coverage indicates that more information is being diffused across communities by the detected spanners. Figure 4.13 shows the coverage of the various methods in the Twitter dataset. We observe that WeakTie has the best performance, with WeakTie-Local being a close runner-up. On average, WeakTie outperforms the state-of-the-art method MaxD by more than 20% in both datasets.

Mention Recommendation

Next, we discuss how top-k brokers can be used in mention recommendation to expand the spread of tweets. By adding "@username" in a tweet, the system will push the post to the mentioned user whose retweets could enable this post to reach more users. Note that for this mention recommendation task, since we need the retweet cascades as ground truth, we keep only tweets that have been retweeted more than 5 times. This prunes the TWITTER dataset to 27,754 users, 1,285,097 edges and 35,480 paths.

The state-of-the-art mention recommendation method is WTM [WWB+13]. Given a tweet posted by a user u, WTM will rank the other users based on how well their interests match u's post, their interaction with u and how influential they are. In order to demonstrate how brokers are also useful in mention recommendation, we incorporate the WeakTie score into WTM's model of ranking the users. We call this model WTM-WeakTie.

We evaluate the effectiveness of WTM and WTM-WeakTie by the number of users reached using the top-k ranked users. The evaluation measure in [WWB+13] sums up the number of followers of the users in a cascade initiated by a recommended user regardless of whether these followers are duplicates. Based on this measure, we found that each tweet reaches 589 users on average for our TWITTER dataset. However, if we consider only the distinct followers, each tweet actually reaches only 364 users on average. In order to present a more accurate picture of the actual number of people reached, we define the following measure $numReached$ which counts only the distinct followers.

Let L be the set of users who actually retweet, R_u denote the set of users involved in the retweet cascades initiated by a user u, and $Follower(w)$ denote the set of w's followers. We have:

$$numReached = |\cup_{u \in L} \cup_{w \in R_u} Follower(w)|. \tag{4.4}$$

Figure 4.14 shows the results of WTM and WTM-WeakTie. We observe that after incorporating the WeakTie score, we have increased the number of users reached by 23% because brokers can help diffuse the tweet to users located far away from the author and initiate a cascade with more distinct users.

We further analyze the recommended users' ability to reach different parts of the network. We divide the dataset into 100 communities. Here we divide the dataset into smaller communities to avoid many retweets' cascades traveling within one big community. We consider that a user is able to reach a certain community if he/she has a follower from this community. In this set of experiments, we also wish to compare the performance of the identified brokers to the nodes selected by Target-IM algorithm. Target-IM algorithm identifies the nodes that can influence the largest number of targeted nodes regardless of the distance between these nodes. To adopt Target-IM in our mention recommendation task, we set the deadline to inf and consider all users as target users. We compute the score for each node using Target-IM.

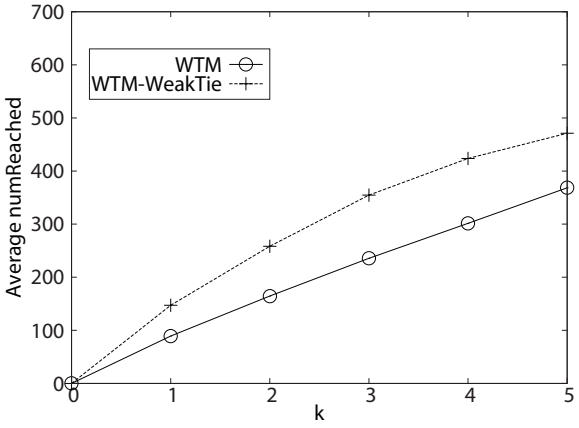


Figure 4.14: Number of users reached in Twitter.

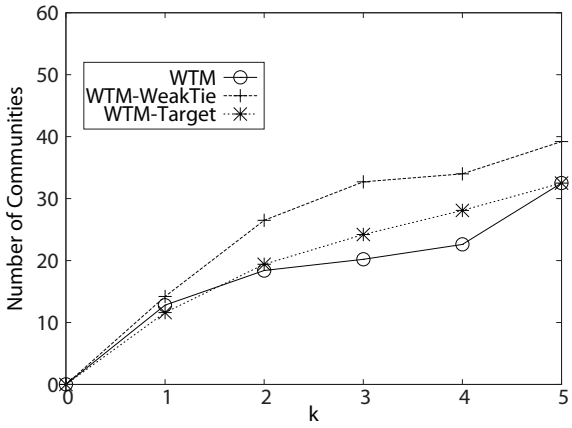


Figure 4.15: Number of communities reached in Twitter.

The score for each node is normalized to fall in the range $[-1, 1]$ and is incorporated into WTM as an additional feature. We call this algorithm WTM-Target. Figure 4.15 shows the average number of communities the users in the recommendation list can reach. We observe that when $k > 1$, WTM-WeakTie can reach many more communities than WTM and WTM-Target, indicating brokers are capable of broadcasting information to a wider range. On the other hand, the performance of WTM-Target is close to WTM because they both fail to identify the nodes that can reduce the pair-wise distance between nodes. In other words, many of the influenced nodes recommended by WTM and WTM-Target are placed within the same community and have many overlapping friends.

4.4 Summary

In this chapter, we have defined the problem of detecting brokers in social networks, and proved that the problem is NP-hard. We have also designed two incremental algorithms WeakTie-Local and WeakTie-Bi to deal with evolving social networks. Experiments on two real world datasets demonstrate the effectiveness and efficiency of the algorithms. We have also shown that utilizing brokers can improve the precision of spanner detection by 35% and help the mention recommendation task to reach 23% more distinct users.

Chapter 5

Node Immunization over Infectious Period

When there are rumors propagating in the social network, instead of boosting their influence, we wish to minimize their effect and prevent them from infecting more users. In this section, we address the first challenge in preventing misinformation spread. When there is an infectious period over which we are allowed to distribute immunization resources, what is the best solution for preventing the spread of rumor. We introduce our *Node Immunization over Infectious Period* problem and present our proposed algorithms.

5.1 Problem Definition

We consider a directed, weighted graph $G = (V, E)$ as the input graph. For each edge $(u, v) \in E$, there is a probability p_{uv} representing the probability that u passes infection to v. Apart from getting infected via network connections, healthy nodes have a probability of getting infected from external sources during the infectious period, and newly infected nodes can, in turn, infect their neighbors and propagate the contagion. To model this, we introduce two parameters α and τ, where α denotes the probability that a healthy node gets infected by some external sources, and τ denote the number of time points where a healthy node can be infected by some external sources and $1 \leq t \leq \tau$. Our goal is to find a set of nodes to immunize such that the expected number of healthy nodes at the end of the propagation is maximized.

Definition 7. ***NIIP problem.*** *Let $G = (V, E)$ be an input network where V is the set of nodes and E is the set of edges. Each edge $(u, v) \in E$ has an infection probability p_{uv}. Let $I_t \subset V$ be a set of infected nodes at time point t where $1 \leq t \leq \tau$, and α be the probability that a healthy node is infected by an external source. We define $\Phi(A_1, A_2, \cdots, A_\tau)$ to be the number of nodes that remain healthy if the nodes in A_t are immunized at time point t. Our goal is to find τ sets of nodes, $A_1, A_2, \cdots, A_\tau$, such that $\sum_i |A_i| = k$ and $\Phi(A_1, A_2, \cdots, A_\tau)$ is maximized.*

Note that the Data-Aware Vaccination (DAV) problem defined in [ZP14a] is a special case of NIIP with $\tau = 1$ and $\alpha = 0$. Table 5.1 summarizes the notations used in this section.

Table 5.1: Summary of Notations

Symbol	Description
$G = (V, E)$	directed graph G with set of nodes V and set of edges E
p_{uv}	the infection probability from u to v
k	number of nodes that can be immunized
τ	number of time points where external infection can occur
I_t	the set of infected node at time point t
α	probability of nodes getting infected by external sources
A_t	the set of nodes to be immunized at time point t
$\Phi()$	the expected number of healthy nodes at the end of propagation
$parent(u)$	the set of nodes that point to node u
$child(u)$	the set of nodes that node u points to

5.1.1 Problem Analysis

Next, we show that the NIIP problem is NP-hard, and even when we restrict the problem to directed acylic graphs (DAG), the problem remains NP-hard.

Theorem 4. *The NIIP problem in Definition 7 is NP-hard.*

Proof. (Sketch.) Zhang et al. [ZP14a] has shown that the Data-Aware Vaccination (DAV) problem is NP-hard. Since the DAV problem is a special case of NIIP problem with $\tau = 1$, we see that if there is a polynomial time solution to NIIP problem, we can set $\tau = 1$ and solve the DAV problem in polynomial time. In other words, the NIIP problem is NP-hard. □

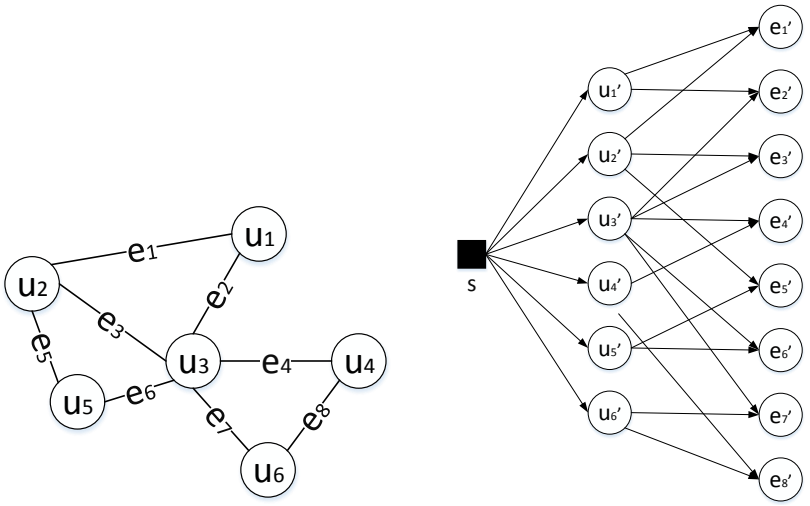
Theorem 5. *The NIIP problem restricted to a directed acyclic graph is NP-hard.*

Proof. We show that the base case of NIIP problem on a directed acyclic graph with $p_{uv} = 1$ for any $(u, v) \in E$, $\tau = 1$ and $\alpha = 0$ is NP-hard. If the base case is NP-hard, then the general NIIP problem on directed acyclic graph must be NP-hard. To prove that the base case NIIP problem is NP-hard, we first show that the decision version of the problem can be reduced from its function version in polynomial time, and then we prove the decision version of the problem is NP-hard.

The decision version is stated as follows: Is there a subset of nodes A in a DAG where $|A| = k$ such that $\Phi(A)$ in Definition 7 is greater than a given value? Clearly, given a directed acyclic graph with n nodes and the infected nodes are I_1, there are at most $|V| - |I_1|$ possible values for $\Phi(A)$. If we can solve the decision version in polynomial time, then the solution to our problem is the set of nodes that gives the largest $\Phi(A)$ value.

Next, we show that the decision problem on a DAG is NP-hard by reducing it from the *k-densest subgraph* problem, namely, determining a k-node subgraph with at least p edges in a given graph [AHI02]. Given an instance $G = (V, E), k, p$ of the k-densest subgraph problem, let $n = |V|$ and $m = |E|$. We can construct a directed acyclic graph $G' = (V', E')$ of $(1+n+m)$ nodes and $2m$ edges as follows. We first create a source node s in G'. For each node u in G, we create a corresponding node u' in G' and connect s to u'. For each edge $e(u, v)$ in G, let u' and v' be the corresponding nodes of u and v respectively. We create a node e' in G' and connect u' and v' to e'. Let $Y_1 = \{u'_1, \cdots, u'_n\}$ and $Y_2 = \{e'_1, \cdots, e'_m\}$. Then $V' = Y'_1 \cup Y'_2 \cup \{s\}$.

Figure 5.1 shows an example of the construction from graph G to G'. It is clear that the construction can be done in polynomial time.



(a) G **(b)** G'

Figure 5.1: Construction of G' from G.

Next, we prove that the k-densest subgraph instance is satisfiable *if and only if* there exists a subset $A \subset Y_1$ where $|A| = k$ such that $\Phi(A) \geq (k + p)$.

For the *only if* direction, suppose the k-densest subgraph instance is satisfiable, that is we have a subgraph with k nodes and at least p edges among them in G. Let $A \subset Y_1$ be the set of nodes in G' that corresponds to these k nodes. When we remove the nodes in A from G', there are at least p nodes in Y_2 in G' that become isolated from s. These p isolated nodes correspond to the p edges in the k-densest subgraph solution. Thus, by immunizing the k nodes in A, we have made p nodes healthy. Hence $\Phi(A) = (k + p)$.

For the *if* direction, suppose the NIIP problem is satisfiable, i.e., $\exists A \subset V'$ such that $\Phi(A) \geq (k + p)$. Assume $|A \cap Y_1| = l$ and $|A \cap Y_2| = (k - l)$. Nodes in Y_1 can possibly immunize nodes in Y_2 while immunizing nodes in Y_2 can only immunize themselves. This means the nodes in $A \cap Y_1$ immunize $(k + p) - (k - l) = (p + l)$ nodes in Y_2 in order for

A to immunize $(k+p)$ nodes in total. These $p+l$ nodes in Y_2 correspond to the edges in G whose both end points are in $A \cap Y_1$. Thus we have found a subgraph in G with $l \leq k$ nodes and at least p edges among them. In other words, the k-densest subgraph instance I is satisfiable. □

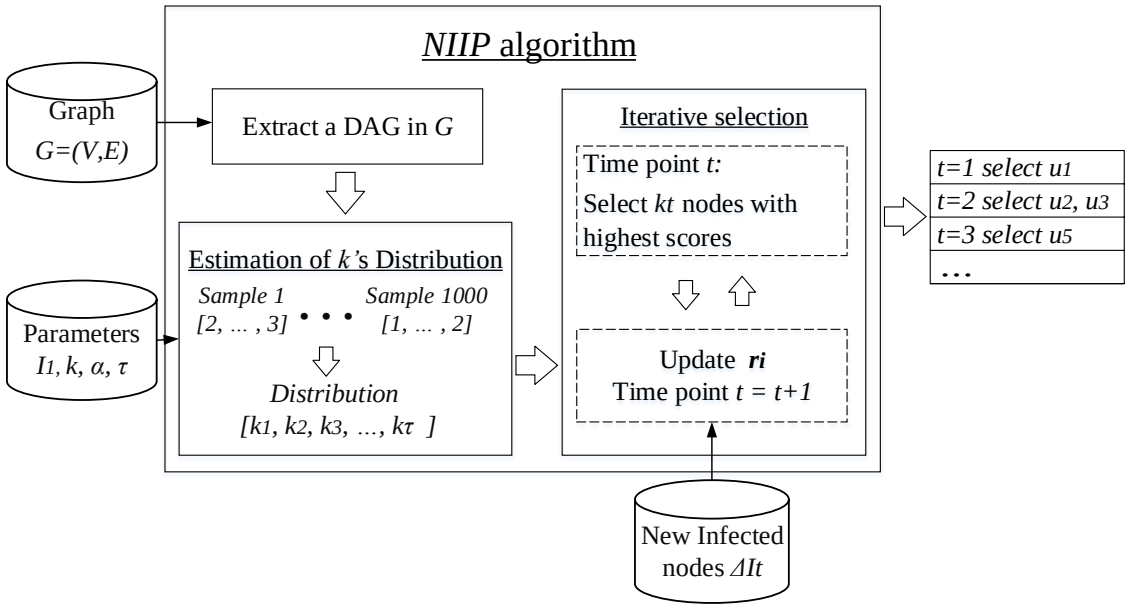


Figure 5.2: Overview of proposed approach.

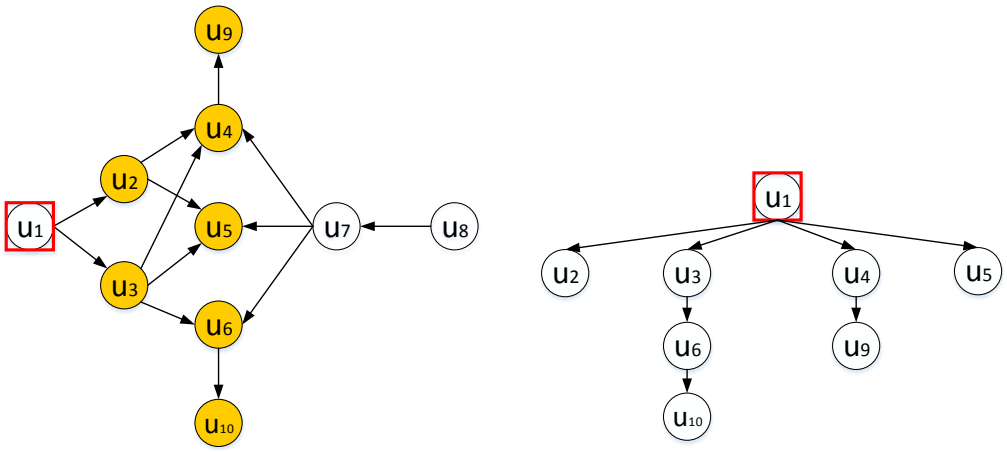
5.2 Algorithms

Given that the NIIP problem is NP-hard, we use a combination of simulation-based and greedy approach to solve the problem. Figure 5.2 shows an overview of the main steps in our proposed approach. Given a network graph, we first extract a maximum DAG from the graph. We then perform a Monte Carlo simulation to estimate the distribution of k over each time point t in τ given the probability α of a healthy node being infected. Next, for each time point t in τ, we compute a score for each node that reflects the node's immunization ability. Suppose the estimated distribution of k at time point t is k_t, this implies that at time point t, we need to select k_t nodes to be immunized. Hence, the top k_t nodes with the highest scores are selected and the scores of the affected nodes are updated. We repeat this process for each t in τ.

The details of each step are given in the following subsections. We first discuss how to select the best k_t nodes at a given time point t. Then we describe how to perform simulation to estimate the distribution of k over the τ time points. Based on this estimated distribution of k, we design a scalable NIIP algorithm to select τ sets of nodes to immunize at each time point t.

5.2.1 Single time point NIIP

Different nodes have different immunization abilities depending on the network structure and the infected nodes. The Dava algorithm models this with a dominator tree. Figure 5.3 shows a dominator tree constructed for Figure 5.3a given u_1 is infected. Based on this tree, if we can only immunize 2 nodes, the Dava algorithm will choose u_3 and u_4 instead of the preferred choice u_2 and u_3. This is because the dominator tree does not capture the joint effect when two or more nodes are immunized. If only u_2 is immunized, none of the nodes can be saved. However, immunizing both u_2 and u_3 will result in the saving of 5 nodes. To model this joint effect, we introduce the concept of immunization ability of a node.



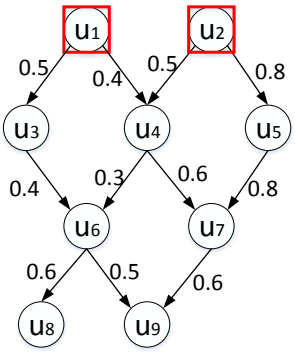
(a) Example network **(b)** Dominator tree

Figure 5.3: Dava algorithm for the example network.

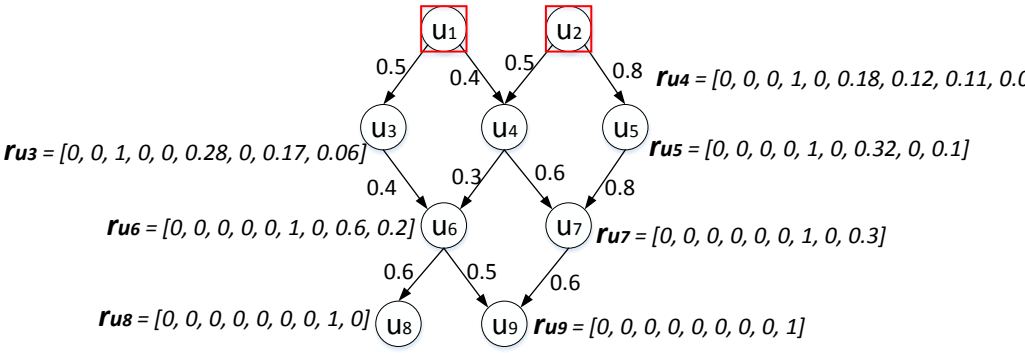
Consider the example in Figure 5.4a, where each edge (u, v) is labeled with the probability of node u infecting v, denoted as p_{uv}. We see that u_9 has two parents u_6 and u_7. If we immunize u_6 only, u_9 can still be infected by u_7 or via some external source. To measure the protection gain to u_9 by immunizing u_6, we first compute the probability that u_9 does not get its infection from either u_6 or u_7. This is given by $\prod_{v\in\{u_6,u_7\}}(1-p_{vu_9}) = (1-0.5)(1-0.6) = 0.2$. The probability that u_9 is not infected by u_7 is $1-0.6 = 0.4$. Hence, the gain by immunizing u_6 is $0.4 - 0.2 = 0.2$. We term this the immunization ability of u_6 over u_9.

Denote $parent(u)$ as the parents of a node u and $child(u)$ denote the children of u. We model the immunization ability of u over the rest of the nodes as a vector $\mathbf{r_u}$. The v^{th} entry in $\mathbf{r_u}$, denoted as m_{uv}, represents the immunization ability of u over v. m_{uv} is given as:

$$m_{uv} = \begin{cases} \prod\limits_{\substack{w\in parent(u)\\ w\neq u}} (1-p_{wv}) - \prod\limits_{w\in parent(u)} (1-p_{wv}), & u \in parent(v) \\ \\ \sum\limits_{w\in child(u)} (m_{uw} \times m_{wv}), & u \notin parent(v) \end{cases} \tag{5.1}$$



(a) Example network.



(b) Computation of $\mathbf{r_u}$ for each node.

Figure 5.4: Illustration of computing $\mathbf{r_u}$.

Intuitively, this makes sense because when node v has many parents, even though u is immunized, all the other parents of v could still pass the infection to v. Hence, in this case, the m_{uv} value should be small. On the other hand, if node v has only two parents, when u is immunized, the probability of v getting infected is much lower. In other words, m_{uv} value should be large. Note that when v has only one parent u, $m_{uv} = 1 - (1 - p_{uv}) = p_{uv}$. Also, for any node u, $m_{uu} = 1$. This is because when a node is immunized, the probability of it getting infection is 0.

For each node u, we compute the immunization ability of node u over the rest of the nodes in a vector, i.e. $\mathbf{r_u}$, as follows:

$$\mathbf{r_u} = \sum_{v \in child(u)} m_{uv} \cdot \mathbf{r_v} + \mathbf{e_u} \tag{5.2}$$

where $\mathbf{e_u}$ is a unit vector with its component corresponding to u being one and elsewhere being zeros.

Figure 5.4b illustrates the computation of $\mathbf{r_u}$ vector for each node. Assume u_1 and u_2 are the infected nodes. The susecptible nodes are $[u_3, \cdots, u_8, u_9]$. Traversing this in the reverse order, we start with u_9. Since u_9 has no children, $\mathbf{r_{u_9}}$ has a "1" on the 9th column and zeros elsewhere. Similarly for $\mathbf{r_{u_8}}$ with a "1" on the 8th column. Next, we come to u_7. The immunization ability of u_7 over u_9 is $m_{u_7u_9} = (1 - 0.5) - (1 - 0.5) \times (1 - 0.6) = 0.3$, thus we have $\mathbf{r_{u_7}} = 0.3 \times \mathbf{r_{u_9}} + \mathbf{e_{u_7}}$, as shown in Figure 5.4b. Similarly, we compute $m_{u_6u_9} = (1 - 0.6) - (1 - 0.5) \times (1 - 0.6) = 0.2$. Note that $m_{u_7u_9} > m_{u_6u_9}$ makes sense since $p_{u_7u_9} > p_{u_6u_9}$. This implies that the gain obtained by immunizing u_7 is greater than immunizing u_6.

With this, we can now compute the score for each node u:

$$score(u) = \sum_{j=1,\cdots,|V|} \mathbf{r_u}[j] \tag{5.3}$$

Given a graph G and the set of infected nodes I_t, our goal is to select the top k_t nodes with the greatest scores. We first obtain the set of susceptible nodes S by performing depth first search starting with each node in I_t. Then we apply the Acyclic algorithm [EIL+12] to find the maximum DAG in S. With the DAG, we fix a topological order σ where the nodes are ordered such that each edge in the graph originates from an earlier node to a latter node in this ordering. We traverse the nodes and compute its score in the reverse order of σ. This guarantees that we will compute the score of a child node before computing the score of its parent. The node with the highest score is marked as *immunized*. The scores of its parent nodes need to be updated as they have one less child whom they can potentially infect, and hence, their immunizing ability is reduced. In addition, the descendants of u will be affected. For a node $v \in child(u)$, if u is v's only parent, we mark v as *immunized* since the only infection path to v is from u which has been immunized. For those nodes $w \in child(u)$ with more than one parents, their parents' scores need to be updated too.

The details of S-NIIP algorithm is given in Algorithm 5.1. Given the set of infected nodes I_t, we first obtain the set of susceptible nodes S and find the maximum DAG (Lines 2-4). Next, we compute the vectors as well as the scores of the nodes in S in a reverse topological order (Lines 5-11). We select the node with the highest score and call the function $Immunize(u)$ to mark u as "immunized" and place the the parents of u into the $UpdateList$. For each node in the $UpdateList$, we call $ProcessUpdate$ to recompute the score for the node and propagate the effect to the parent nodes if they are susceptible. The process is repeated till k_t nodes have been immunized (Lines 12-16).

Consider the example in Figure 5.4a. The infected nodes are u_1 and u_2, i.e. $I_1 = \{u_1, u_2\}$. The set of susceptible nodes is $S = \{u_3, u_4, u_5, u_6, u_7, u_8, u_9\}$. We compute the vectors as well as the scores for the nodes in S and select the node with the highest score, i.e. u_6 for immunization (See Figure 5.4b). After immunization, u_6's parents u_3 and u_4 lose one child, and we put u_3 and u_4 into $UpdateList$. Its child u_8 is "immunized" as well since u_8 has only one parent u_6. Its child u_9 has not been "immunized" because it still has u_7 as a parent. Since u_7 becomes the only parent of u_9, it is also placed in $UpdateList$ for updates. Now $UpdateList = \{u_3, u_4, u_7\}$. We first update $\mathbf{r_{u_7}}$, and we notice u_7's parent $u_5 \in S$ is not in $UpdateList$, hence we insert u_5 into $UpdateList$. We then update $\mathbf{r_{u_4}}$ and remove u_4 from $UpdateList$. Since u_4's parents u_1 and u_2 are not in S, we do not put u_1 or u_2 in $UpdateList$. Similarly for u_3, we update $\mathbf{r_{u_3}}$ and remove u_3 from $UpdateList$. This process causes $\mathbf{r_{u_3}}$, $\mathbf{r_{u_4}}$, $\mathbf{r_{u_5}}$ and $\mathbf{r_{u_7}}$ to be updated as marked in red in Figure 5.5.

Algorithm 5.1: S-NIIP algorithm

input : 1. Graph $G = (V, E)$
2. Number of nodes to immunize k_t
3. Infected node set I_t

output : Top-k nodes for immunization

Initialize $UpdateList = \emptyset$, $S = \emptyset$;
foreach $u \in I$ **do**
    Peform DFS from u and insert visited nodes into S;
end
Apply Acyclic on S to generate a DAG;
Find a topological sorting σ in the DAG;
foreach *node* v *in the reverse order of* σ **do**
    foreach $(u, v) \in E$ **do**
        $parent(v) = parent(v) \cup \{u\}$;
        $child(u) = child(u) \cup \{v\}$;
    end
    Compute $\mathbf{r_v}$ with Equation 5.2;
    Compute $score(v)$ with Equation 5.3;
end
repeat
    Return node u with highest $score(u)$;
    Call $Immunize(u)$;
    Call $ProcessUpdate(UpdateList)$;
until k *nodes are selected*
Return top-k_t nodes for immunization.

Function ProcessUpdate(UpdateList)

foreach $u \in UpdateList$ **do**
    Compute $\mathbf{r_u}$ with Equation 5.2 and $score(u)$ with Equation 5.3;
    $UpdateList = UpdateList \cup (parent(u) \cap S)$;
    $UpdateList = UpdateList - \{u\}$;
end

Function Immunize(u)

```
1  score(u) = 0, S = S − {u};
2  foreach v ∈ child(u) do
3      if parent(v) = {u} then
4          Call Immunize(v);
5      else
6          parent(v) = parent(v) − {u};
7          UpdateList = UpdateList ∪ (parent(v) ∩ S);
8  end
```

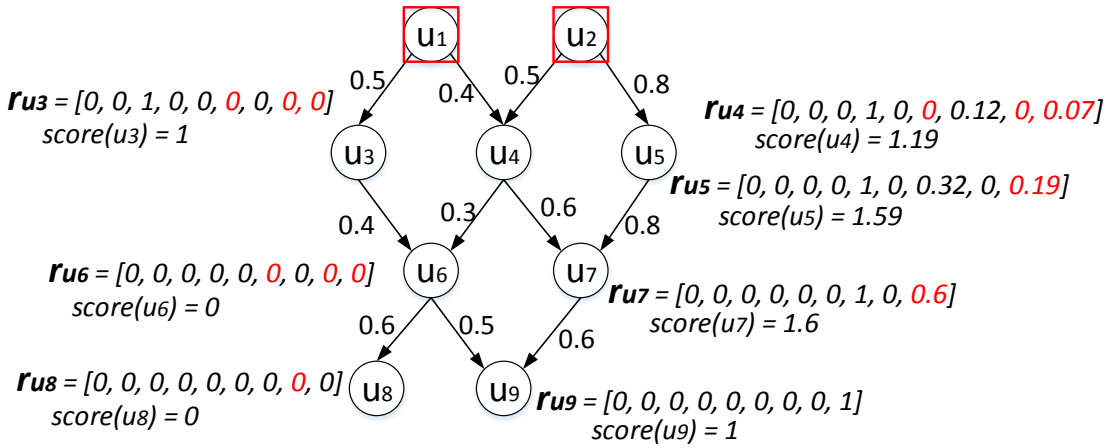


Figure 5.5: Updated $\mathbf{r_u}$ and scores after u_6 is immunized.

5.2.2 Estimation of k's distribution over τ

When an infection happens over a time period, putting all the available resources at the start of the infection is not a good strategy as shown in Section 1. We note that the number of nodes we should choose to immunize at each time point is influenced by the initial set of infected nodes I_1 and the rate of infection from external sources α. Given a large I_1 set, we would want to select more nodes to be immunized earlier on so as to prevent the spread of infection from I_1. On the other hand, when we have a large α value, we would want to reserve more quota to later time points.

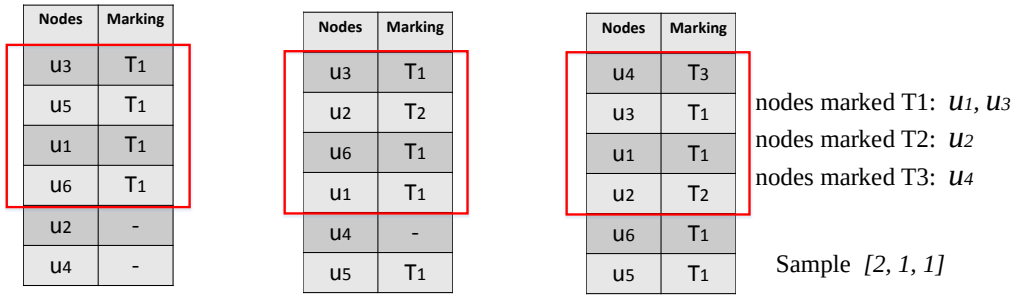
We perform a Monte Carlo simulation to decide how to distribute the limited resources k over the time point t in τ. We generate τ lists $\{I_1, I_2, \cdots, I_\tau\}$ of randomly infected nodes where $I_{t+1} = I_t \cup \{\text{new infected nodes by } I_{t-1} \text{ and by external sources with probability } \alpha\}$. Then we apply the S-NIIP algorithm to get the top k nodes for each list of infected nodes. We process these lists in increasing time order, i.e. from I_1 to I_τ. When all the lists have been processed, we examine each node in the top k results corresponding to time point τ. If the node has appeared in the top k results of an earlier time point, we tag the node with the earlier time point label. The number of the nodes that are tagged with time points $T_1, T_2, \cdots, T_\tau$ forms an instance of the distribution of k over τ. We repeat the process 1000 times and take the average to obtain an estimated distribution of k over τ.

Figure 5.6 shows the estimation process with $k = 4$ and $\tau = 3$. When $t = 1$, we retrieve the top 4 nodes based on the infected node list I_1 and tag them with T_1. At the next time point, we produce a new top-4 list regarding to I_2. Here node u_2 is selected for immunization, and we mark u_2 with T_2. Note that if the same node appears in multiple time points, we mark it with the earliest time point. Similarly in time point $t = 3$, we mark u_4 with T_3. Afterwards, we select the last set of top-k nodes and count the number of markings for each time point. In this example, 2 nodes are selected in time point 1 while one node is selected in time point 2 and 3 respetively. This creates a sample of $[2, 1, 1]$. We repeat the process 1000 times and choose the average number of nodes appearing in each time point as an estimated distribution of k over τ.

5.2.3 NIIP over infectious period

With the estimated distribution of k over τ, we extend the S-NIIP algorithm to allow for the selection of k nodes to immunize over a time period. A naive extension is to call the S-NIIP algorithm repeatedly for each time point in τ with a different k_t corresponding to the estimated k at time point t. However, this is not scalable as the number of nodes whose scores need to be updated increases significantly when the number of infected nodes grows.

A careful study reveals that the immunization ability is greatest when it is from a parent to its immediate child. To reduce the number of updates needed, we model only the direct



I_1 is given. $I_2 = I_1 \cup \{$ new infected nodes $\}$. $I_3 = I_2 \cup \{$ new infected nodes $\}$.

(a) $t = 1$ **(b)** $t = 2$ **(c)** $t = 3$

Figure 5.6: Estimating Distribution of k. $k = 4$, $\tau = 3$.

immunization ability between a parent node and its children nodes. We divide the children of node u, $child(u)$, into two sets, $C1$ and $C2$ such that the nodes in $C1$ have only u as their sole parent, while the nodes in $C2$ have multiple parents besides u. The direct immunization ability of node u, denoted as $\mathbf{r'_u}$, is computed as follows:

$$\mathbf{r'_u} = \sum_{v \in C1} p_{uv} \cdot \mathbf{r'_v} + \sum_{v \in C2} m_{uv} \cdot \mathbf{e_v} + \mathbf{e_u} \tag{5.4}$$

where $\mathbf{e_u}$ is a unit vector with its component corresponding to u being one and elsewhere being zeros.

Let us illustrate the computation of $\mathbf{r'_u}$ with the same example network in Figure 5.4a. Vectors for u_6, u_7, u_8 and u_9 are the same because they have no grandchildren in the network. When we compute $\mathbf{r'_{u_3}}$, since u_3 is one of the parents of u_6, u_6 contribute $m_{u_3u_6} \cdot \mathbf{e_{u_6}}$ to $\mathbf{r_{u_4}}$. However, u_3 no longer has immunization ability over u_8 or u_9 as they are not the direct children of u_3 and their direct parent u_6 has multiple parents besides u_3. Hence we compute $\mathbf{r_{u_3}} = m_{u_3u_6} \cdot \mathbf{r_{u_6}} + \mathbf{e_{u_3}}$. The final direct immunization vectors are shown in Figure 5.7a.

Algorithm 5.2 details how we incorporate the direct immunization vector computation to select the top k nodes over τ time points. We first estimate the distribution of k in Line 1. We then apply Acyclic algorithm to extract a maximum DAG in G and find a topological sorting σ in Lines 2 and 3. We find the susceptible nodes in Lines 5 and 6 with depth-first search algorithms and compute their direct immunization vectors in the reverse order of σ (Lines 7 and 8). The $ComputeVector()$ function update the parent set as well as the parents' children set when we reach each node. We then compute the vector following Equation 5.4 and the score following Equation 5.3. At each time point, Lines 10-17 take in the newly infected nodes and set their scores to zero. Line 14 puts their parents in $UpdateList$ as they loses a direct child as well as their corresponding immunization ability. Line 16 computes the score for nodes that newly become susceptible.

Algorithm 5.2: NIIP algorithm

input : 1. Graph $G = (V, E)$
2. Number of nodes to immunize k
3. Time period τ
4. Infected node set I_1
5. Infection rate α

output : Top-k nodes for immunization

Call $EstimateDistribution(G, I_1, \alpha, k)$;
Apply Acyclic on G to generate a DAG;
Find a topological sorting σ in the DAG;
Initialize $UpdateList = \emptyset$, $t = 0$, $S = \emptyset$;
foreach $u \in I_1$ **do**
    Peform DFS from u and insert visited nodes into S;
end
foreach *node* $v \in S$ *in the reverse order of* σ **do**
    Call $ComputeVector(v)$;
end
repeat
    Let ΔI_t be the new infected nodes by I_{t-1} or by external sources with probability α;
    $I_t = I_{t-1} \cup \Delta I_t$;
    foreach $u \in \Delta I_t$ **do**
        $score(u) = 0$;
        $UpdateList = UpdateList \cup (parent(u) \cap S)$;
        Perform DFS from u, insert visited node v in S;
        Call $ComputeVector(v)$;
    end
    Call $ProcessUpdate'(UpdateList)$;
    repeat
        Return node u with the highest $score(u)$;
        Call $Immunize(u)$ in Algorithm 5.1;
        Call $ProcessUpdate'(UpdateList)$;
    until k_t *nodes are selected*
    $t = t + 1$;
until $t = \tau$

Function ComputeVector(v)

foreach $(w, v) \in E$ **do**
  $parent(v) = parent(v) \cup \{w\}$;
end
if $|parent(v)| = 1$ **then**
  foreach $w \in parent(v)$ **do**
    $C1 = C1 \cup \{v\}$
  end
else
  foreach $w \in parent(v)$ **do**
    $C2 = C2 \cup \{v\}$
  end
Compute $\mathbf{r'_v}$ with Equation 5.4;
Compute $score(v)$ with Equation 5.3;

Function ProcessUpdate'(UpdateList)

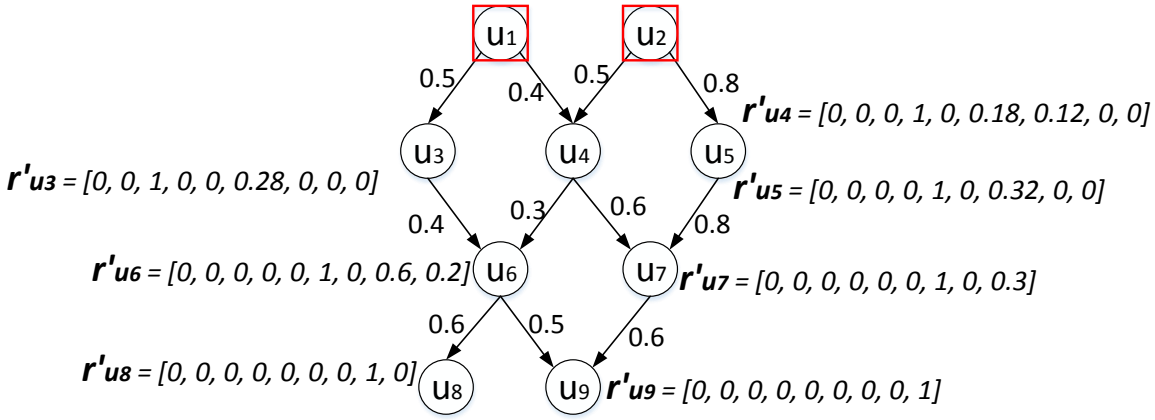
foreach $u \in UpdateList$ **do**
  Compute $\mathbf{r'_u}$ with Equation 5.4 and $score(u)$ with Equation 5.3;
  if $parent(u) = \{w\}$ *and* $w \in S$ **then**
    $UpdateList = UpdateList \cup \{w\}$;
  end
  $UpdateList = UpdateList - \{u\}$;
end

Function $ProcessUpdate'()$ processes the nodes whose vectors need updates. Since we merely consider direct immunization, we propagate the change only when a node has a single parent as indicated in Line 39. This enables the $ProcessUpdate'()$ function to end much faster compared to $ProcessUpdate()$ in S-NIIP algorithm. Lines 18-22 select k_t nodes at time point t, and we repeat the process for τ times.

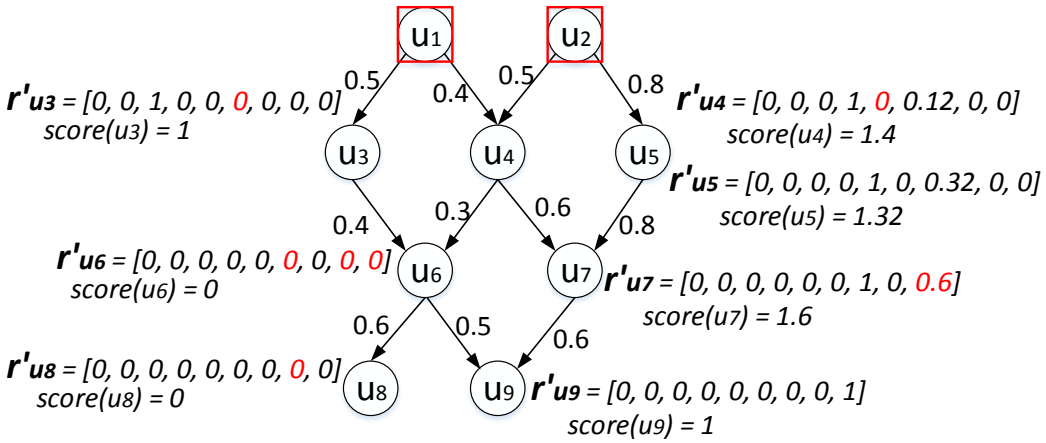
Consider the example in Figure 5.7a. Initially u_1 and u_2 are infected i.e. $I_1 = \{u_1, u_2\}$. The set of susceptible nodes are $S = \{u_3, u_4, u_5, u_6, u_7, u_8, u_9\}$. We compute the scores for the nodes in S and select the node with the highest score, i.e. u_6 for immunization. Its parents u_3 and u_4 lose one child, and we put u_3 and u_4 into $UpdateList$. Since u_7 gains more immunization ability from u_9 because of the immunization of u_6, we insert u_7 into $UpdateList$ for updates. Now $UpdateList = \{u_3, u_4, u_7\}$. Note that all nodes in $UpdateList$ do not need to propagate their updates to their parents because they have multiple parents. Hence only three vectors are changed as marked in red in Figure 5.7b. Note that $\mathbf{r_{u_5}}$ is not updated in this case while it is updated in Figure 5.5. This is because u_7 is not the only child of u_5 and does not propagate the update. The updated vectors for the nodes are illustrated in Figure 5.7b.

Assume in the next time point, we realize u_7 is also infected as illustrated in Figure 5.7c. We update S as the set of nodes that are susceptible as $S = \{u_3, u_4, u_5, u_9\}$. For nodes in $parent(u_7) \cap S$, i.e. u_4 and u_5, they lose the immunization ability over u_7 and hence we put u_4 and u_5 into $UpdateList$ to re-compute their vectors using Equation 5.4. The updated scores are illustrated in red in Figure 5.7c.

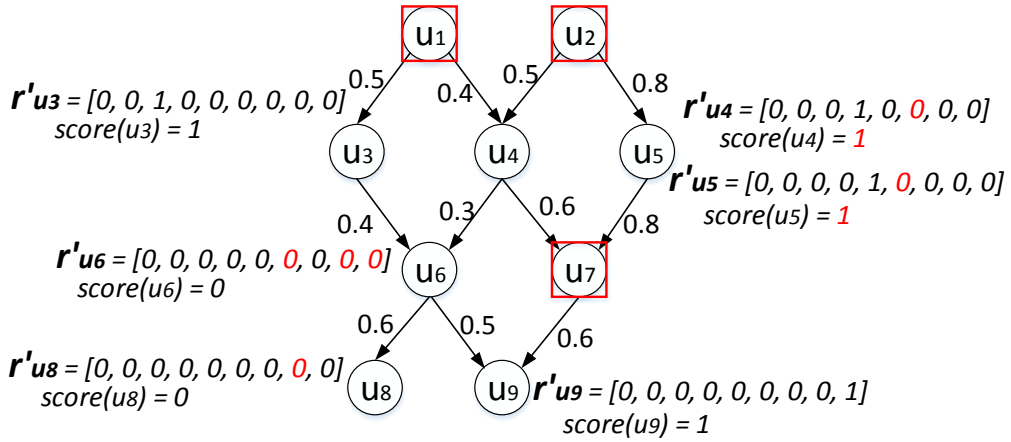
Complexity analysis. In S-NIIP, we first perform a DFS traversal of the nodes in $O(|V| + |E|)$ time and find a DAG with the largest number of edges maintained in $O(|E| \cdot log(|V|))$. When a node is immunized, we trace back to all the ancesters of the affected nodes and update their corresponding vectors. In the worst case, we update each vector of every node and hence its time complexity is $O(k_t \cdot (|V| + |E|))$ if we select k_t nodes. In NIIP algorithm, the initialization takes the same time complexity as S-NIIP, i.e. $O(|V| + |E|)$ for DFS traversal and $O(|E| \cdot log(|V|))$ for DAG extraction. When a node is immunized, we update the vector of its parents and propagate the change if the parents only have one parent. This process costs $O(|V|)$ in the worst case. Similarly when new nodes get infected, the propagation of updates is dealt with in $O(|V|)$. In practice, a node with only one parent is likely to be a leaf node and will not be chosen for immunization as it only immunizes itself. Hence on average, the propagation of updates in NIIP can finish rather quickly as shown in the experiments in Section 5.3.



(a) Initial network with $\mathbf{r}'$



(b) u_6 being immunized



(c) u_7 gets infected

Figure 5.7: Illustration of NIIP algorithm.

5.3 Experiments

In this section, we perform comparative experimental evaluation of the various methods for node immunization problem using real world datasets covering both computer networks and information networks. All the experiments are run on a linux machine with 2 Xeon E5440 2.83 GHz CPU and 16G RAM.

5.3.1 Experimental Setting

Datasets. We conduct experiments on the four real-world datasets described in Section 3.3, *i.e.*Gowalla, Twitter, Weibo and Foursquare. Additionally, since the node immunization problem is also important for prevention of virus spreading in computer networks and malicious news in website networks, we also use a computer network dataset Oregon and a website network dataset MemeTracker in this chapter. Further, to illustrate the effectiveness of preventing rumor spread in the MH370 incident, we have also crawled the propagated tweets talking about the MH370 airline. We describe the new datasets as follows and summarize them in Table 5.2.

Table 5.2: Dataset Summary

Dataset	Nodes	Edges	#Nodes	#Edges
Pig	web sites	hyperlink	8,727	30,309
Economy	web sites	hyperlink	39,639	156,781
Oregon	routers	peering	6,461	21,530
MH370	users	retweet	1.4 million	4.9 million

- MemeTracker[LBK09][1] This dataset contains the hyperlinks between 96 million news sites, ranging from news distributors to personal blogs. Contagion could be false informtaion published on some news sites. Since the original dataset is very large, we select 2 phrases that has been popularly diffused by various news sites, namely "lipstick on a pig" (Pig) and "american economy is in danger"(Economy). Each phrase initiates a cascade of sites reporting this news and hence captures the propogation in real world web sites.
- Oregon[2] This dataset is the Oregon AS router graph collected from the Oregon router views. The contagion could possibly be malware and viruses.

[1] http://snap.stanford.edu/data/memetracker9.html

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

- MH370[3] This dataset contains the users' tweets that contains the hashtag #MH370 after the Malaysian Airlines MH370 flight went missing on March 8th. Retweet relationships form the edges in the graph. Contagion could be rumors about the MH370 flight.

Parameters. τ is the number of steps that we are allowed to assign our immunization nodes. α is a small probability that decides the fraction of healthy nodes getting infected by independent sources. We set $\tau = 10$ and $\alpha = 0.05$ in all our experiments. In our experiment, we set a uniform propagation probability 0.6 [ZP14b]. On each dataset, we conduct our experiment 1000 times and take the average.

Evaluation Metric. We measure the effectiveness of the various methods by defining a metric called *Save Ratio* (SR) which gives the ratio between the reduction in infected nodes when k nodes are immunized over the number of infected nodes with no immunized nodes. We denote the set of infected nodes when nodes in A are immunized as $Immunized(A)$.

$$SR(A) = \frac{|Immunized(\emptyset)| - |Immunized(A)|}{|Immunized(\emptyset)|} \tag{5.5}$$

Note that the value of SR directly correspond to our objective function $\Phi(A)$, i.e. higher $SR(A)$ represents a higher $\Phi(A)$ value. However, the value of SR will fall in $[0, 1]$ where 1 indicates a full immunization of all healthy nodes while $\Phi(A)$ value is not bounded.

Comparison Methods. We compare the performance of the following methods:

- S-NIIP: our proposed approach to select k nodes in one single time point;
- NIIP: our proposed approach to handle node immunization over τ time points
- Betweenness[HC13]: this algorithm returns the top k nodes that carry the largest number of shortest paths;
- PageRank[PBMW98]: this algorithm returns the top k nodes with the highest PageRank scores;
- NetShield [TPT+10]: this method pre-emptively immunize the network to minimize the epidemic threshold of the graph;
- Dava[ZP14a]: this algorithm aims to identify the top k nodes in presence of already infected nodes;

[3] http://www.comp.nus.edu.sg/ a0095629/

5.3.2 Effectiveness Experiments

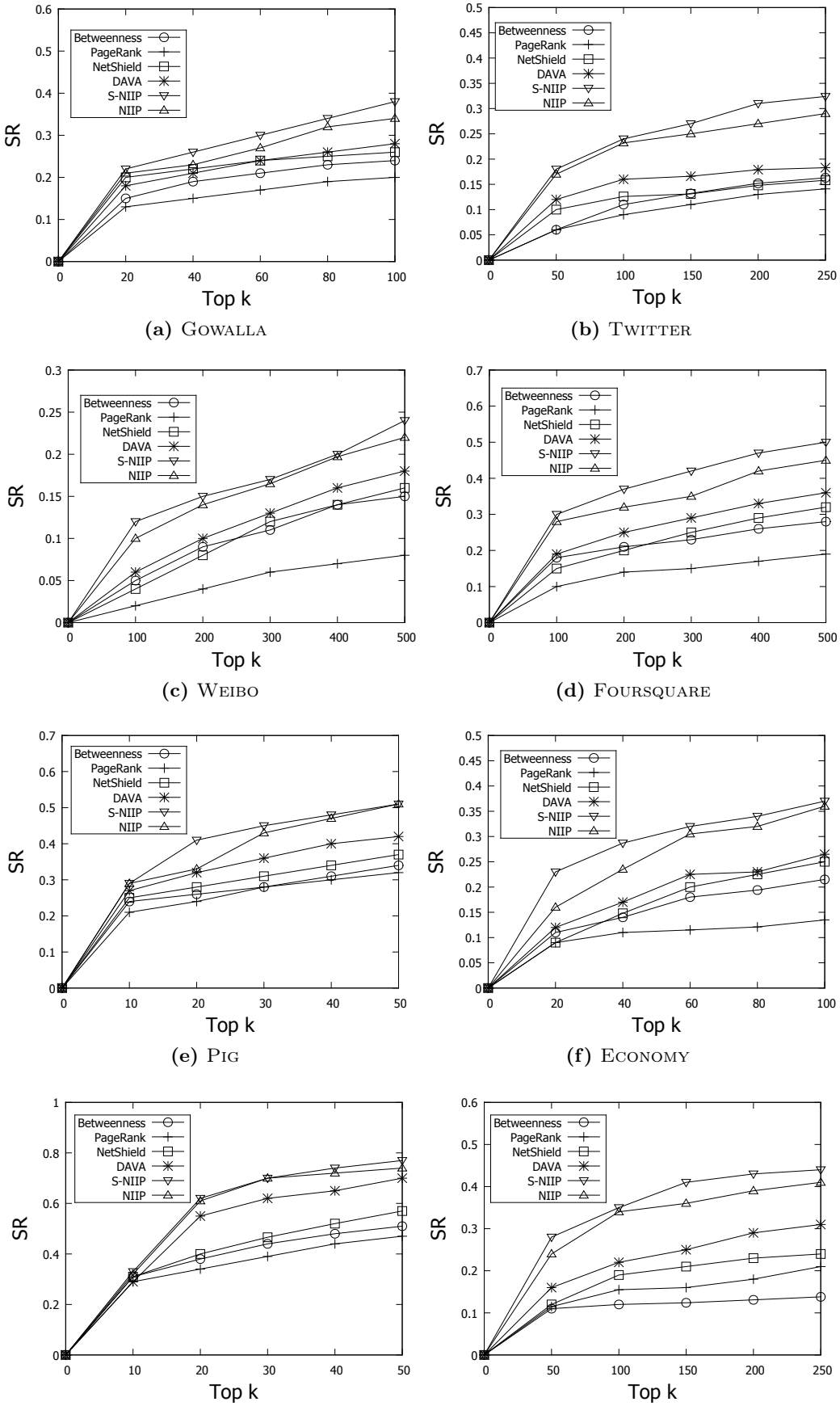
In this section, we show the effectiveness of the proposed methods on node immunization tasks.

Results when $\tau = 1$ and $\alpha = 0$. In this set of experiments, we set the time period to one ($\tau = 1$) and assume there is no external source of infection ($\alpha = 0$). This setting reduces our problem to the DAV problem proposed in [ZP14a]. Figure 5.8 shows the performances of the various algorithms. In all datasets, S-NIIP gives the best performance while NIIP is a close runner-up. The narrow gap between these two algorithms demonstrates that it is sufficient to model the immunization ability based only on the direct parent-child relationships. We also observe that both NIIP and S-NIIP outperform DAVA. This shows clearly that it is advantageous to take into account the joint immunization effect of multiple nodes. In general, algorithms which take into account the infected nodes (namely, S-NIIP, NIIP, DAVA) perform better than those which pre-emptively choose nodes for immunization confirming that it is good to perform immunization while an attack is underway.

In the PIG and the OREGON graphs, S-NIIP and NIIP outperform the state-of-the-art methods by more than 20% when k is large. We note that for these two datasets, they have a small number of nodes with high out-degrees. These nodes are good candidates for immunization. As a result, when k is small, all the algorithms pick this small set of nodes to immunize and their performances are similar. However, as k increases, S-NIIP and NIIP rapidly outperform the other methods.

In all the other larger datasets, S-NIIP and NIIP manage to outperform all the other algorithms even when k is small. We observe that in these datasets, the networks are composed of multiple clusters and the positions of the infected nodes are critical to the selection of immunization nodes. Pre-emptive methods like NetShield, Betweenness and PageRank could not effectively utilize such information and hence perform poorly. As for the Dava algorithm, it constructs a dominator tree from the infected nodes. If a node has multiple parents, the Dava algorithm will link this node directly to the ancestors of its parents and therefore ignore the immunization ability of its parents. As a result, the performance of Dava suffers. We observe that S-NIIP consistently outperforms Dava by 45% in TWITTER and 30% in WEIBO and MH370 datasets.

Results when $\tau = 10$, $\alpha = 0.05$. In this set of experiments, we perform node immunization over 10 time points. For pre-emptive methods PageRank, Betweenness and NetShield, we simply run them with the given network structure as they do not consider the set of infected nodes as input. For the Dava algorithm, we give the final set of infected nodes I_τ as input. This allows the Dava algorithm to have the complete picture of infection nodes before making its selection of k nodes. Figure 5.9 gives the results. We see that the overall SR value is less compared to the graph in Figure 5.8. This is because introducing an infectious period



(a) Gowalla **(b)** Twitter

(c) Weibo **(d)** Foursquare

(e) Pig **(f)** Economy

(g) Oregon **(h)** MH370

Figure 5.8: Performance of different algorithms. $\tau = 1$ and $\alpha = 0$.

implies we have a much larger set of infected nodes. Hence even if we increase the number of nodes for immunization, the overall performance decreases. As expected, pre-emptive methods give the worst performance; while the NIIP algorithm is the clear winner and outperforms all the other methods significantly (+30%) in all datasets. This shows that the immunization strategy adopted by NIIP is the most effective in immunizing the nodes that lead to the largest number of healthy nodes in the network.

Effect of distributions of k. In this set of experiments, we examine how the distributions of k affect our results. We compare three strategies for assigning k:

- NIIP-Estimated: we follow the learned distribution of k to select k_t nodes at each time point;
- NIIP-Uniform: we uniformly select $\frac{k}{\tau}$ nodes for immunization at each time point;
- NIIP-After: we wait until time point τ to select the k nodes all at once.

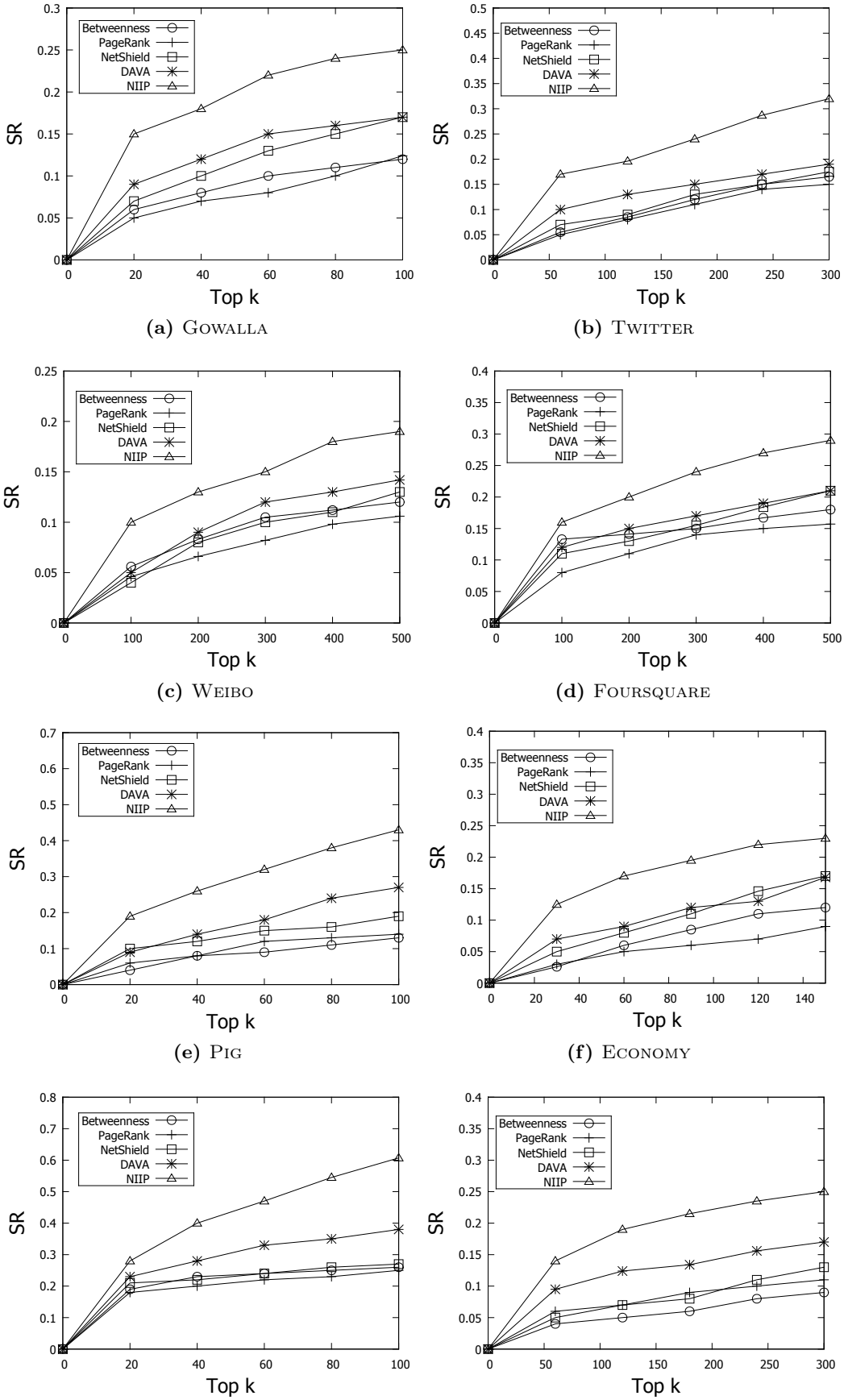
Results are shown in two datasets, Twitter and Foursquare. Figure 5.10 demonstrates that while the estimated distribution of k has the best performance, uniform decision also achieves good results in both datasets. However, choosing the k nodes after τ time points hugely impact the performance, indicating it is an effective method to prevent propagation of infection by immunizing nodes as soon as we observe infections in the network.

5.3.3 Efficiency

In this section, we report the runtime of the algorithms on the two largest datasets Weibo and Foursquare. Table 5.3 shows the running time for S-NIIP, NIIP, Dava and NetShield in the case of $\tau = 1, \alpha = 0$. We see that NIIP can produce outputs within the shortest time. When the input graph is large, Dava runs the slowest. This is because Dava needs to re-generate the dominator tree when each node is selected which requires a near-linear time algorithm. For S-NIIP, it requires near-linear pre-processing time while the worst case for updating the scores is in linear time.

Table 5.3: Execution time (minutes) $k = 100$, $\tau = 1$ and $\alpha = 0$.

	S-NIIP	NIIP	Dava	NetShield
Weibo	352.5	**102.3**	1023.6	248.1
Foursquare	671.0	**155.6**	1620.9	383.8



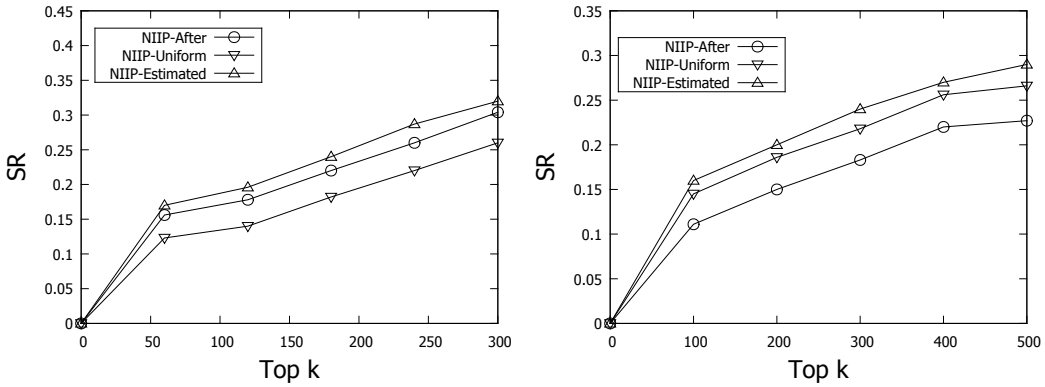
(a) Gowalla (b) Twitter

(c) Weibo (d) Foursquare

(e) Pig (f) Economy

(g) Oregon (h) MH370

Figure 5.9: Performance of different algorithms. $\tau = 10$ and $\alpha = 0.05$.



(a) Twitter **(b)** Foursquare

Figure 5.10: Performance of NIIP given different distribution of k. $\tau = 10$ and $\alpha = 0.05$.

5.4 Summary

In this section, we have modeled a realistic situation in real world contagions where healthy nodes have a certain chance of getting infected from sources outside the network and proposed the problem of node immunization over infectious period. We have shown that this problem is NP-hard in both arbitrary graphs and directed acyclic graphs. The S-NIIP algorithm has been developed to take into consideration the joint immunization effect of multiple nodes. We have also presented the NIIP algorithm that strategically immunized k nodes over a time period. Extensive experiments have been conducted and demonstrated conclusively that our algorithms outperform state-of-the-art algorithms significantly.

Chapter 6

Temporal Influence Blocking

We now navigate to address the second challenge in preventing misinformation propagation by considering time delays and deadline in information diffusion. In this chapter, we select k users as truth starters to actively propagate the truth information to combat the influence of a misinformation campaign. We formalize the *temporal influence blocking* problem [SHL17] and present our approach TIB-Solver to tackle this problem.

6.1 Problem Definition and Solution Overview

In this section, we formally define the TIB problem. Given a social network $G = (V, E)$ where V is a set of nodes denoting the users and E is a set of edges denoting the friend relationships. Each node $v \in V$ has a probability $login(v)$ to be online at each time point.

When a node $v \in V$ logs in at time point $t > 0$, its neighboring node u will have a probability $p(u, v)$ to influence v with rumor/truth if there is an edge from u to v, u is influenced by rumor/truth at time $t' < t$ and t is the first time v logs in after t'. If both rumor and truth get to influence a node v at the same time, then v will choose to believe the truth. Further, once a user has been influenced, s/he will not change her mind.

We say a node is *saved* by a truth campaign if in the absence of the truth campaign, this node will be influenced by the rumor. Under this setting, we define the problem of *Temporal Influence Blocking* (TIB) as follows:

Definition 8 (TIB problem)**.** *Let R be the set of rumor starters and Z be the set of nodes that start a truth campaign, $|Z| = k$. Let $\phi(R, Z, \alpha)$ denote the set of nodes influenced by R in the presence of the truth campaign started by Z before the deadline α. The TIB problem aims to find a set of k nodes, Z^*, that maximizes the expected number of saved nodes, that is,*

$$Z^* = argmax_Z \ (|\phi(R, \emptyset, \alpha)| - |\phi(R, Z, \alpha)|)$$

Note that $\phi(R, \emptyset, \alpha)$ corresponds to the set of nodes influenced by R only, that is, there is no truth campaign when $Z = \emptyset$.

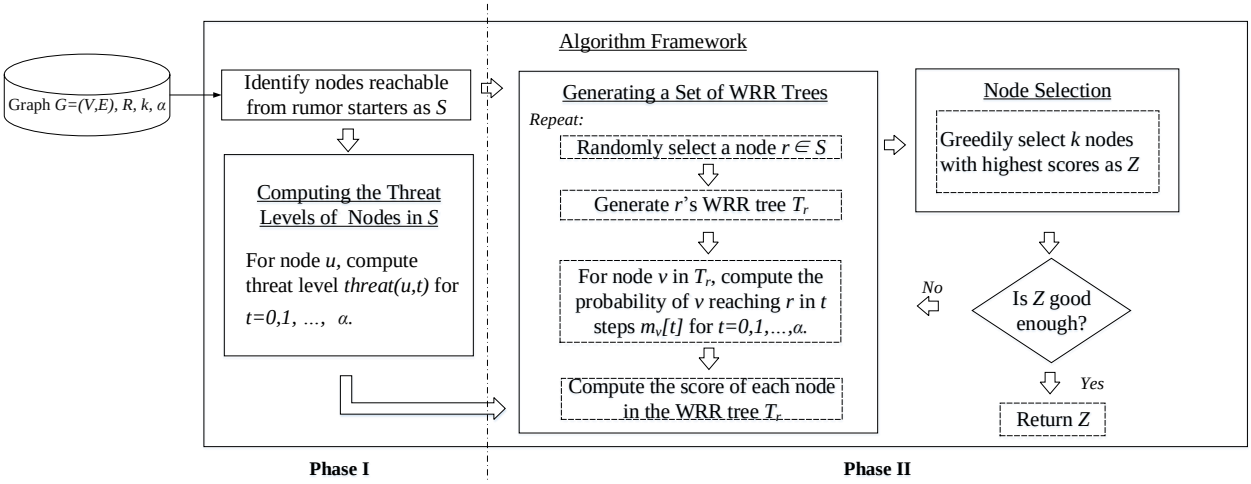


Figure 6.1: Framework of Proposed Approach

In order to determine Z^*, we need to answer the following questions:

- Among the nodes that could possibly be infected by a rumor if there is no truth campaign, which ones are likely to infect more other nodes?
- For the nodes that have been identified to be more likely to infect other nodes, can we find a set of nodes to reach these nodes earlier than R?

We design a solution that seamlessly incorporates login and influence probabilities to determine the top-k nodes to start the truth campaign. Figure 6.1 gives an overview of the proposed solution. There are two main phases:

- **Phase I.** This phase takes as input the social network and identifies all the nodes that could possibly be reached by rumor before the deadline. For each node, we compute its *threat level*, i.e., the potential number of nodes that this node could influence if itself is influenced.
- **Phase II.** In this phase, we adopt a sampling approach and generate *Weighted Reverse Reachable* (WRR) trees [SHL16] to estimate the time taken for a node to be reached by nodes in R. We greedily select k nodes that best save other nodes from rumor before the deadline.

The details of these two phases are described in the following sections.

6.2 Estimating Nodes' Threat Levels

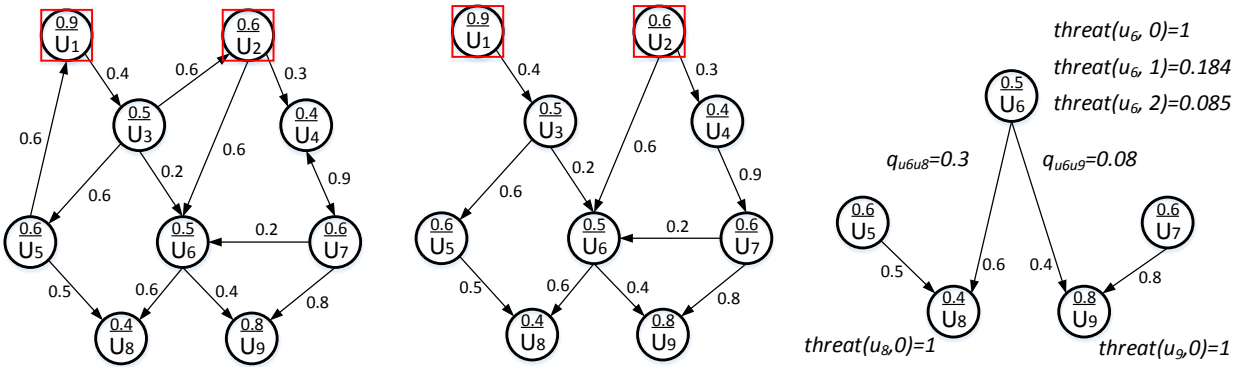
We define the *leverage* of a node u over its child node v, denoted as q_{uv}, as the probability that v is influenced by u only and is not influenced by the other parents of v. In other words, if Q is the set of parent nodes of v, then the leverage of u over v is given by:

$$q_{uv} = p(u,v) \cdot \prod_{w \in Q \setminus \{u\}} (1 - p(w,v)) \tag{6.1}$$

The *threat level* of a node u is dependent on the leverage of u over its child nodes as well as the time available for u to propagate its influence to its child nodes. Specifically, the threat of u increases when there is more time for u to propagate its influence. This is because if u is influenced way before the deadline α, then the child nodes that u have leverage over will have more time to influence other nodes.

In order to compute the threat levels of the nodes in a social network G given a set of rumor starters R and deadline α, we first perform a depth-first search from R until α steps and put visited nodes in a set S. We apply the Acyclic algorithm [EIL+12] on S to find the maximum directed acyclic graph (DAG) starting from R, as well as the topological ordering of the nodes in S such that the parents of a node precede it in the ordering.

An example network G with $R = \{u_1, u_2\}$ is shown in Figure 6.2a. The numbers on the edges indicate the influence probabilities while the numbers in the node indicate the login probabilities. Figure 6.2b shows the DAG obtained and a possible topological ordering of the nodes in S is $\langle u_3, u_4, u_5, u_7, u_6, u_8, u_9 \rangle$.



(a) Network G with login and influence probabilities.

(b) DAG obtained from G

(c) Threat levels for u_6

Figure 6.2: Example network and its DAG

We denote the *threat level* of a node u at time point t as $threat(u,t)$. Initially at time point 0, u only influences itself, hence the potential number of nodes influenced by u is 1. We write $threat(u,0) = 1$. At time point t, the threat level of u is determined by the nodes

that u influences directly in the first s time points as well as the nodes that are indirectly influenced through its child nodes from the time s to t. We have

$$threat(u,t) = \sum_{v \in C} \big(q_{uv} \cdot \sum_{s=1}^{t} \big(online(v,s) \cdot threat(v, t-s)\big)\big) \tag{6.2}$$

where C is the set of child nodes of u, and $online(v,s)$ is the probability that v will login at time point s, which is given by

$$online(v,s) = (1 - login(v))^{s-1} \times login(v)$$

Example 1. Let us consider the node u_6 in Figure 6.2b which has two child nodes u_8 and u_9. The leverage of u_6 over u_8 and u_9 are:

$$\begin{aligned} q_{u_6u_8} &= p(u_6,u_8) \times (1 - p(u_5,u_8)) \\ &= 0.6 \times (1 - 0.5) \\ &= 0.3 \\ q_{u_6u_9} &= p(u_6,u_9) \times (1 - p(u_7,u_9)) \\ &= 0.4 \times (1 - 0.8) \\ &= 0.08 \end{aligned}$$

The probabilities that u_8 and u_9 log in at time point 1 are given by:

$$\begin{aligned} online(u_8,1) &= (1 - login(u_8))^0 \times login(u_8) \\ &= 1 \times 0.4 = 0.4 \\ online(u_9,1) &= (1 - login(u_9))^0 \times login(u_9) \\ &= 1 \times 0.8 = 0.8 \end{aligned}$$

Then we can determine the threat level of u_6 at $t = 1$ as:

$$\begin{aligned} threat(u_6,1) &= q_{u_6u_8} \cdot online(u_8,1) \cdot threat(u_8,0) \\ &\quad + q_{u_6u_9} \cdot online(u_9,1) \cdot threat(u_9,0) \\ &= 0.184 \end{aligned}$$

Note that $threat(u_8,0) = 1$ and $threat(u_9,0) = 1$.

Figure 6.2c illustrates the threat levels computed for u_6.

Algorithm 6.1: Compute Threat Levels

input : 1. Graph $G = (V, E)$
2. Deadline α
3. Rumor starter R

output : Threat levels of nodes reachable by R

```
1  Initialize S = ∅;
2  foreach u ∈ R do
3  |  Peform DFS from u and insert visited nodes within α hops into S;
4  end
5  Apply Acyclic on S to generate a DAG and a topological ordering;
6  foreach node u in the reverse of topological ordering do
7  |  foreach child v of u do
8  |  |  Compute q_uv with Equation 6.1;
9  |  end
10 |  threat(u, 0) = 1;
11 |  for t = 1, ··· , α do
12 |  |  Compute threat(u, t) with Equation 6.2;
13 |  end
14 end
15 Return S and threat(u, t) for each u ∈ S where t = 0, 1, ··· , α.
```

Algorithm 6.1 gives the details of computing the threat levels of nodes that are reachable from rumor starters. The input is a social network $G(V, E)$ and a set of rumor starter nodes R. For each node in R, we perform a depth-first search to find the set of nodes S that are reachable within α hops, where α is the input deadline (Lines 1-3). We apply the Acyclic algorithm to obtain the DAG of S and the topological ordering (Lines 4). Then we traverse each node $u \in S$ in the topological ordering reversely. For each node u, we compute its leverage over its child v using Equation 6.1 (Lines 6 and 7). After that, we set $threat(u, 0) = 0$ (Line 8) and compute $threat(u, t)$ for $t = 1, \cdots, \alpha$ using Equation 6.2 (Lines 9 and 10). We return the threat levels for all nodes in S in Line 11.

6.3 Selecting Truth Seed Set

In this section, we first introduce a *weighted reverse reachable* tree structure to estimate which nodes can reach a particular node $v \in S$ faster than the rumor starters. Based on the threat level of v, we compute a score for each node that could possibly save v from being influenced by rumor. With a pool of random WRR trees, we greedily select k nodes with the highest scores among all trees.

6.3.1 WRR Tree Generation

The work in [BBCL14] introduces a *Reverse Influence Sampling*(RIS) algorithm to estimate the expected number of nodes that each node can influence in a network. RIS generates a set of Reverse Reachable (RR) sets by randomly sampling nodes in the graph. For each sampled node, it obtains an instance of the graph starting from this sampled node by following the in-edges of any visited node. An edge is kept with a probability that is equal to the influence probability on this edge. Each graph instance forms an RR set. RIS generates a sufficient number of RR sets and consider the nodes that appear in many RR sets with large influence [TSX15].

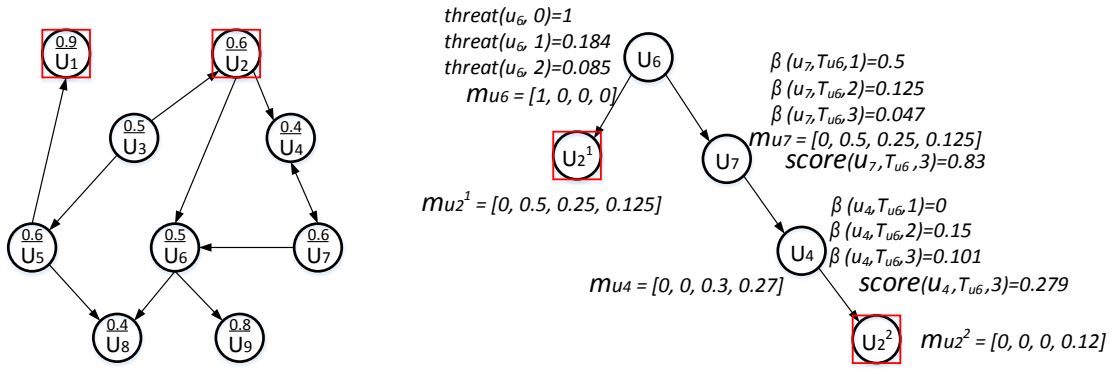
Song et al. [SHL16] extends the RR set to a WRR tree structure to estimate the probability of each node u in the tree reaching the root r at a particular time. We present the definition of WRR tree here.

Definition 9 (WRR Tree)**.** *Given a graph G, let g be a graph instance of G obtained by removing each edge $\langle u, v\rangle$ with probability $1 - p(u, v)$. Let α be the deadline. A WRR tree for a node r denoted as T_r, is an $(\alpha + 1)$-level tree such that each path $p \in T_r$ from r to a descendant v corresponds to a path from v to r in the graph instance g. Each node $v \in T_r$ is associated with a $(\alpha + 1)$-vector m_v where the j^{th} entry of m_v, denoted as $m_v[j]$, keeps track of the probability of v reaching the root in exactly j steps.*

The vector for the root node r is given by $m_r = [1, 0, \cdots, 0]$ as it takes exactly 0 steps for the root node to reach itself. Suppose v is at the d^{th} level in the WRR tree, it will take at least d hops for v to reach r as there are at least d nodes along the path from v to r. When $i < d$, the probability that v can reach r in exactly i steps is 0. When $i \geq d$, let node w be the immediate parent of v along the path from v to r, then the probability that v reaches r in exactly i steps is the probability that w logs in on the j^{th} step multiplied by the probability that w takes exactly $i - j$ steps to reach r, that is

$$m_v[i] = \begin{cases} 0 & i < d \\ \sum_{j=1}^{i} (online(w, j) \cdot m_v[i - j]) & otherwise \end{cases} \tag{6.3}$$

Given the graph G and set of rumor starters R, we create a graph instance g of G by flipping a coin for each edge $\langle u, v\rangle$ such that there is a probability $p(u, v)$ that the edge will be retained in g. With this graph instance, we can generate a WRR tree rooted at node r by performing a breadth-first traversal starting from r following the in-links. Each time we reach a node v, we create a corresponding node and add the node and its associated edge to the WRR tree. Note that if v has been visited before, a new copy of v is created. Since we are only interested in the instances where the root nodes can be infected by the rumor, we discard WRR trees that do not contain any node in R.



(a) Sampled graph instance g **(b)** WRR Tree T_{u_6}

Figure 6.3: Generation of a WRR tree

Example 2. Figure 6.3a shows a sampled instance g from the example network in Figure 6.2a. We can obtain a WRR tree as shown in Figure 6.3b. There are two paths from the rumor node u_2 to u_6, thus we create two copies of u_2 in the WRR tree, namely u_2^1 and u_2^2. We have

$$online(u_6, 1) = 0.5$$
$$online(u_6, 2) = (1 - login(u_6)) \times login(u_6)$$
$$= (1 - 0.5) \cdot 0.5 = 0.25$$

Then the vector entries for u_2^1 are computed as follows:

$$m_{u_2^1}[1] = online(u_6, 1) \times m_{u_6}[0] = 0.5$$
$$m_{u_2^1}[2] = online(u_6, 2) \times m_{u_6}[0] + online(u_6, 1) \times m_{u_6}[1] = 0.25$$

Note that $m_{u_6}[0] = 1$ and $m_{u_6}[1] = 0$.

Next, we want to determine the probability of a node u reaching the root node r of a WRR tree at time point t before any rumor starter. Let $\beta(u, T_r, t)$ denote this probability, and $R' \subset R$ be the set of rumor starters in T_r. We have the following cases:

1. u is a descendant of some rumor starter $w \in R'$. In this case, $\beta(u, T_r, t) = 0$ since w will always reach r before u.

2. All the nodes in R' are descendants of u. Since u will always reach r before any rumor node, $\beta(u, T_r, t)$ equals the probability of u reaching the root at time step t, *i.e.*$\beta(u, T_r, t) = m_u[t]$.

3. $\exists\ R_u \subseteq R'$ s.t. the nodes in R_u are not ancestors or descendants of u, $R_u \neq \emptyset$. Then we need to compute $\beta(u, T_r, t)$ as follows:

   Probability that r is not reached by some $w \in R'$ from time 0 to $t-1$ is $\prod_{s=0}^{t-1}(1-m_w[s])$.

   Probability that r is not reached by *any* $w \in R_u$ from time 0 to $t-1$ is $\prod_{w\in R_u}\prod_{s=0}^{t-1}(1-m_w[s])$.

   Thus we have
   $$\beta(u, T_r, t) = m_u[t] \times \prod_{w\in R_u}(\prod_{j=0}^{t-1}(1 - m_w[j])) \tag{6.4}$$

If u reaches the root r of a WRR tree at time point t before any rumor starter in R', then all the nodes in G that could potentially be influenced by r will be saved. Recall $threat(r, t)$ gives the expected number of nodes that r could influence at time point t. Then the total number of nodes that r could influence from time t to α is given by $\sum_{s=0}^{\alpha-t} threat(r, s)$. With this, we can compute a score for each node u indicating the expected number of nodes that could be saved by u before a deadline α as follows:

$$score(u, T_r, \alpha) = \sum_{t=1}^{\alpha}\Big(\beta(u, T_r, t) \cdot \sum_{j=0}^{\alpha-t} threat(u, j)\Big) \tag{6.5}$$

Example 3. Consider again the WRR tree in Figure 6.3b. Suppose $\alpha = 3$, and we want to compute the score of u_7. Since u_2^2 is a descendant of u_7, we only need to consider u_2^1 in the computation of $\beta(u_7, T_{u_6}, 1)$.

When $t = 1$:

$\beta(u_7, T_{u_6}, 1) = m_{u_7}[1]\cdot(1-m_{u_2^1}[0]) = 0.5$

$score(u_7, T_{u_6}, 1) = \sum_{s=0}^{2} threat(u, s)\cdot\beta(u_7, T_{u_6}, 1) = 0.635$

When $i = 2$:

$\beta(u_7, T_{u_6}, 2) = m_{u_7}[2] \cdot \prod_{k=0}^{1}(1-m_{u_2^1}[k]) = 0.125$

$score(u_7, T_{u_6}, 2) = score(u_7, T_{u_6}, 1) + \sum_{s=0}^{1} threat(u, s) \cdot \beta(u_7, T_{u_6}, 2) = 0.783$

When $i = 3$:

$\beta(u_7, T_{u_6}, 3) = m_{u_7}[3] \cdot \prod_{k=0}^{2}(1-m_{u_2^1}[k]) = 0.047$

$score(u_7, T_{u_6}, 3) = score(u_7, T_{u_6}, 2) + \sum_{s=0}^{0} threat(u, s) \cdot \beta(u_7, T_{u_6}, 3) = 0.83$

Hence the score of u_7 in the WRR tree is 0.83. Similarly, we can compute the score for u_4 as shown in Figure 6.3b.

Algorithm 6.2 gives the details for generating a single WRR tree and computing the scores for the corresponding nodes. We first randomly sample a node from the node set S as the tree root. We start a breadth-first search from r and flip a coin for each in-link to decide whether to include this link in the tree. If a node is added in the tree, we compute its influencing time vector in Line 6 and check whether it is a rumor starter in R. If the node is not a rumor starter, we continue to add its adjacent in-neighbors to the tree if the breadth-first traversal is within α levels from r (Lines 8-14). However, if it is a rumor starter, then we terminate this branch of traversal and for any of its ancestor node v, we update the set R_v by excluding u (Lines 16-17). After constructing the WRR tree T_r, we check if any rumor node is contained in the tree (Line 18). If yes, we compute the score for each node in T_r. For a node u whose $R_u = \emptyset$, it means u is an ancestor of all rumor nodes, thus we set $\beta(u, T_r, t) = m_u[t]$ (Lines 21 - 22). Otherwise, we compute $\beta(u, T_r, t)$ with Equation 6.4 (Lines 24 and 25). Finally we compute the score of u in Line 23 and return the WRR tree.

6.3.2 Node Selection

We adopt the D-SSA algorithm [NTD] to generate a sufficient pool of random WRR trees. The $score(u, T_r, \alpha)$ estimates the potential number of nodes u can save given a WRR tree rooted at r. We repeatedly select the node u with the highest score in the pool of randomly sampled WRR trees. If u has been selected as a truth starter, the *threat level* of u's parent nodes would decrease since they cannot influence u any more. If u's parent nodes happen to be the root of some WRR tree, T_r, then the threat level of the root r is updated as follows:

$$threat(r, t) = threat(r, t) - q_{ru} \cdot (\sum_{s=1}^{t} online(u, s) \cdot threat(u, t - s)) \tag{6.6}$$

With the new threat level of root r, we update the scores of nodes in T_r with Equation 6.5.

Example 4. Figure 6.4a illustrates the change in the threat level of u_6 when u_9 is selected as truth starter (labeled in green box). Once u_9 has been selected as truth starter at time point 0, u_6 can no longer influence u_9. Hence, the threat level of u_6 at time point 1 and 2 are reduced accordingly. We have:

$$\begin{aligned} threat(u_6, 1) &= threat(u_6, 1) - q_{u_6 u_9} \cdot online(u_9, 1) \cdot threat(u_9, 0) \\ &= 0.12 \end{aligned}$$

$$\begin{aligned} threat(u_6, 2) &= threat(u_6, 2) - q_{u_6 u_9} \cdot online(u_9, 2) \cdot threat(u_9, 0) \\ &= 0.072 \end{aligned}$$

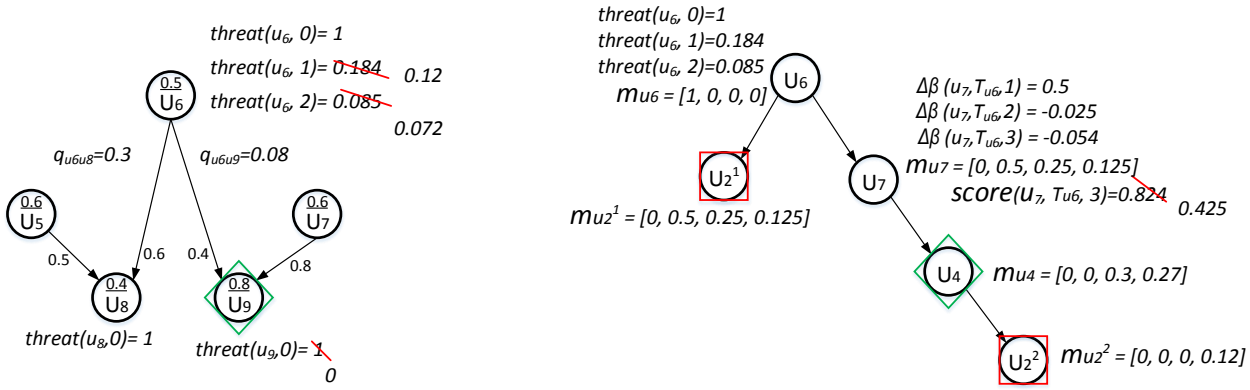
With the new threat level of u_6, we update the scores of the nodes in all WRR trees whose root is u_6.

Algorithm 6.2: Generate a WRR tree

input : 1. Graph $G = (V, E)$
2. Deadline α
3. Rumor starters R
4. Nodes reachable from rumor starters S
5. Threat level $threat(u, t)$ for each $u \in S$

output : A WRR tree

Initialize processing queue $A = \emptyset$;
Randomly choose a node $r \in S$;
$A.enqueue(r)$;
while $A \neq \emptyset$ **do**
  $u = A.dequeue()$;
  Compute m_u with Equation 6.3;
  if $u \notin R$ **then**
    if *breadth-first search is within* α *levels of* r **then**
      foreach *in-link* $\langle v, u \rangle$ **do**
        Flip a coin with probability $p(v, u)$;
        if *decision is YES* **then**
          Create a copy of v;
          $A.enqueue(v)$;
          Initialize $R_v = R$;
      end
    end
  else
    foreach *node* v *on the path from* r *to* u **do**
      $R_v = R_v \setminus \{u\}$;
    end
end
if T_r *contains any node in* R **then**
  foreach *node* u *in the tree* **do**
    if $R_u = \emptyset$ **then**
      for $i = 0, 1, \cdots, \alpha$ **do**
        $\beta(u, T_r, t) = m_u[t]$;
      end
    else
      for $i = 0, 1, \cdots, \alpha$ **do**
        Compute $\beta(u, T_r, t)$ with Equation 6.4;
      end
    Compute $score(u, T_r, \alpha)$ with Equation 6.5;
  end
  Return the generated WRR tree.
else
  Return **void**;



(a) Updated DAG **(b)** Updated WRR Tree

Figure 6.4: Updating node scores

The second scenario is where the selected truth starter node u is a node in some WRR trees. In this case, the scores of the nodes in these WRR trees may change. This is because if u is selected as a truth starter, the node v in the WRR trees not only needs to reach the root nodes before any rumor starters, but it must also do so before u. Otherwise, r would have already been saved by u, rendering the selection of v as an additional truth starter pointless. We model this as:

$$\Delta\beta(v, T_r, t) = \beta(Z \cup \{v\}, T_r, t) - \beta(Z, T_r, t) \tag{6.7}$$

For each WRR tree, T_r where u is a node in T_r, we update the scores of the nodes in T_r by replacing $\beta(u, T_r, t)$ with $\Delta\beta(u, T_r, t)$ in Equation 6.5.

Example 5. Figure 6.4b illustrates how the score of u_7 is updated after u_4 has been selected as a truth starter. We first compute $\Delta\beta(u_7, T_{u_6}, t)$ for $t = 1, 2$ and 3.
When $t = 1$:
$\beta(\{u_4, u_7\}, T_{u_6}, 1) = 0.5$
$\beta(u_4, T_{u_6}, 1) = 0$
$\Delta\beta(u_7, T_{u_6}, 1) = beta(\{u_4, u_7\}, T_{u_6}, 1) - \beta(u_4, T_{u_6}, 1) = 0.5.$
When $t = 2$:
$\beta(\{u_4, u_7\}, T_{u_6}, 2) = 0.125$
$\beta(u_4, T_{u_6}, 2) = 0.15$
$\Delta\beta(u_7, T_{u_6}, 2) = \beta(\{u_4, u_7\}, T_{u_6}, 2) - \beta(u_4, T_{u_6}, 2) = -0.025.$
Note that if we select u_7 as a truth starter, it is likely to reach the root u_6 at time point 1. Hence the probability of saving u_6 at time point 2 is reduced, resulting in a negative value of $\Delta\beta(u_7, T_{u_6}, 2)$.

When $t = 3$:

$\beta(\{u_4, u_7\}, T_{u_6}, 3) = 0.047$

$\beta(u_4, T_{u_6}, 3) = 0.101$

$\Delta\beta(u_7, T_{u_6}, 2) = \beta(\{u_4, u_7\}, T_{u_6}, 3) - \beta(u_4, T_{u_6}, 3)$

$= -0.054.$

Finally, we compute a new score for u_7 by replacing $\beta(u_7, T_{u_6}, t)$ with $\Delta\beta(u_7, T_{u_6}, t)$ in Equation 6.5 as

$$score(u_7, T_{u_6}, 3) = 0.5 \times 1 + (-0.025) \times 0.184 + (-0.054) \times 0.085$$
$$= 0.425$$

Algorithm 6.3 gives the details for selecting k nodes among a set of sampled WRR trees. Firstly, we generate a sufficent set of random WRR trees by calling the D-SSA algorithm [NTD] (Line 1). We then select a node u with the highest score among all WRR trees in Lines 4-5. Since the selection of u decreases the threat levels of its parent nodes, we update the threat levels of these parent nodes and recompute the score for all the nodes in the WRR tree rooted at any of u's parents (Lines 7-11). Lines 12-17 update the scores of nodes in the WRR trees where u is present. After the scores have been updated, we proceed to select the next node with the highest score until k nodes have been selected.

6.4 Experiments

In this section, we evaluate the performance of our TIB-Solver algorithm on four real-world datasets. All the experiments are carried out on a Linux machine with 2 Xeon E5440 2.83 GHz CPU and 16G Ram. We use the four real-world social network datasets described in Section 3.3, *i.e.*Gowalla, Twitter, Weibo and Foursquare.

Evaluation Metric. We measure the effectiveness of different methods using a metric called *Save Ratio* (SR) [SHL15b] which gives the percentage of reduction in the number of nodes infected by R after we start a truth campaign from Z in α time points:

$$SR(Z) = \frac{|\phi(R, \emptyset, \alpha)| - |\phi(R, Z, \alpha)|}{|\phi(R, \emptyset, \alpha)|}$$

where $\phi(R, Z, \alpha)$ is the set of influenced users by the R before deadline α when truth campaign starts from Z under our diffusion model with login events.

Algorithm 6.3: TIB-Solver

input : 1. Number of nodes to select k
2. Deadline α
3. Nodes reachable from rumor starters S
4. Threat level $threat(u, t)$ for $u \in S$

output : Set of truth starters Z

Call D-SSA to generate a pool of WRR trees $\mathcal{T}$;
Initialize $Z = \emptyset$
repeat
  Let u be the node with the highest $score(u, T_r, \alpha)$
  $Z = Z \cup \{u\}$;
  foreach *parent* v *of* u **do**
    for $t = 0, \cdots, \alpha$ **do**
      Compute $threat(v, t)$ with Equation 6.6;
    end
    foreach *WRR tree* T_v **do**
      foreach w *in* T_v **do**
        Compute $score(w, T_v, \alpha)$ with Equation 6.5;
      end
    end
  end
  foreach $T_r \in \mathcal{T}$ *involving* u **do**
    Set $score(u, T_r, \alpha) = 0$
    foreach *node* w *in* T_r **do**
      for $t = 1 \cdots, \alpha$ **do**
        Compute $\Delta\beta(w, T_r, t)$ with Equation 6.7;
      end
      Compute $score(w, T_r, \alpha)$ with Equation 6.5;
    end
  end
until $|Z| = k$
Return Z

Settings. We set the login probability of each node by randomly choosing a real number from the interval $[0, 1]$. Similar to [SHL16, NTD], we set the influence probability $p(u, v)$ for each edge $\langle u, v\rangle$ as $1/inDegree(v)$ where $inDegree(v)$ is the number of incoming edges to node v. We randomly choose 30 rumor seed nodes and repeat each set of experiments 5 times on each dataset. We measure the reduction in the number of infected nodes after 20,000 Monte-Carlo simulations.

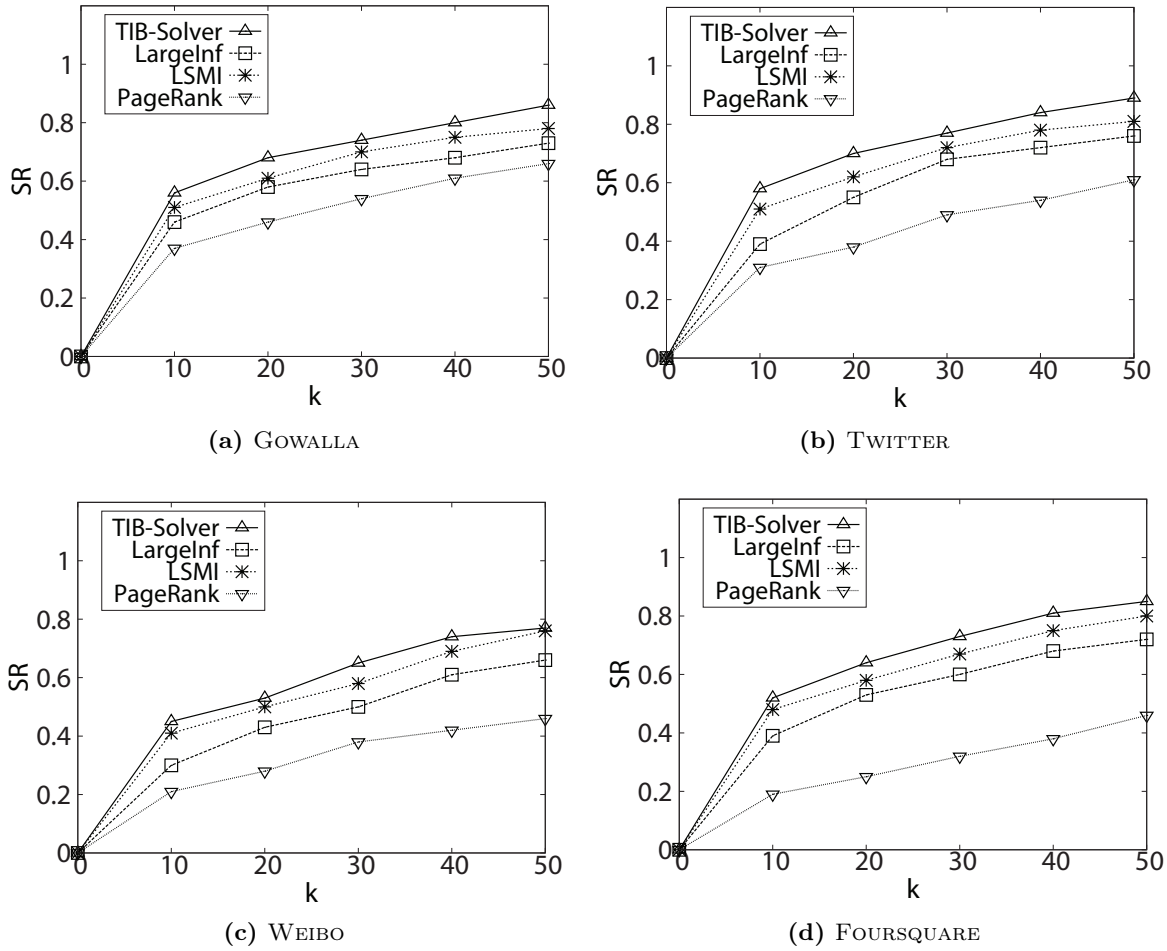
6.4.1 Comparative Experiments

In this set of experiments, we compare the performance TIB-Solver with the following methods:

- PageRank [PBMW98]. This approach selects nodes with the highest PageRank scores regardless of the temporal information.
- LSMI [TNT12]. This algorithm estimates the influence of each node based on local shortest paths and selects the nodes with high local influence given rumor starters. We extend this approach by computing an edge length from the login probabilities and call it LSMI-Delay.
- LargeInf [BAEA11]. This method adopts a simulation approach to estimate the influence of nodes reachable by the rumor starters and selects the nodes with the highest influence. We extend this method by running simulations in our multi-campaign independent cascade model with login events and deadline. We call the extended method LargeInf-L.

Results with $login(u) = 1$ and $\alpha = \infty$. In this set of experiments, we set the login probability of each node to be 1 and assume there is no deadline. This setting reduces our problem to the EIL problem proposed in [BAEA11]. Figure 6.5 shows the performances of the various algorithms. In all datasets, TIB-Solver gives the best performance.

LSMI is a close runner up in this setting by using the shortest paths to estimate the selected nodes' influence among the nodes that are reachable from the given rumor nodes. Since the login probability is 1, the length of a path between two nodes equals the number of time points it takes for these nodes to reach each other via this path. Although LSMI is efficient as it only considers shortest paths, it may overlook longer paths with high influence probabilities. In contrast, the TIB-Solver is able to take into consideration paths with high influence probabilities as such paths are more likely to be sampled in the WRR tree generation process.



(a) Gowalla **(b)** Twitter

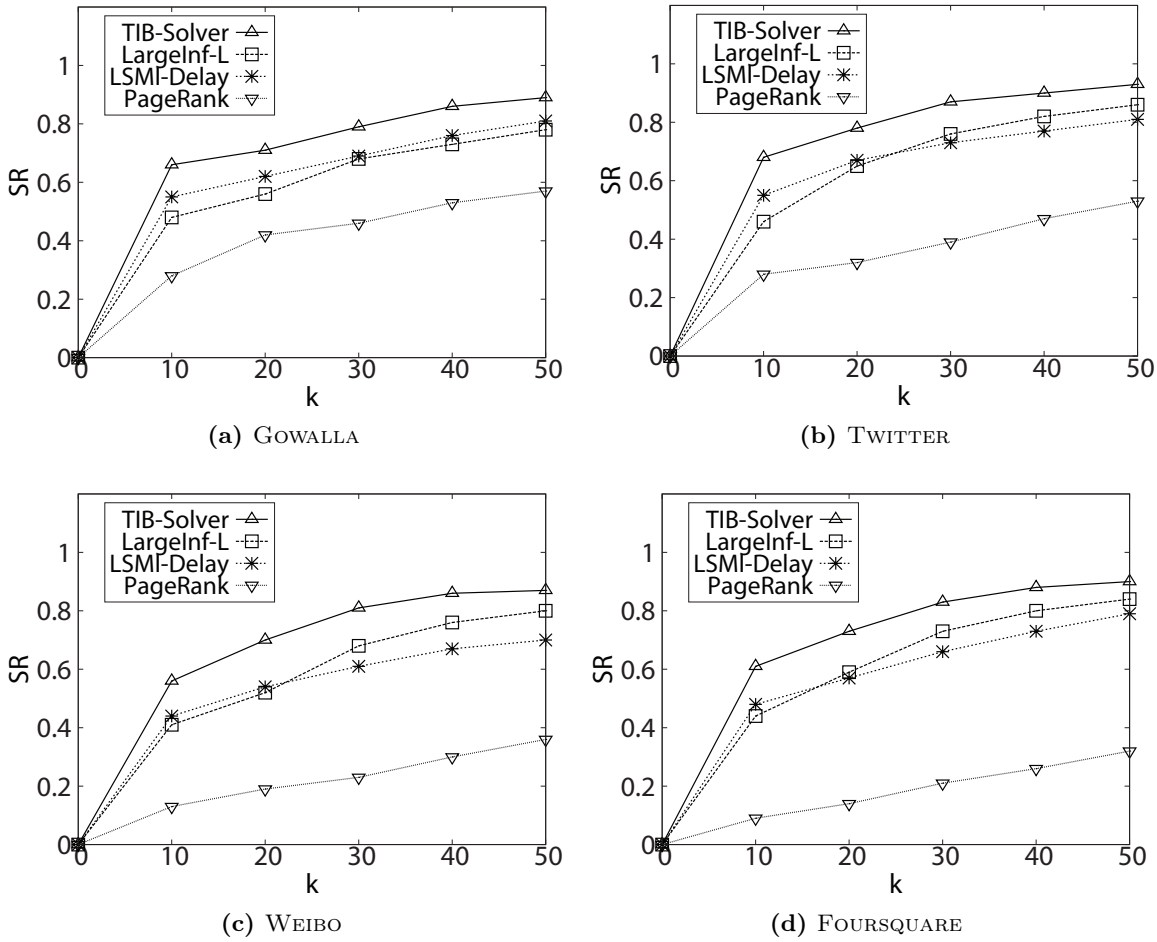
(c) Weibo **(d)** Foursquare

Figure 6.5: Save Ratio as k varies (login=1, $\alpha = \infty$).

LargeInf performs slightly poorer than LSMI and TIB-Solver because it considers only nodes that can be reached by the rumor starters. This strategy may miss nodes that are not reachable by rumor starters, and yet can save the most number of nodes. PageRank gives the poorest performance in all four datasets since it does not consider which nodes are the rumor starters.

Results with random $login(u)$ and $\alpha = \infty$. In this set of experiments, we randomly choose a real value in the interval $[0, 1]$ to be the login probability of each node but assume there is no deadline.

Figure 6.6 shows the results. We observe that TIB-Solver gives the best performance. In particular, when $k = 10$, TIB-Solver outperforms the closest runner-up LSMI-Delay by more than 20% in all datasets. In order to incorporate time delays into computation, LSMI-Delay calculates the length of each edge $\langle u, v\rangle$ as $\sum_{i=1}^{\infty}(i\cdot(1-login(v))^{i-1}\cdot login(v))$ which converges to $\frac{1}{login(v)}$. However, this expected delay does not capture the influence propagation at each time point, leading to a solution that is not as good as TIB-Solver.



(a) Gowalla **(b)** Twitter

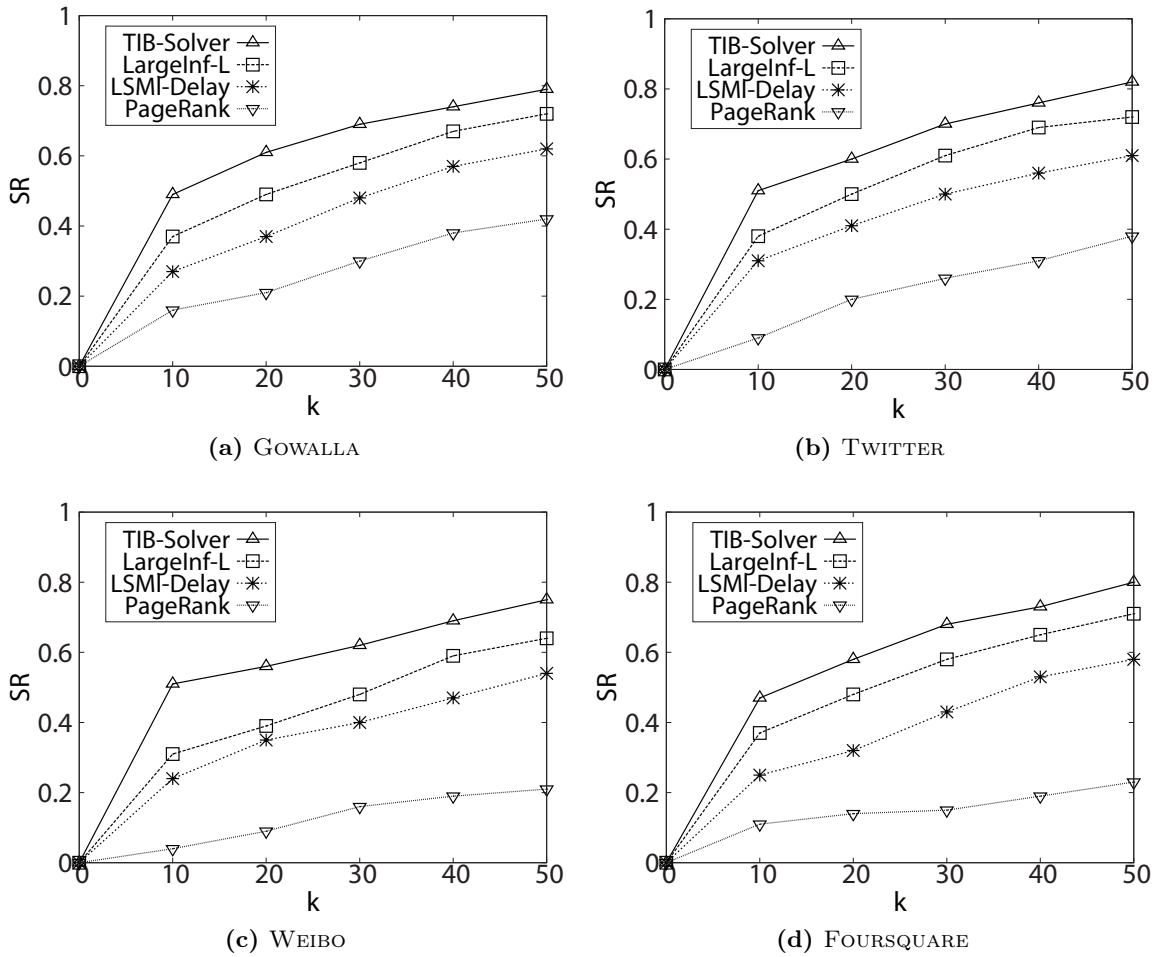
(c) Weibo **(d)** Foursquare

Figure 6.6: Save Ratio as k varies (random login, $\alpha = \infty$).

Since LargeInf-L only considers the nodes reachable from the rumor starters to start a truth campaign, it does not perform as well as TIB-Solver which considers all the nodes. PageRank has the poorest performance as it simply selects high connectivity nodes with no consideration of the rumor starters or login probabilities.

Results with random $login(u)$ **and** $\alpha = 10$**.** In this set of experiments, we set the login probability of each node randomly and assign a deadline of 10. In other words, we determine the number of saved nodes within 10 time points from the start of rumor propagation. Figure 6.7 shows the results. Once again, TIB-Solver gives the best performance with a given deadline. This is particularly evident in datasets with high density of edges. In Weibo dataset where the average degree of nodes is the highest, TIB-Solver outperforms the runner up method over 55% when $k = 10$ and 22% when $k = 50$. This is because TIB-Solver aims to find nodes that are connected to high threat nodes. In high density networks, such high threats nodes are more likely to be reached within the deadline, leading to more nodes being saved before the deadline.

LSMI-Delay performs poorly because when we set a deadline α, any path with expected delay longer than α value will not be considered by LSMI-Delay. Hence, its performance drops significantly when there is a deadline. Also note that, PageRank performs much poorer than the other methods even in Gowalla because, with a restricted time for propagation, the nodes selected by PageRank may not be able save any nodes before the deadline.

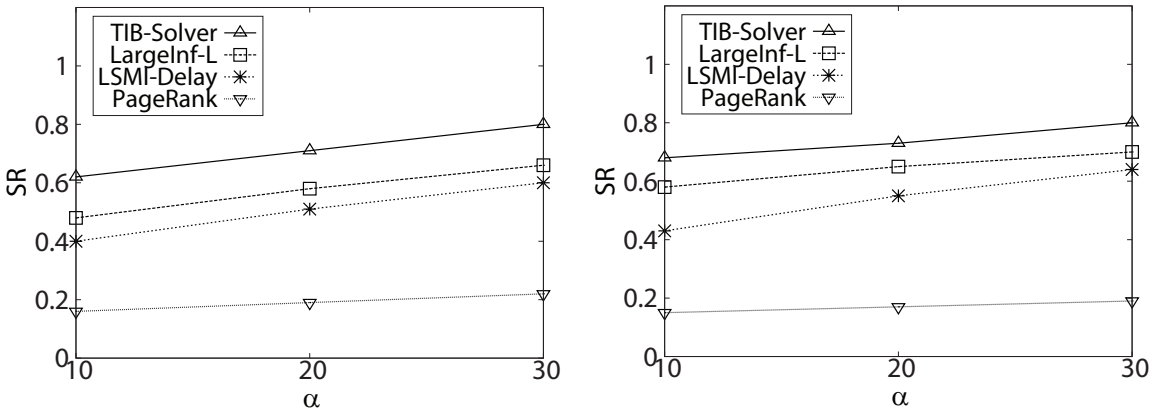


(a) Gowalla **(b)** Twitter

(c) Weibo **(d)** Foursquare

Figure 6.7: Save Ratio as k varies (random login, $\alpha = 10$).

We further fix $k = 30$ and vary the deadline α from 10 to 30 on the two larger datasets Weibo and Foursquare. Figure 6.8 shows the results. We see that the performance of all methods tend to improve slightly with the increase of α and TIB-Solver consistently gives the best results for all α values.



(a) Weibo **(b)** Foursquare

Figure 6.8: Save Ratio as α varies (random login, $k = 30$).

6.4.2 Sensitivity Experiments

Recall that TIB-Solver computes the *threat levels* of each node in Phase I and the m_u *vectors* in Phase II to estimate the influencing time. In this set of experiments, we examine the effect of *threat level* and the m_u vectors on the save ratio SR respectively. We compare the performance of TIB-Solver with the following variants:

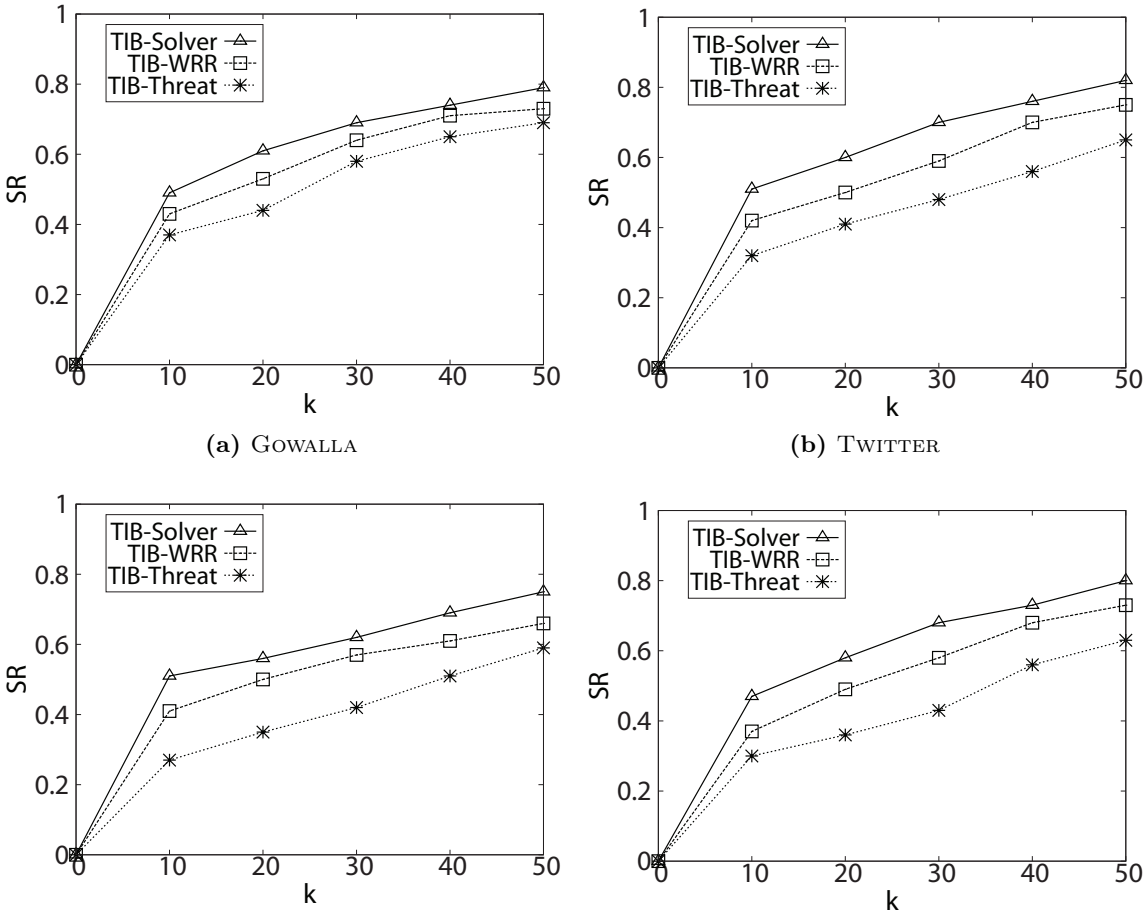
- TIB-Threat. We compute the threat levels $threat(u, t)$ and set $m_u[j] = 1$ where j is the path length from u to root r and $m_u[i] = 0$ where $i = 0, \cdots, \alpha$ and $i \neq j$.

- TIB-WRR. We set $threat(u, t) = 1$ for all $u \in S$ before generating the WRR trees to compute m_u.

Figure 6.9 shows the results when we have random login probabilities and $\alpha = 10$. We see that TIB-Threat performs worse that TIB-WRR, indicating that the estimation of the influencing time is more important than determining the threat level of nodes. This is because without knowing the influencing time, the selected truth seed set may reach nodes after the rumor starters have reached them.

6.4.3 Efficiency

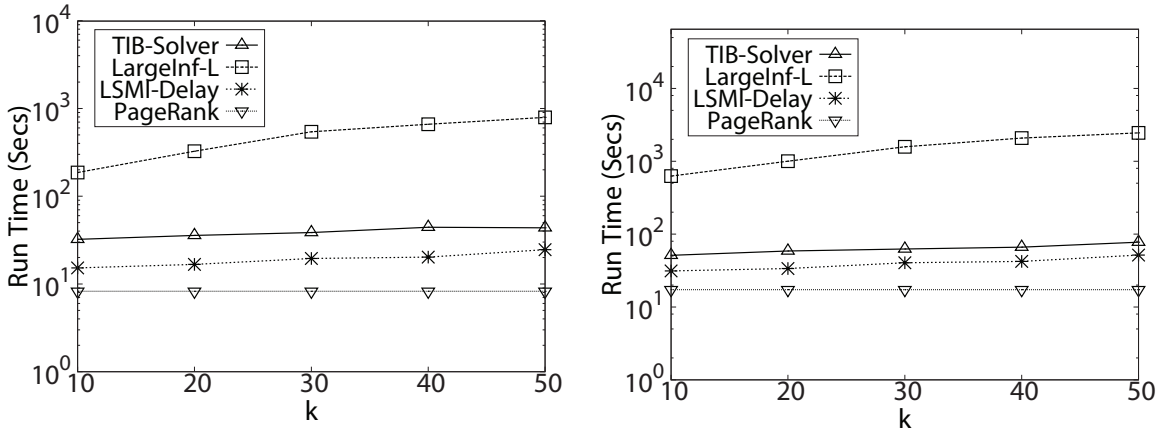
Finally, we report the runtime of the different approaches on the two larger datasets, Weibo and Foursquare. Figure 6.10 shows the results when we fix $\alpha = 10$ and vary k from 10 to



(a) Gowalla (b) Twitter

(c) Weibo (d) Foursquare

Figure 6.9: Save Ratio of TIB-Solver and its variants (random login, $\alpha = 10$).



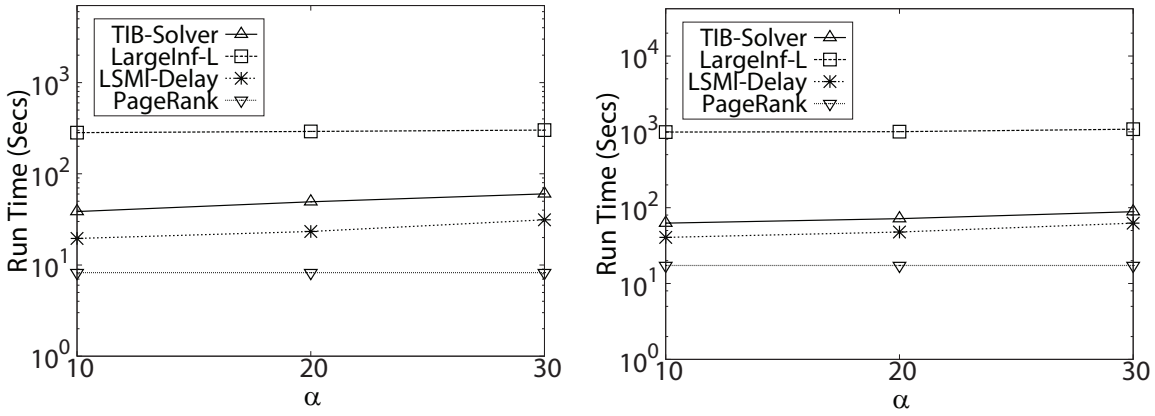
(a) Weibo (b) Foursquare

Figure 6.10: Runtime as k varies (random login, $\alpha = 10$).

50. We observe that LargeInf-L has the longest computation time among all methods. This is because in order to select one node, LargeInf-L needs to run a large number of simulations to see how many nodes the selected node can influence others when it has been infected by rumor. Further, the selection of one node greatly affects the propagation of rumor: after selecting each node, it has to re-run the simulations to select the second node. Hence as k increases, we can see a clear increase in the runtime for LargeInf-L.

LSMI-Delay is slightly faster than TIB-Solver since it considers the local structure of each node and only focuses on the shortest paths. PageRank gives the lowest computation time as it simply utilizes the topology and ignores the login probabilities and rumor starters.

Figure 6.11 shows the runtime results when we fix $k = 30$ and vary α from 10 to 30. When α increases, the computation time for each method only increases slightly. Again, we find LargeInf-L has the highest runtime while TIB-Solver and LSMI-Delay has similar performance (within 100 seconds).



(a) Weibo **(b)** Foursquare

Figure 6.11: Runtime as α varies (random login, $k = 30$).

6.5 Summary

In this chapter, we have incorporated the login events in the multi-campaign Independent Cascade model to simulate real-world information propagations with time delays. We have formally defined the *Temporal Influence Blocking* problem which aims to find the set of users to start a truth campaign so as to minimize the number of users affected by rumors. We have proposed a 2-phase approach called TIB-Solver that measures the threat levels of each node and estimate the potential number of nodes that could be saved by a node if this node is selected as truth starter. Extensive experimental studies have been conducted to show the effectiveness and efficiency of our approach.

Chapter 7

Conclusion and Future work

7.1 Conclusion

Social networks nowadays have become the major platform for users to read news and exchange information. Studies have shown that the diffusion of information on social platforms is faster and wider than traditional information platforms like newspaper and television. The research on information diffusion in online social networks is valuable in both theoretical and practical perspectives. On one hand, researchers design diffusion models to simulate the diffusion of information in social platforms. On the other hand, with the diffusion models, researchers can identify critical users that can aid in real-world applications like viral marketing and rumor prevention. In this thesis, we seek to address the research challenges in managing information diffusion given different incentives, *i.e.*boosting good information propagation and limiting misinformation spreading.

We started with the exploration to expand information diffusion in social networks when we consider the user locations and deadline for propagation. We have extended the Independent Cascade model by introducing a login event to incorporate diffusion delays and form the problem of targeted influence maximization. We have designed a weighted reverse reachable tree structure and developed a sampling based algorithm that considers the event location and deadline.

We have further considered, instead of influencing users in a targeted region, propagating information to nodes that are far away from the initial seeds, which users can help us spread this information? This question brought us to the problem of identifying social network brokers that enable the dissemination of information between otherwise disconnected or remote users. We have designed a heuristic solution to find top-k brokers based on the weak tie theory and refined the algorithms to handle dynamic updates in social networks. Utilizing the detected brokers in mention recommendation task significantly increases the number of distinct users reached.

Next, we explored the approaches to control the diffusion of misinformation in social networks. In node immunization problem, existing works mostly focused on immunizing nodes before rumor starts spreading or consider all the infectious nodes are known. We realized

in real-world contagions, there is a time period where multiple independent sources may actively infect different nodes. We have formalized the problem of node immunization over infectious period and designed a scoring function to model a node's immunization ability in the network. We have also proposed a simulation-based strategy to estimate the distribution of immunization resources over the infectious period.

In the influence blocking problem where we can select a set of users to broadcast the truth information to combat the spreading of misinformation, we have recognized the temporal effect is of crucial importance since truth must reach a node earlier than the rumor in order to save it from infection. We incorporated time delays in the diffusion model and form the temporal influence blocking problem. We presentd a sampling and greedy-based solution that utilizes the technique we developed for estimating a node's immunization ability and the weighted reverse reachable tree for measuring the influencing time. Extensive experiments show that our proposed solution is able to significantly increase the percentage of saved nodes in a rumor attack.

7.2 Future Work

There are several directions that could be further investigated for managing information diffusion in social platforms more effectively. We list three directions for future work.

Incorporating User Generated Content. The rich content data generated by users' social behaviors, such as messages, replies or posts describe the users' hobbies, topics of interest, liked products or trusted news websites both explicitly and implicitly. The techniques for understanding the contents could be incorporated in our proposed algorithms when we identify influential spreaders or prevent misinformation diffusion since different users possess different probability of liking or sharing the same information because of their various interests.

Parallel Processing. With the increasing volume of users on social platforms, scalability becomes a huge challenge for network algorithms. One possible solution is to design effective algorithms to divide the input graph into multiple small networks while maintaining the necessary information and utilize parallelized framework such as MapReduce to allocate the computation tasks and combine them to form a unified solution.

Handling Uncertainty. We have assumed that we are fully aware of the network (users, edges and influence probabilities) and the initial seed users for misinformation prevention tasks. However in the real-world, it is unlikely that we can obtain the entire network structure or identify all the initial spreaders. This requires us to predict the existence of a user/edge or if a node is infected or not while we identify the critical nodes for boosting/limiting influence.

References

[AH11] Zeinab Abbassi and Hoda Heidari. Toward optimal vaccination strategies for probabilistic models. In *WWW*, pages 1–2, 2011.

[AHI02] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

[BAEA11] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *WWW*, pages 665–674, 2011.

[BAH12] Roja Bandari, Sitaram Asur, and Bernardo A. Huberman. The pulse of news in social media: Forecasting popularity. *CoRR*, abs/1202.0332, 2012.

[BBCL14] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *SIAM SODA*, pages 946–957, 2014.

[BFO10] Allan Borodin, Yuval Filmus, and Joel Oren. Threshold models for competitive influence in social networks. In *WINE*, pages 539–550, 2010.

[BKS07] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *Springer WINE*, pages 306–311, 2007.

[BLP03] Linda Briesemeister, Patrick Lincoln, and Phillip Porras. Epidemic profiles and defense of scale-free networks. In *WORM*, pages 67–75, 2003.

[BRMA12] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.

[Bur07] Ronald S. Burt. Secondhand brokerage: Evidence on the importance of local structure for managers, bankers, and analysts. *Academy of Management Journal*, 50:119–148, 2007.

[CCRea11] Wei Chen, Alex Collins, and Cummings Rachel et al. Influence maximization in social networks when negative opinions may emerge and propagate. In *SIAM SDM*, pages 379–390, 2011.

[CDCS10] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. Emerging topic detection on twitter based on temporal and social terms evaluation. In *ACM MDMKDD*, page 4, 2010.

[CDK10] Po-An Chen, Mary David, and David Kempe. Better vaccination strategies for better people. In *ACM EC*, pages 179–188, 2010.

[CDPW14] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Timed influence: Computation and maximization. *CoRR*, abs/1410.6976, 2014.

[CFL+15] Shuo Chen, Ju Fan, Guoliang Li, Jianhua Feng, Kian-lee Tan, and Jinhui Tang. Online topic-aware influence maximization. *PVLDB*, 8(6):666–677, 2015.

[CGD12] B. Chandramouli, J. Goldstein, and S. Duan. Temporal analytics on big data for web advertising. In *IEEE ICDE*, pages 90–101, 2012.

[CHbA03] Reuven Cohen, Shlomo Havlin, and Daniel ben Avraham. Efficient immunization strategies for computer networks and populations. *Physics Revivew Letter*, 91:247901, 2003.

[Che03] Jing-Chao Chen. Dijkstra's shortest path algorithm. *Journal of Formalized Mathematics*, 15:144–157, 2003.

[CLZ12] Wei Chen, Wei Lu, and Ning Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. *CoRR*, abs/1204.3074, 2012.

[CSH+14] Suqi Cheng, Huawei Shen, Junming Huang, Wei Chen, and Xueqi Cheng. Imrank: Influence maximization via finding self-consistent ranking. In *ACM SIGIR*, pages 475–484, 2014.

[CWL+11] Peng Cui, Fei Wang, Shaowei Liu, Mingdong Ou, Shiqiang Yang, and Lifeng Sun. Who should share what?: Item-level social influence prediction for users and posts ranking. In *ACM SIGIR*, pages 185–194, 2011.

[CWW+08] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic thresholds in real networks. *Transactions on Information and System Security*, 10:1:1–1:26, 2008.

[CWW10] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *ACM KDD*, pages 1029–1038, 2010.

[CWY09] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *ACM KDD*, pages 199–208, 2009.

[CYZ10] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *IEEE ICDM*, pages 88–97, 2010.

[Dan06] Chavdar Dangalchev. Residual closeness in networks. *Physica A: Statistical Mechanics and its Applications*, 365(2):556–564, 2006.

[DCLS+10] Munmun De Choudhury, Yu-Ru Lin, Hari Sundaram, K Selcuk Candan, Lexing Xie, and Aisling Kelliher. How does the data sampling strategy impact the discovery of information diffusion in social media? In *AAAI ICWSM*, pages 34–41, 2010.

[DNT12] Thang N Dinh, Dung T Nguyen, and My T Thai. Cheap, easy, and massively effective viral marketing in social networks: truth or fiction? In *ACM HT*, pages 165–174, 2012.

[DPV+07] Jonathan Dushoff, Joshua B Plotkin, Cecile Viboud, Lone Simonsen, Mark Miller, Mark Loeb, and David JD Earn. Vaccinating to protect a vulnerable subpopulation. *PLoS Med*, 4(5):e174, 2007.

[DR01] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *ACM SIGKDD*, pages 57–66, 2001.

[DSGZ13] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. *CoRR*, abs/1311.3669, 2013.

[DYM+14] Rong Du, Zhiwen Yu, Tao Mei, Zhitao Wang, Zhu Wang, and Bin Guo. Predicting activity attendance in event-based social networks: Content, context and social influence. In *ACM UbiComp*, pages 425–434, 2014.

[EIL+12] Dóra Erdös, Vatche Ishakian, Andrei Lapets, Evimaria Terzi, and Azer Bestavros. The filter-placement problem and its application to minimizing information multiplicity. *Proc. VLDB Endow.*, 5:418–429, 2012.

[FCBM14] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, and Shuai Ma. In search of influential event organizers in online social networks. In *ACM SIGMOD*, pages 63–74, 2014.

[FCF+06] Neil M Ferguson, Derek AT Cummings, Christophe Fraser, James C Cajka, Philip C Cooley, and Donald S Burke. Strategies for mitigating an influenza pandemic. *Nature*, 442(7101):448–452, 2006.

[Fen11] Philip E Brown Junlan Feng. Measuring user influence on twitter using modified k-shell decomposition. In *AAAI ICWSM*, 2011.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[GAC+10] Wojciech Galuba, Karl Aberer, Dipanjan Chakraborty, Zoran Despotovic, and Wolfgang Kellerer. Outtweeting the twitterers-predicting information cascades in microblogs. In *USENIX WOSN*, 2010.

[GH12] Adrien Guille and Hakim Hacid. A predictive model for the temporal dynamics of information diffusion in online social networks. In *WWW*, pages 1145–1152, 2012.

[GLL11] Amit Goyal, Wei Lu, and Laks V.S. Lakshmanan. Celf++: Optimizing the greedy algorithm for influence maximization in social networks. In *WWW*, pages 47–48, 2011.

[GLM01] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[Gra78] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, pages 1420–1443, 1978.

[Gra83] Mark Granovetter. The strength of weak ties: A network theory revisited. *Sociological theory*, 1(1):201–233, 1983.

[GRBS11] M. Gomez Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, pages 561–568, 2011.

[GRLK10] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *ACM SIGKDD*, pages 1019–1028, 2010.

[GRLS13] Manuel Gomez Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Structure and dynamics of information pathways in online media. In *ACM WSDM*, pages 23–32, 2013.

[HC13] Mostafa Haghir Chehreghani. An efficient algorithm for approximate betweenness centrality computation. In *ACM CIKM*, pages 1489–1492, 2013.

[Het00] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42:599–653, 2000.

[HSCJ11] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model: Technical report. *CoRR*, abs/1110.4723, 2011.

[Kat73] Elihu Katz. The two-step flow of communication: An up-to-date report on an hypothesis. *Enis and Cox(eds.)*, pages 175–193, 1973.

[KGH+10] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 6(11):888–893, 2010.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM KDD*, pages 137–146, 2003.

[KS06] Masahiro Kimura and Kazumi Saito. Approximate solutions for the influence maximization problem in a social network. In *Springer KES*, pages 937–944, 2006.

[KSHdM02] Anton J. Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. on Optimization*, 12(2):479–502, 2002.

[KSM08] Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Minimizing the spread of contamination by blocking links in a network. In *AAAI*, pages 1175–1180, 2008.

[KSM09] Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Blocking links to minimize contamination spread in a social network. *ACM TKDE*, 3:9:1–9:23, 2009.

[LBK09] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *ACM KDD*, pages 497–506, 2009.

[LCF+14] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-lee Tan, and Wen-syan Li. Efficient location-aware influence maximization. In *ACM SIGMOD*, pages 87–98, 2014.

[LCXZ12] B. Liu, G. Cong, D. Xu, and Y. Zeng. Time constrained influence maximization in social networks. In *IEEE ICDM*, pages 439–448, 2012.

[LKG+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *ACM KDD*, pages 420–429, 2007.

[LLS11] Cheng-Te Li, Shou-De Lin, and Man-Kwan Shan. Finding influential mediators in social networks. In *WWW*, pages 75–76, 2011.

[LMF+07] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S Glance, and Matthew Hurst. Patterns of cascading behavior in large blog graphs. In *SIAM SDM*, pages 551–556, 2007.

[LOTW13] Zhunchen Luo, Miles Osborne, Jintao Tang, and Ting Wang. Who will retweet me?: Finding retweeters in twitter. In *ACM SIGIR*, pages 869–872, 2013.

[LT13] Tiancheng Lou and Jie Tang. Mining structural hole spanners through information diffusion in social networks. In *WWW*, pages 825–836, 2013.

[LXC+14] Qi Liu, Biao Xiang, Enhong Chen, Hui Xiong, Fangshuang Tang, and Jeffrey Xu Yu. Influence maximization over large-scale social networks: A bounded linear approach. In *ACM CIKM*, 2014.

[LZT15] Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. Real-time targeted influence maximization for online advertisements. *PVLDB*, 8(10):1070–1081, 2015.

[MKC+04] Nilly Madar, Tomer Kalisky, Reuven Cohen, Daniel ben Avraham, and Shlomo Havlin. Immunization and epidemic dynamics in complex networks. *European Physical Journal B*, 38(2):269–276, 2004.

[MSM15] Azadeh Mohammadi, Mohamad Saraee, and Abdolreza Mirzaei. Time-sensitive influence maximization in social networks. *Journal of Information Science*, 41:765–778, 2015.

[MZL12] Seth A. Myers, Chenguang Zhu, and Jure Leskovec. Information diffusion and external influence in networks. In *ACM KDD*, pages 33–41, 2012.

[New03] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[New04] Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69:066133, 2004.

[NPW01] Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. In *Springer COCOON*, pages 278–287, 2001.

[NTD] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *ACM SIGMOD*, pages 695–710.

[OAS10] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, 2010.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *WWW*, pages 161–172, 1998.

[Pre02] Luca Pretto. A theoretical analysis of google's pagerank. In *Springer SPIRE*, pages 131–144, 2002.

[Rog04] Everett M Rogers. A prospective and retrospective look at the diffusion model. *Journal of Health Communication*, 9(S1):13–19, 2004.

[Sei83] Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[SGaMZ13] Arlei Silva, Sara Guimarães, Wagner Meira, Jr., and Mohammed Zaki. Profilerank: Finding relevant content and influential users based on information diffusion. In *ACM SNAKDD*, pages 2:1–2:9, 2013.

[SHL15a] Chonggang Song, Wynne Hsu, and Mong Li Lee. Mining brokers in dynamic social networks. In *ACM CIKM*, pages 523–532, 2015.

[SHL15b] Chonggang Song, Wynne Hsu, and Mong Li Lee. Node immunization over infectious period. In *ACM CIKM*, pages 831–840, 2015.

[SHL16] Chonggang Song, Wynne Hsu, and Mong Li Lee. Targeted influence maximization in social networks. In *ACM CIKM*, pages 1683–1692, 2016.

[SHL17] Chonggang Song, Wynne Hsu, and Mong Li Lee. Temporal influence blocking: Minimizing the effect of misinformation in social networks. In *IEEE ICDE*, 2017.

[SKOM09] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Learning continuous-time information diffusion model for social behavioral data analysis. In *Springer ACML*, pages 322–337. 2009.

[SMLGM11] Eldar Sadikov, Montserrat Medina, Jure Leskovec, and Hector Garcia-Molina. Correcting for missing data in information cascades. In *ACM WSDM*, pages 55–64, 2011.

[SNK08] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *Springer KES*, pages 67–75, 2008.

[SOY+11] Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. Learning diffusion probability based on node attributes in social networks. In *Springer ISMIS*, pages 153–162. 2011.

[SS12] Katherine Stovel and Lynette Shaw. Brokerage. *Annual Review of Sociology*, 38(1):139–158, 2012.

[Tar72] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TBM10] Rudra M. Tripathy, Amitabha Bagchi, and Sameep Mehta. A study of rumor control strategies on social networks. In *ACM CIKM*, pages 1817–1820, 2010.

[TNT12] Jason Tsai, Thanh H. Nguyen, and Milind Tambe. Security games for controlling contagion. In *AAAI*, pages 1464–1470, 2012.

[TPER+12] Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *ACM CIKM*, pages 245–254, 2012.

[TPT+10] Hanghang Tong, B Aditya Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. On the vulnerability of large graphs. In *IEEE ICDM*, pages 1091–1096, 2010.

[TSX15] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *ACM SIGMOD*, pages 1539–1554, 2015.

[TXS14] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *ACM SIGMOD*, pages 75–86, 2014.

[Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

[WCF+16] Biao Wang, Ge Chen, Luoyi Fu, Li Song, Xinbing Wang, and Xue Liu. Drimux: Dynamic rumor influence minimization with user experience in social networks. In *AAAI*, pages 791–797, 2016.

[WCSX10] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *ACM KDD*, pages 1039–1048, 2010.

[WCWF03] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *IEEE SRDS*, pages 25–34, 2003.

[WLG+13] Shengxian Wan, Yanyan Lan, Jiafeng Guo, Chaosheng Fan, and Xueqi Cheng. Informational friend recommendation in social media. In *ACM SIGIR*, pages 1045–1048, 2013.

[WWB+13] Beidou Wang, Can Wang, Jiajun Bu, Chun Chen, Wei Vivian Zhang, Deng Cai, and Xiaofei He. Whom to mention: Expand the diffusion of tweets by @ recommendation on micro-blogging systems. In *WWW*, pages 1331–1340, 2013.

[WWX12] Feng Wang, Haiyan Wang, and Kuai Xu. Diffusive logistic model towards predicting information diffusion in online social networks. In *IEEE ICDCSW*, pages 133–139, 2012.

[YDG+15] Zhiwen Yu, Rong Du, Bin Guo, Huang Xu, Tao Gu, Zhu Wang, and Daqing Zhang. Who should i invite for my party?: Combining user preference and influence maximization for social events. In *ACM UbiComp*, pages 879–883, 2015.

[YMPH16] Yu Yang, Xiangbo Mao, Jian Pei, and Xiaofei He. Continuous influence maximization: What discounts should we offer to social network users? In *ACM SIGMOD*, pages 727–741, 2016.

[ZCL+15] Tao Zhou, Jiuxin Cao, Bo Liu, Shuai Xu, Ziqing Zhu, and Junzhou Luo. Location-based influence maximization in social networks. In *ACM CIKM*, pages 1211–1220, 2015.

[ZP14a] Yao Zhang and B. Aditya Prakash. Dava: Distributing vaccines over networks under prior information. In *SIAM SDM*, pages 556–564, 2014.

[ZP14b] Yao Zhang and B. Aditya Prakash. Scalable vaccine distribution in large graphs given uncertain data. In *ACM CIKM*, pages 1719–1728, 2014.

[ZP15] Yao Zhang and B. Aditya Prakash. Data-aware vaccine allocation over large networks. *ACM TKDD*, pages 20:1–20:32, 2015.

[ZYF+14] Shengfu Zhou, Kun Yue, Qiyu Fang, Yunlei Zhu, and Weiyi Liu. An efficient algorithm for influence maximization under linear threshold model. In *IEEE CCDC*, pages 5352–5357, 2014.