Leveraging the Power of Social Propagations: Algorithm Designs for Social Marketing

by

Yu Yang

M.Sc., University of Science and Technology of China, 2013
B.Sc., Hefei University of Technology, 2010

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Science

© Yu Yang 2019
SIMON FRASER UNIVERSITY
Spring 2019

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Approval

Name:	**Yu Yang**
Degree:	**Doctor of Philosophy (Computing Science)**
Title:	**Leveraging the Power of Social Propagations: Algorithm Designs for Social Marketing**
Examining Committee:	**Chair:** Jiannan Wang Assistant Professor
	Jian Pei Senior Supervisor Professor
	Martin Ester Supervisor Professor
	Andrei Bulatov Internal Examiner Professor
	Xiaofang Zhou External Examiner Professor School of Information Technology and Electrical Engineering University of Queensland
Date Defended:	**Feb 6, 2019**

Abstract

Social propagation is a fundamental and prevalent process taking place in social networks, where due to the peer influence of social network users, behaviors of a few influential users can spread widely in a network. Leveraging the social propagation effect lies at the core of mining the marketing values of social networks, and has been extensively studied in the past decade. Most existing studies aim at identifying influential users in a social network, the first step of making good use of propagation in applications like viral marketing and computational advertising. However, in many deeper marketing applications, such as personalized pricing in promotional campaign planning and real-time recommendation of influential bloggers, effectively exploiting social propagations faces many unsettled and challenging algorithmic problems.

In this thesis, we investigate some crucial algorithmic problems in leveraging the propagation effect of social networks in social marketing. In particular, we first discuss how to efficiently monitor top influential users in a rapidly evolving social network. Then we investigate how to spend a budget wisely to motivate influential users to trigger large-scale propagations for marketing purposes. We also study how to schedule an effective propagation to maximize the interaction activities of users influenced, which aims at the marketing effect after the propagation is finished. Our work provides powerful algorithmic tools to solve these problems effectively, which at the same time are efficient and can deal with large networks containing millions or even tens of millions of vertices in a single machine. We conclude this thesis by discussing some future directions in mining social propagations.

Keywords: social propagations; social marketing; dynamic networks; influence spread; budget allocation; influence maximization; activity maximization

To my family.

"Either mathematics is too big for the human mind or the human mind is more than a machine."
— Kurt Friedrich Godël

Acknowledgements

I wish to express my deepest gratitude to my senior supervisor Prof. Jian Pei. This thesis would not have been possible without his supervision and continuous encouragement. Throughout the years of my Ph.D. study, I have learned so much from him, not only research skills but also the wisdom of life.

My gratitude goes to my supervisor Prof. Martin Ester for his valuable feedback for improving the quality of the thesis. I am also grateful to Prof. Andrei Bulatov and Prof. Xiaofang Zhou (University of Queensland) for examining the thesis and providing very insightful comments.

I am sincerely thankful to Prof. Enhong Chen in the University of Science and Technology of China, who guided my virgin steps on the amazing road of data mining research when I was a Master student.

Many thanks to the faculty and support staff in the School of Computing Science for their help over the years. I thank my friends at Simon Fraser University for always being helpful and supportive. A particular acknowledgement goes to Guanting Tang, Xiao Meng, Juhua Hu, Chuancong Gao, Xiangbo Mao, Kui Yu, Yu Tao, Xiaoning Xu, Lumin Zhang, Beier Lu, Li Xiong, Lin Liu, Jiaxing Liang, Zicun Cong, Xiaojian Wang, Dong-wan Choi, Zijin Zhao, Mingtao Lei, Zhefeng Wang, Lingyang Chu, Xia Hu, Yajie (Jill) Zhou and Yanyan Zhang.

Last, but most importantly, I am greatly indebted to my family and especially my parents, my wife and my son. My Ph.D. study has been a rather long journey since I decided to pursue an academic career. Without the unconditional love and support from my family, I would have not been able to follow my passion in research in these years. This thesis is devoted to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Due to the booming of online social network services, our world today is unprecedentedly connected. Social networks like Facebook and Twitter even connect billions of people all around the world together. In social networks, social propagation is a fundamental and prevalent process taking place all the time. Social propagation has been extensively studied in social science research [61, 60, 31, 6], where the conclusion is that people's opinions and behaviors are often affected by their families, friends, and colleagues. As a result, a behavior (e.g., adopting a product, sharing an article) or an opinion (e.g., a political point of view, an attitude to a new technology) may spread in a social network like a virus. For example, social-behavioral research [74] showed that social influence plays a prominent role in many self-organized phenomena such as herding in cultural markets, the spread of ideas and innovations, and the amplification of fears during epidemics.

The propagation effect makes social networks full of business values, especially in marketing applications. One classic example is how Hotmail became hot. When Hotmail first started, the email provider added a link of signing up Hotmail for free at the end of every email sent out. Such a simple strategy helped Hotmail gain 8 million users in only 1.5 years in the 1990s, which was astonishing at that time. How to effectively and efficiently leverage the power social propagations lies at the core of mining marketing values of social networks.

1.1 Motivation

Motivated by the great business value of social propagations, computational social propagation/influence analysis became an important research area in data mining in the past decade. The aim is to use algorithmic techniques to take advantage of social propagations in business applications like marketing. In computational social propagation analysis research, Kempe *et al.* [54] first proposed to model social influence propagations as stochastic processes taking place in social networks. Following this seminal work [54], numerous studies had been made in exploring social propagations in marketing applications.

Most existing studies [35, 21, 19, 10, 103, 71, 36, 109, 83, 77, 70] aim at identifying a set of influential users to maximize the total influence spread and assume that the underlying network where social propagations take place is static. However, considering the highly dynamic nature of real social networks and the needs of many deeper marketing applications, effectively exploiting social propagations faces many unsettled and challenging algorithmic problems. Below, we list some important application scenarios where existing social propagation mining techniques cannot provide satisfactory solutions.

Scenario I: Dealing with Fast-Evolving Social Networks. To effectively utilize social propagations, we need to pay attention to influential users who have the ability to trigger large scale propagations. A big challenge is that computing users' influence is usually a #P-hard problem [35, 21, 19]. Although there are some techniques that can approximate users' influence in polynomial time [10, 71, 55], all these techniques work on static networks. It is well known that real social networks evolve rapidly [2], thus, re-running algorithms for approximating users' influence [10, 71, 55] every time when the network changes is not practical. How to efficiently track influential users with good quality guarantees in a highly dynamic social network poses a new challenge.

Scenario II: Budget Allocation for Influence Maximization. When we want to promote a product through a social network, we need to spend money to motivate influential users to trigger a large scale propagation of adopting our product. Most existing studies [35, 10, 103, 83, 70] consider how to select some users to offer free samples of the product, whom are called the initial adopters, to maximize the influence spread. However, free samples may not be the most effective way to entice potential customers. Sometimes by offering discounts to users we can have more initial adopters using the same budget. Suppose we know the purchase probability curves of social network users, given a budget on the total discounts that we can offer, how do we allocate personalized discounts to social network users in order to maximize the influence spread?

Scenario III: Optimizing Post-Propagation Marketing Effect. The result of a social propagation is a subgraph induced by the set of users who are influenced. Almost all existing studies [35, 21, 19, 10, 103, 71, 36, 109, 83, 77, 70] use influence spread, which is the expected number of users in the propagation induced subgraph, as the objective for optimization. However, the famous Metcalfe's Law [95] suggests that edges matter in deciding the value of a network. In marketing applications, connections/edges of the propagation induced subgraph capture the post-propagation effect. For example, if users influenced in a propagation are densely connected to each other, they may interact a lot on the product/topic of the propagation to further enhance the marketing effect. In a social network, what users should we choose as the initial adopters of our propagation if we hope the post-propagation effect is maximized?

1.2 Overview of Contributions

In this thesis, to tackle the problems mentioned above that cannot be solved by existing studies in mining social propagations, we conduct extensive research to solve three critical problems in social propagation aware marketing. In particular, we make the following contributions.

- **Tracking Influential Vertices in Fast-Evolving Social Networks.** We design an incremental algorithm to maintain a number of poll samples by which we can approximate users' influence. When the network is only slightly updated, instead of re-generating all poll samples from scratch, our incremental algorithms only retrieve and update a few poll samples. Our algorithm also dynamically adjusts the number of poll samples maintained, such that we always can extract top influential users with provable quality guarantees. Moreover, we devise an efficient data structure to efficiently maintain user ranking based on users' approximate influence.
- **Budget Allocation for Maximizing Adoption of Promoted Product.** Given the purchase probability curves of the users in a social network, we formalize the continuous influence maximization (CIM) problem which aims at allocating personalized discounts to users for maximizing influence spread. We devise an algorithmic framework for the CIM problem, where any propagation models can be plugged into. We also investigate the connections between CIM and traditional influence maximization. To implement our algorithmic framework under the family of triggering models, we propose principled algorithms that can effectively approximate the influence spread function to avoid the "overfitting" issue caused by the #P-hardness of computing influence spread.
- **Maximizing Activity by Information Propagation.** We formulate the Activity Maximization (AM) problem whose goal is to maximize the expected total edge weights of the propagation induced subgraph. We first show that AM is a hard problem by investigating the intractability, the inapproximability and the non-modularity of the AM problem. To solve the AM problem, we propose a Sandwich Approximation scheme which enjoys a data-dependent approximation ratio.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we briefly introduce influence propagation models, the most important background knowledge of our research. We also review state-of-the-art studies related to our research in Chapter 2. We present our research of tracking top influential vertices in dynamic networks in Chapter 3. In Chapter 4, we study the problem of how to wisely spend a predefined budget to motivate influential users to trigger large-scale propagations for our marketing purposes. In Chapter 5, we consider

a new objective, the expected activities among active users, in designing algorithms for scheduling effective social propagations. Finally, we conclude this thesis and discuss some important directions in mining social propagations for business applications like marketing.

Chapter 2

Influence Propagation Models and Related Work

In this chapter, we briefly introduce popularly adopted influence propagation models in literature, which is the most important background knowledge of our research. We also review state-of-the-art studies in social propagation mining that are closely related to our work presented in this thesis.

2.1 Influence Propagation Models

To study and utilize social propagations from a computational perspective, we first need to model how propagations happen in social networks. In literature, an influence propagation is often modeled as a stochastic process taking place in a network [46, 99, 16, 9]. In this section, we introduce two most popular stochastic propagation models, the Linear Threshold (LT) model and the Independent Cascade (IC) model. Both the IC model and the LT model are based on well-known studies of the diffusion phenomenon in marketing research [38, 39], epidemiology [4], and behavior research in social science [45].

2.1.1 Linear Threshold Model

Consider a directed social/propagation network $G = \langle V, E, w \rangle$ where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and each edge $(u, v) \in E$ is associated with an influence weight $w_{uv} \in [0, +\infty)$. Each vertex $v \in V$ also carries a weight w_v, which is called the *self-weight* of v. Denote by $W_v = w_v + \sum_{u \in N^{in}(v)} w_{uv}$ the total weight of v, where $N^{in}(v)$ is the set of v's in-neighbors.

We define the *influence probability* p_{uv} of an edge (u, v) as $\frac{w_{uv}}{W_v}$. Clearly, for $v \in V$, $\sum_{u \in N^{in}(v)} p_{uv} \leq 1$.

In the Linear Threshold (LT) model [54], given a seed set $S \subseteq V$, the influence propagates in G as follows. First, every vertex u randomly selects a threshold $\lambda_u \in [0, 1]$, which reflects our lack of knowledge about users' true thresholds. Then, influence propagates iteratively.

Denote by S_i the set of vertices that are active in step i ($i = 0, 1, \ldots$) and $S_0 = S$. In each step $i \geq 1$, an inactive vertex v becomes active if

$$\sum_{u \in N^{in}(v) \cap S_{i-1}} p_{uv} \geq \lambda_v$$

The propagation stops at step t if $S_t = S_{t-1}$. Let $I(S)$ be the expected number of vertices that are finally active when the seed set is S. We call $I(S)$ the *influence spread* of S. Let I_u be the influence spread of a single vertex u.

Kempe *et al.* [54] proved that the LT model is equivalent to a "live-edge" process where each vertex v picks at most one incoming edge (u, v) with probability p_{uv}. Consequently, v does not pick any incoming edges with probability $1 - \sum_{u \in N^{in}(v)} p_{uv} = \frac{w_v}{W_v}$. All edges picked are "live" and the others are "dead". Then, the expected number of vertices reachable from $S \subseteq V$ through live edges is $I(S)$, the influence spread of S.

It is worth noting that our description of the LT model here is slightly different from the original [54]: we use a function of edge weights and self-weights of vertices to represent influence probabilities. Representing influence probabilities in this way is widely adopted in the existing literature [21, 44, 104, 103, 42].

2.1.2 Independent Cascade Model

A social/propagation network in the Independent Cascade (IC) model is also a weighted graph $G = \langle V, E, w \rangle$. Let w_{uv} represent the propagation probability of the edge (u, v), which is the probability that v is activated by u through the edge in the next step after u is activated. Clearly for the IC model, all $w_{uv} \in [0, 1]$.

In the IC model [54], given a seed set $S \subseteq V$, the influence propagates in G iteratively as follows. Denote by S_i the set of vertices that are active in step i ($i = 0, 1, \ldots$) and $S_0 = S$. At step $i + 1$, each vertex u in S_i has a single chance to activate each inactive neighbor v with an independent probability w_{uv}. The propagation stops at step t if $S_t = \emptyset$. Similar to the LT model, the influence spread $I(S)$ denotes the expected number of vertices that are finally active when the seed set is S.

The "live-edge" process [54] of the IC model is to keep each edge (u, v) with a probability w_{uv} independently. All kept edges are "live" and the others are "dead". Then, the expected number of vertices reachable from S via live edges is the influence spread $I(S)$.

2.1.3 Properties of Influence Spread

Some of our research considers the LT model and the IC model as the influence propagation models, while some of our research can be applied to any influence propagation models whose influence spread functions satisfy certain intuitive properties. Thus, we also introduce some

intuitive properties of influence spread that are widely accepted in literature. Denoted by $I(S)$ the influence spread of the seed set S.

First, influence spread often is hard to compute, for example, computing the influence spread $I(S)$ is #P-hard for the LT model [21], the IC model [19], and many other triggering models including the Continuous-Time Diffusion model [35]. We often use Monte Carlo simulations [54] or reverse Monte Carlo simulations (also called the polling method) [10] to approximate $I(S)$.

Second, the influence spread function $I(\cdot)$ is monotone [54], which means $I(S) \leq I(T)$ if $S \subseteq T$. Second, $I(\cdot)$ is submodular [54], also known as "diminishing return", which indicates that $I(S \cup \{u\}) - I(S) \geq I(T \cup \{u\}) - I(T)$ for any $S \subseteq T$. These two properties are used to design the greedy algorithm for influence maximization and prove its effectiveness [54]. These two properties are very easy to be satisfied [73] and hold for many influence propagation models including the LT model and the IC model.

Note that since our work is majorly for solving algorithmic challenges in social propagation aware marketing, we assume that we have the full information of the propagation network. There is a rich body of research investigating how to learn the propagation network from data [29, 40, 42, 79, 93, 59, 91, 76].

2.1.4 Approximating Influence Spread by the Polling Sketch

Since computing influence spread is #P-hard under many influence propagation models [73], how to efficiently approximate influence spread becomes an essential building block in influence analysis tasks [16, 9]. Recently, a polling-based method [10, 104, 103] was proposed for approximating influence spread of triggering models [54] like the LT model and the IC model. Here we briefly review the polling method for computing influence spread.

Given a social network $G = \langle V, E, w \rangle$, a poll is conducted as follows: we pick a vertex $v \in V$ in random and then try to find out which vertices are likely to influence v. We run a Monte Carlo simulation of the equivalent "live-edge" process. The vertices that can reach v via live edges are considered as the potential influencers of v. The set of influencers found by each poll is called a *poll sample* (or *RR (Reversely Reachable) set* in [104, 103, 82]).

Let h_1, h_2, ..., h_M be a sequence of poll samples generated by M independent polls. The M poll samples form a polling sketch $\mathcal{R} = \{h_1, h_2, ..., h_M\}$. Denote by $\mathcal{D}(S)$ the degree of a set of vertices S in the polling sketch, which is the number of poll samples containing at least one vertex in S. By the linearity of expectation, it has been shown that $n\frac{\mathcal{D}(S)}{M}$ is an unbiased estimator of $I(S)$ [10, 103]. Thus, $n\frac{\mathcal{D}(S)}{M}$ is often used as the approximate influence spread of the seed set S.

To bound the error of the approximate influence spread $n\frac{\mathcal{D}(S)}{M}$, Tang *et al.* [103] proved that the corresponding sequence x_1, x_2, ..., x_M is a martingale [25], where $x_i = 1$ if $S \cap h_i \neq \emptyset$ and $x_i = 0$ otherwise. We have $E[\sum_{i=1}^{M} x_i] = E[\mathcal{D}(S)] = \frac{MI(S)}{n}$. The following results [103]

show how $E[\sum_{i=1}^{M} x_i]$ is concentrated around $\frac{MI(S)}{n}$, even when variables $x_1, x_2, ..., x_M$ may be weakly dependent due to the stopping condition on M.

Corollary 0.1 ([103]). *For any $\xi > 0$,*

$$\Pr\Big[\sum_{i=1}^{M} x_i - Mp \geq \xi Mp\Big] \leq \exp\Big(-\frac{\xi^2}{2+\frac{2}{3}\xi}Mp\Big)$$
$$\Pr\Big[\sum_{i=1}^{M} x_i - Mp \leq -\xi Mp\Big] \leq \exp\Big(-\frac{\xi^2}{2}Mp\Big)$$

where $p = \frac{I(S)}{n}$.

Besides Corollary 0.1, another powerful probability theory tool to analyze the quality of a polling sketch is the Stopping Rule Theorem [28], which is based on the Stopping Rule algorithm proposed by Dagum *et al.* [28]. Suppose we try to estimate the expectation of a random variable Z distributed in the interval $[0, 1]$ with $\mu_Z = E[Z] > 0$. Let $\{Z_1, Z_2, Z_3, ...\}$ be an infinite series of i.i.d. (independently and identically distributed) random variables according to Z. Define $\Upsilon^{\epsilon,\delta} = \frac{4(e-2)\ln\frac{2}{\delta}}{\epsilon^2}$ and $\Upsilon_1^{\epsilon,\delta} = 1 + (1+\epsilon)\Upsilon^{\epsilon,\delta}$. The Stopping Rule algorithm (Algorithm 1) shows how to obtain an (ϵ, δ) estimation[1] of μ_Z.

Algorithm 1 Stopping Rule Algorithm

```
Υ_1^{ε,δ} = 1 + (1 + ε)Υ^{ε,δ}
M ← 0, A ← 0
while A < Υ_1 do
    M ← M + 1; A ← A + Z_M
end while
return μ̂_Z ← Υ_1/M
```

The following Corollary states the effectiveness of Algorithm 1.

Corollary 0.2 (Stopping Rule Theorem [28]). *Let Z be a random variable distributed in the interval $[0, 1]$ with $\mu_Z = E[Z] > 0$. Let $\hat{\mu}_Z$ be the estimate produced and let M be the number of experiments that the Stopping Rule algorithm runs with respect to Z on the input ϵ and δ. Then, (1) $\Pr\{\hat{\mu}_Z \geq (1-\epsilon)\mu_Z\} \geq 1 - \frac{\delta}{2}$ and $\Pr\{\hat{\mu}_Z \leq (1+\epsilon)\mu_Z\} \geq 1 - \frac{\delta}{2}$, (2) $E[M] \leq \Upsilon_1^{\epsilon,\delta}/\mu_Z$.*

Obviously, the random variables $x_1, x_2, ..., x_M$ defined above are i.i.d. and distributed in $[0, 1]$. In the following of this thesis, both Corollary 0.1 and Corollary 0.2 are frequently applied to analyze the estimation errors of a given polling sketch $\mathcal{R} = \{h_1, h_2, ..., h_M\}$ and guide the algorithm designs in our work.

[1] An estimate $\hat{\mu}$ is an (ϵ, δ) estimation of μ if $\Pr\{|\hat{\mu} - \mu| \geq \epsilon\mu\} \leq \delta$.

2.2 Related Work

In this section, we review state-of-the-art studies in social propagation mining that are related to our research. We categorize related work into two groups, namely identifying influential users, which is related to our work in Chapter 3, and optimization for viral marketing, which is related to our work in Chapter 4 and Chapter 5.

2.2.1 Related Work in Identifying Influential Users

Domingos *et al.* [33] proposed to take advantage of peer influence between users in social networks for marketing. Kempe *et al.* [54] formulated the problem using two discrete influence models, namely Independent Cascade model and Linear Threshold model. Since then, influence computation, especially influence maximization, has drawn much attention from both academia and industry [16, 35, 10, 103, 21, 44, 71, 92, 26, 71]. Chen *et al.* proved that computing influence spread is #P-hard for both the Linear Threshold model [21] and the Independent Cascade model [19]. Recently, a polling-based method [10, 103, 104] was proposed for influence maximization under general triggering models. The key idea is to use some poll samples [103, 104] to approximate the real influence spread of vertices. The error of approximation can be bounded with a high probability if the number of poll samples is large enough.

Extracting influential vertices in social networks is also an important problem in social network analysis and has been extensively investigated [41, 1, 111, 14]. In addition to the marketing value, influential individuals are also useful in recommender systems in online web service [1, 111]. Due to the computational hardness of influence spread [21, 19], most methods did not use influence models to measure a user's influence, but adopted measures like PageRank which can be efficiently computed.

In a few applications, the underlying networks are evolving all the time [63, 64, 2]. Rather than re-computing from scratch, incremental algorithms are more desirable in graph analysis tasks on dynamic networks. Maintaining PageRank values of vertices on an evolving graph was studied in [5, 85]. Hayashi *et al.* [47] proposed to utilize a sketch of all shortest paths to dynamically maintain the edge betweenness value. The dynamics considered by the above work is a stream of edge insertions/deletions, which is not suitable for influence computation. The dynamics of influence network is more complicated, because besides edge insertions/deletions, influence probabilities of edges may also evolve over time [62].

Aggarwal *et al.* [3] explored how to find a set of vertices that has the highest influence within a time window $[t_0, t_0 + h]$. They modeled influence propagation as a non-linear system which is very different from triggering models like the Linear Threshold model or the Independent Cascade model. The algorithm in [3] is heuristic and the results produced do not come with any provable quality guarantee.

Chen *et al.* [22] investigated incrementally updating the seed set for influence maximization under the Independent Cascade model. They proposed an algorithm which utilizes the seed set mined from the former network snapshot to efficiently find the seed set of the current snapshot. An Upper Bound Interchange heuristic is applied in the algorithm. However, the algorithm in [22] is costly in processing updates, since updating the Upper Bound vector for filtering non-influential vertices takes $O(m)$ time where m is the number of edges. Moreover, the SP1M heuristic [57], which does not have any approximation quality guarantee, was adopted in [22] for estimating influence spread. Thus, the set of influential vertices, even when the size of the seed set is set to 1, does not have any provable quality guarantee.

Independently and simultaneously[2] Ohsaka *et al.* [84] studied a related problem, maintaining a number of poll samples over a stream of network updates under the IC model such that $(1 - 1/e - \epsilon)$-approximation influence maximization queries can be achieved with probability at least $1 - \frac{1}{n}$. Our work in Chapter 3 is different from [84] in the following aspects. First, the problems are different. The problem tackled in [84] is influence maximization, while our problem is tracking influential individuals. Second, [84] only studied the IC model while in our work we addressed both the IC and the LT models. Moreover, our algorithm is theoretically sound and was strictly implemented to fulfill the theoretical guarantee in experiments, while it is not the case in [84]. To enable theoretical guarantees for the algorithm in [84], one has to collect enough poll samples until the cost of all poll samples (i.e., the number of edges traversed when generating those poll samples) is $4(1 + \epsilon)(1 + \frac{1}{k})k\frac{m \log n}{\epsilon^2}$ [10][3]. This number is very large under reasonable settings of k and ϵ, for example, $k = 50$ and $\epsilon = 0.4$. Thus, in the experiments of [84], the demanded cost is empirically set to $32(m + n) \log n$ for any k and ϵ, which means the experiments of [84] are not strictly implemented to fulfill the theoretical analysis.

2.2.2 Related Work in Optimization for Viral Marketing

Domingos *et al.* [33] proposed to take advantage of peer influence between users in social networks for marketing. The essential idea is that, by targeting on only a small number of users (called seed users), it is possible to trigger a large cascade of users purchasing a specific product through a social network. Consequently, the technical challenge is to find a small set of users who can trigger the largest cascade in the network. Kempe *et al.* [54] formulated the problem as a discrete optimization problem, which is well known as the influence maximization problem. Since then, influence maximization has drawn much attention from both academia and industry [19, 20, 21, 44, 35, 104, 103, 82].

[2]Early versions of our work in Chapter 3 can be found at `https://arxiv.org/abs/1602.04490`

[3]In [84] the number was incorrectly set as $\Theta(\frac{(m+n) \log n}{\epsilon^3})$, which is based on an early version of [10]. The latest version of [10] corrects this number to be $4(1 + \epsilon)(1 + \frac{1}{k})k\frac{m \log n}{\epsilon^2}$.

Most influence maximization algorithms are designed for triggering models [54]. Among these algorithms, a polling-based method [10] has the lowest worst-case time complexity, $O((k+l)(n+m)\log^2 n/\epsilon^3)$. Tang *et al.* [104, 103] further improved the method to make it run in $O((k+l)(n+m)\log n/\epsilon^2)$ expected time. The empirical studies showed that their improved algorithms are orders of magnitude faster than the other influence maximization algorithms. Lei *et al.* [62] proposed a method that learns the propagation probabilities while running the viral marketing campaigns. Another line of algorithmic viral marketing research is budgeted influence maximization [109, 83]. Under such problem settings, every user in a social network is associated with a threshold value that indicates the amount of money a company needs to spend to persuade her/him to be an initial adopter. One key problem of this setting is how to obtain users' threshold values. Singer [97] proposed a mechanism that can elicit users' true threshold values if they are rational agents. Chen *et al.* [16] provided a comprehensive survey on influence maximization algorithms.

Farajtabar *et al.* [37] modeled social events using multivariate Hawkes processes, and developed a convex optimization framework for determining the required level of external incentives (the money spent on users) in order for the network to reach a desired activity level. Although the objective function in [37] is flexible since it only requires that the objective is a concave utility function, both the properties explored in [37] and the algorithm proposed are only suitable for multivariate Hawkes processes rather than a general propagation model. Descriptive propagation models, such as the independent cascade model and the linear threshold model [54], the two most widely used models, cannot fit in the framework [37].

Eftekhar *et al.* [36] discovered that sometimes instead of targeting on very few individual users, persuasion attempts on groups of users, for example, displaying advertisements to them, may lead to a wider range of cascades in social networks. The motivation of persuasion on groups is that by spending less money on a targeted individual a company can target at much more users and, as a result, in expectation such a strategy may bring in more initial adopters [106]. Eftekhar *et al.* [36] assumed that the probability that a user is persuaded to be a seed user is given and fixed, once the user is targeted. A more realistic strategy is that we can adjust the resource spent on a specific individual to manipulate the probability the user becomes a seed user, which is the subject studied in Chapter 4.

Demaine *et al.* [30] studied the problem of influencing people with partial derivatives (discounts). The output of the method in [30] is similar to ours, which is an n-dimensional vector $C \in [0,1]^n$, where n is the number of vertices. But the problem setting of [30] is very different from ours. First, [30] is based on the Linear Threshold model, while our work does not assume any specific propagation model. In Chapter 4, all theoretical results are applicable to submodular and monotonic propagation models, and the Linear Threshold model is just one of them. Our implementation in Section 4.6 can be applied to any triggering models whose reverse propagation can be simulated. Also, the Linear Threshold model is

just one of these models. Second, in [30], a discount c offered to any user u is equivalent to reducing u's threshold θ_u by c, that is, making θ_u a random variable in $[0, 1-c]$, no matter what u is. While in reality, how much the discount c affects a user u's threshold should depend on u's purchase probability curve, which is a personalized property of u. Thus, [30] actually does not utilize the purchase probability curves of users.

Kempe *et al.* [55] investigated the general marketing strategies whose problem setting is similar to ours. A major difference in the problem setting is that [55] assumes that all seed probability functions (please refer to Section 4.2 for the definition) have the "diminishing return" property, which means all seed probability functions are concave or near-concave. In Chapter 4, we do not assume this "diminishing return" property. Instead, a seed probability function can be in any arbitrary shape, even convex. Moreover, the method in [55] is based on discretizing the budget, which means a discount offered to a user is discrete (finitely many), while we provide a totally continuous solution.

An important issue that [55] did not discuss is the complexity of computing the expected influence spread of a given budget allocation plan. Kempe *et al.* [55] just assumed an oracle to return this value when needed. Moreover, Kempe *et al.* [55] did not conduct any experiments of the budget allocation problem. In Chapter 4, we proved that unfortunately, computing the expected influence spread of a budget allocation plan is usually #P-hard, which poses a big challenge in implement an efficient budget allocation algorithm. To address this challenge, we devise a polling based algorithm where the idea is to generate a polling sketch to obtain an approximate influence spread function. We also identify the "overfitting" issue caused by the #P-hardness of computing the expected influence spread. Our solution in Chapter 4 provides principled implementations of our algorithms under specific propagation models to avoid the "overfitting" issue.

Although influence maximization has its root in viral marketing, it may still be impractical under many real-life scenarios. To fill this gap, a series of extensions to the influence maximization problem were studied. For example, Goyal *et al.* [43] proposed a data based approach to influence maximization based on a credit distribution model. Instead of maximizing the influence spread under some propagation models with respect to some learned parameters, they tried to find influential vertices from the action log data directly. Chen *et al.* [18] considered the time-delay aspect of influence diffusion and studied the influence maximization with time-critical constraint. Similarly, the spatial factor of influence diffusion is considered and influence maximization on Euclidean space has been studied as well [100, 50, 112]. Tang *et al.* [101] studied the problem of maximizing the influence spread and the diversity of the influenced crowd simultaneously. Bhagat *et al.* [7] argued that product adoption should be distinguished from influence spread in viral marketing, as influence spread is essentially used as "proxy" for product adoption. Wang *et al.* [110] distinguished the information coverage and information propagation, and proposed a new optimization objective that includes the values of the informed vertices. All these exten-

sions were from the perspective of vertices and tried to exploit the values of vertices as separate individuals in different diffusion models and different problem settings. They did not consider activity strengths on edges in their objectives. In contrast, our problem in Chapter 5 captures the interactions among vertices and enables different (often orthogonal) applications of information diffusion.

Chapter 3

Tracking Influential Vertices in Dynamic Networks

In this chapter, we tackle a challenging problem inherent in a series of applications: tracking the influential vertices in dynamic networks. Specifically, we model a dynamic network as a stream of edge weight updates. This general model embraces many practical scenarios as special cases, such as edge and vertex insertions, deletions as well as evolving weighted graphs. Under the popularly adopted Linear Threshold model and Independent Cascade model, we consider two essential versions of the problem: finding the vertices whose influences passing a user specified threshold T and finding the top-k most influential vertices. Our key idea is to adopt the polling sketch to approximate influence spread of each vertex. We developed efficient algorithms to update the poll samples and determine proper sample sizes for the two versions of tracking influential vertices. In addition to the thorough theoretical results, our experimental results on five real network data sets clearly demonstrate the effectiveness and efficiency of our algorithms.

3.1 Introduction

More and more applications are built on dynamic networks and need to track influential vertices. For example, consider cold-start recommendation in a dynamic social network – we want to recommend to a new comer some existing users in a social network. A new user may want to subscribe to the posts from some users in order to obtain hot posts (posts that are widely spread in the social network) at the earliest time. Clearly for such a new user we should recommend her some influential users in the current network. Traditional Influence Maximization cannot find those influential users we want here because it is for marketing in which all seed users have to be synchronized to spread the same content, while in reality online influential individuals often produce and spread their own contents in an asynchronized manner. The influential users we want are those who have high individual influence spread.

More often than not, the underlying network is highly dynamic, where each vertex is a user and an edge captures the interaction from a user to another. User interactions evolve continuously over time. In an active social network, such as Twitter, Facebook, LinkedIn, Tencent WeChat, and Sina Weibo, the evolving dynamics, such as rich user interactions over time, is the most important value. It is critical to capture the most influential users in an online manner. To address the needs, we have to tackle two challenges at the same time, influence computation and dynamics in networks.

Influence computation is very costly, technically #P-hard under most influence models. Most existing studies have to compromise and consider the influence maximization problem only on a static network. Here, influence maximization in a network is to find a set of vertices S such that the combined influence of the vertices in the set is maximized and S satisfies some constraints such as the size of S is within a budget. The incapability of handling dynamics in large evolving networks seriously deprives many opportunities and potentials in applications. Also note that influence maximization is very different from finding influential individuals, for the reason that the best k-vertices set S does not necessarily consist of the k most influential individual vertices because influence spreads of different individual vertices may overlap.

Although influence maximization and finding influential vertices are highly related since they both need to compute influence in one way or another, these two problems serve very different application scenarios and face different technical challenges. For example, influence maximization is a core technique in viral marketing [33]. At the same time, influence maximization is not useful in the cold-start recommendation scenario discussed above, since a user is interested in being connected with individual users of great potential influence and may follow them in interaction.

To the best of our knowledge, our study is the first to tackle the problem of tracking influential individual vertices in dynamic networks. Specifically, we model a dynamic network as a stream of edge weight updates. Our model is general and embraces many practical scenarios as special cases. Under the popularly adopted Linear Threshold model and Independent Cascade model, we consider two essential versions of the problem: (1) finding the vertices whose influences passing a user specified threshold; and (2) finding the top-k most influential vertices. Our key idea is to maintain a number of poll samples so that we can approximate the influence of vertices with provable quality guarantees.

Although recently there is some progress in influence computation in dynamic networks [22, 3, 84], these studies are either heuristic methods or hard to be strictly implemented to fulfill the theoretical guarantees. Thus, computing influence spread in a dynamic network with practical and provable quality guarantees still is a challenging problem.

To tackle the novel and challenging problem of finding influential vertices in dynamic networks, we make several technical contributions. We develop an efficient algorithm that incrementally updates the existing poll samples against network changes. We also design

methods to determine the proper sample sizes for the two versions of the problem so that we can provide solutions with strong quality guarantees and at the same time be efficient in both space and time. In addition to the thorough theoretical results, our experimental results on five real data sets clearly demonstrate the effectiveness and efficiency of our algorithms. The largest data set used contains over 41 million vertices, 1.5 billion edges and 0.3 billion edge updates.

The rest of this chapter is organized as follows. We model the dynamics of a social network and propose a framework of updating poll samples in a dynamic network in Section 3.2. In Section 3.3, we present methods updating existing poll samples against a stream of edge weight updates. In Section 3.4, we tackle the problem of tracking vertices whose influence spreads pass a user-defined threshold. In Section 3.5 and Section 3.6, the problem of finding the top-k influential vertices is settled. We devise an efficient data structure to maintain vertex ranking in Section 3.7. We report the experimental results in Section 3.8. We conclude this chapter in Section 3.9.

3.2 Influence in Dynamic Networks

In this section, we first model influence in dynamic networks. Then by adopting the polling sketch to approximate users' influence spreads, we propose a framework of updating poll samples against a dynamic network. For readers' convenience, Table 4.1 lists the frequently used notations in this chapter.

3.2.1 Dynamic Networks

Real online social networks, such as the Facebook network and the Twitter network, change very fast and all the time. Relationships among users keep changing, and influence strength of relationships also varies over time. Lei *et al.* [62] pointed out that influence probabilities may change due to former inaccurate estimation or evolution of users' relations over time. However, the traditional formulation of dynamic networks only considers the topological updates, that is, edge insertions and edge deletions [5, 85, 47]. Such a formulation is not suitable for realtime accurate analysis of influence.

According to the LT model reviewed in Section 2.1.1, the change of influence probabilities along edges can be reflected by the change of edge weights. For the IC model, since the weight of an edge is the propagation probability, the updates on edge weights are updates on propagation probabilities. Therefore, we model a dynamic network as a stream of weight updates on edges.

A *weight update* on an edge is a 5-tuple $(u, v, +/-, \Delta, t)$, where (u, v) is the edge updated, $+/-$ is a flag indicating whether the weight of (u, v) is increased or decreased, $\Delta > 0$ is the amount of change to the weight and t is the time stamp. The update is applied to the self-weight w_u if $u = v$. Clearly, edge insertions/deletions considered in the existing

Table 3.1: Frequently used notations.

Notation	Description
$G = \langle V, E, w \rangle$	A social network, where each edge $(u, v) \in E$ is associated with an influence weight w_{uv}
w_{uv}	weight of the edge (u, v) (LT model); propagation probability of the edge (u, v) (IC model)
$n = \|V\|$	The number of vertices in G
$m = \|E\|$	The number of edges in G
$N^{in}(u)$	The set of in-neighbors of u
w_u	Self-weight of u
W_u	$W_u = w_u + \sum_{v \in N^{in}(u)} w_{vu}$, the total weight of u
p_{uv}	$p_{uv} = \frac{w_{uv}}{W_u}$, the probability that v is influenced by its neighbor u (LT Model)
I_u	The influence spread of vertex u
$\bar{I}$	The average influence spread of individual vertices
M	The number of random poll samples
$\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$	A polling sketch which is a set of poll samples. $\mathcal{R}$ is divided into two disjoint partitions $\mathcal{R}_1$ and $\mathcal{R}_2$.
$\mathcal{D}_1(u)$	The degree of $u \in V$ in $\mathcal{R}_1$, which is the number of poll samples containing u in $\mathcal{R}_1$. Similarly we can also define $\mathcal{D}_2(u)$ and $\mathcal{D}(u)$.
$\mathcal{D}_1^k$	The $k-$th highest $\mathcal{D}_1(u)$ value for $u \in V$
T	Influence threshold set by users
I^*	Influence spread of the most influential individual vertex
I^k	Influence spread of the $k-$th most influential individual vertex

literature [5, 85, 47, 22] can be easily written as weight increase/decrease updates. Moreover, vertex insertions/deletions can be written as edge insertions/deletions, too.

Example 1. A retweet network is a weighted graph $G = \langle V, E, w \rangle$, where V is a set of users. An edge $(u, v) \in E$ captures that user v retweeted from user u. We can set w_{uv} according to the propagation model adopted as follows.

LT Model: The edge weight w_{uv} is the number of tweets that v retweeted from u. The self-weight w_v is the number of original tweets posted by v. The weights reflect the influence in the social network. By intuition, if v retweeted many tweets from u, v is likely to be influenced by u. In contrast, if most of v's tweets are original, v is not likely to be influenced by others.

IC Model: The edge weight w_{uv} is the probability that v retweets from u, which can be calculated according to v's retweeting record in the past [93, 42].

An essential task in online social influence analysis is to capture how the influence changes over time. For example, one may want to consider only the retweets within the past Δt time. Clearly, the set of edges E may change and the weights w_{uv} and w_v may increase or decrease over time. The dynamics of the retweet network can be depicted by a stream of edge weight updates $\{(u, v, +/-, \Delta, t)\}$.

Algorithm 2 Framework of Updating Poll Samples

```
1: retrieve poll samples affected by the updates of the graph
2: update retrieved poll samples
3: if the current poll samples are insufficient then
4:    add new poll samples
5: else
6:    if the current poll samples are redundant then
7:       delete the redundant poll samples
8:    end if
9: end if
```

3.2.2 Updating Poll Samples Against Edge Updates

Given a dynamic network like the retweet network in Example 1, how can we keep track of influential users dynamically? In order to know the influential vertices, the critical point is to monitor influence of users. To solve this problem, we adopt the polling sketch for approximating influence spread described in Section 2.1.4, and extend it to tackle dynamic networks. The major challenge is how to maintain a number of poll samples over a stream of edge updates, such that all poll samples maintained are randomly generated according to the current snapshot of the network. We propose a framework for updating poll samples that addresses various tasks of tracking influential vertices.

The framework is shown in Algorithm 2, which contains two building blocks, namely updating existing poll samples (line 2) and deciding a proper sample size (line 3 and line 6). In Section 3.3, we discuss how to efficiently update the existing poll samples. How to decide if our current poll samples are insufficient, redundant or in proper amount depends on the specific task of tracking influential vertices. In Sections 3.4 , we discuss how to decide a sample size for tracking vertices with influence greater than a threshold. In Section 3.5 and Section 3.6, we settle the sample size issue for tracking top-k influential vertices.

3.3 Updating Existing Poll Samples

In this section, we propose incremental algorithms for updating existing poll samples over a stream of edge weight updates under both the LT model and the IC model. Denote by $\Pr(h \mid G)$ the probability of generating a poll sample h from a given propagation graph G. The idea of our incremental algorithm is to ensure that after updating poll samples at any time t, the probability that a poll sample h appearing in our polling sketch is always $\Pr(h \mid G^t)$, where G^t is the social network at time t.

3.3.1 Updating under the LT Model

First, we have a key observation about random poll samples for the LT model.

Fact 1. *A random poll sample of the LT model is a simple path.*

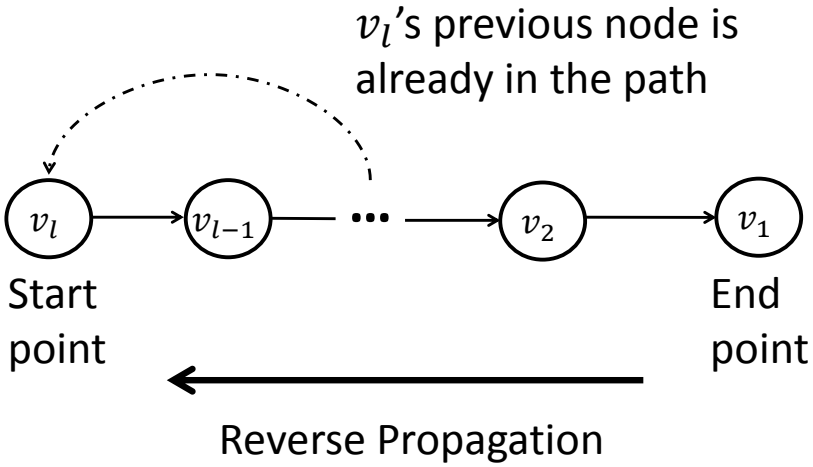


Figure 3.1: A random path. v_i is the previous vertex of v_{i-1}.

RATIONALE. In the equivalent "live-edge" selection process of the LT model, each vertex selects at most one incoming edge as a live edge. In the polling process, a random poll sample is the set of vertices that can be reversely reachable from a randomly picked vertex v via live edges. Thus, the vertices in a random poll sample together form a simple path.

Fig. 3.1 illustrates a random poll sample. The end point v_1 is picked in random at the beginning of the polling process. Then the path is generated by reversely propagating from v_1. The reverse propagation ends at v_l because v_l picks one of the vertices already in the path as its previous vertex. Note that the situation that v_l does not pick any previous vertices can be regarded as v_l picks itself as the previous vertex.

For a random poll sample, suppose the starting vertex is v_l, we also store the previous vertex picked by v_l, which is useful in our algorithm for updating random poll samples maintained. Clearly the space complexity of a poll sample is $O(L)$ where L is the number of vertices in the poll sample. We maintain an inverted index on all random poll samples so that we can access all the random poll samples passing a vertex. Moreover, we assume that the whole graph is stored and maintained in a way allowing random access to every vertex and its in-neighbors. It is not difficult to verify that the expected number of vertices of a poll sample is $\bar{I}$, the average individual influence in the network. Thus, the expected space cost of M poll samples and the inverted index is $O(M\bar{I} + n)$.

When there is an edge weight update $(u, v, +/-, \Delta, t)$ at time t, our incremental algorithm works as follows. Denote by w_{uv}^t the edge weight of (u, v) and W_v^t the total weight of v at time t. We first update the edge weight of (u, v) and the total weight of v in the graph. Then, we consider the following two cases.

1. If the update is a weight increase $(u, v, +, \Delta, t)$, we retrieve all poll samples passing v using the inverted index. For each poll sample retrieved, with probability $\frac{\Delta}{W_v^t}$ it is

rerouted from v. If a poll sample is rerouted, the previous vertex of v is set to u and we keep reversely propagating until no new vertices can be reversely reached.

2. If the update is a weight decrease $(u, v, -, \Delta, t)$, we retrieve all poll samples passing v where the previous vertex of v is u. Each retrieved poll sample is rerouted from v with probability $\frac{\Delta}{w_{uv}^{t-1}}$. If a poll sample is rerouted, we choose u' among the in-neighbors of v at time t as the previous vertex of v with probability $\frac{w_{u'v}^t}{W_v^t}$. We keep reversely propagating until no new vertices can be reversely reached.

When rerouting random poll samples, we use random access to obtain the vertices and the in-neighbors of them in the graph. We also update the inverted index.

The update operations are similar to Reservoir Sampling [107]. The following Theorem demonstrates the correctness of our incremental updating algorithm.

Theorem 1. *At any time t, after our incremental maintenance of the poll samples under the LT model as described in this section, the probability of a poll sample h appearing in our polling sketch is always* $\Pr(h \mid G^t)$.

Proof. To prove the theorem, we only need to prove that at any time t, the probability that v selects its in-neighbor u is $pp_{uv}^t = \frac{w_{uv}^t}{W_v^t}$.

We only need to consider the basic case where, at time t, there is at most one edge weight update. A general case of multiple weight updates can be simply treated as a series of the basic case.

We prove by induction. Apparently, at time 0, when the network has no edges, the theorem holds. Assume when $t = k-1$ ($k \geq 1$), the probability that v selects its in-neighbor u is $pp_{uv}^{k-1} = \frac{w_{uv}^{k-1}}{W_v^{k-1}}$. When $t = k$, three possible situations may arise.

Case 1: There is no update on any incoming edges of v. In such a case, for each u that is an in-neighbor of v, $pp_{uv}^k = pp_{uv}^{k-1} = \frac{w_{uv}^{k-1}}{W_v^{k-1}} = \frac{w_{uv}^k}{W_v^k}$.

Case 2: An edge weight increase $(u, v, +, \Delta, k)$ happens at time $t = k$. So, $w_{uv}^k = w_{uv}^{k-1} + \Delta$ and $W_v^k = W_v^{k-1} + \Delta$. For u, we have

$$pp_{uv}^k = pp_{uv}^{k-1}(1 - \frac{\Delta}{W_v^k}) + \frac{\Delta}{W_v^k} = \frac{w_{uv}^{k-1}}{W_v^{k-1}} \frac{W_v^k - \Delta}{W_v^k} + \frac{\Delta}{W_v^k} = \frac{w_{uv}^k}{W_v^k}$$

For any other in-neighbor u' of v, at time $t = k$,

$$pp_{u'v}^k = pp_{u'v}^{k-1}(1 - \frac{\Delta}{W_v^k}) = \frac{w_{u'v}^{k-1}}{W_v^{k-1}} \frac{W_v^k - \Delta}{W_v^k} = \frac{w_{u'v}^k}{W_v^k}$$

Case 3: An edge weight decrease $(u, v, -, \Delta, t)$ happens at time $t = k$. Note that $w_{uv}^k = w_{uv}^{k-1} - \Delta$ and $W_v^k = W_v^{k-1} - \Delta$. For u,

$$\begin{aligned} pp_{uv}^k &= pp_{uv}^{k-1}[(1 - \frac{\Delta}{w_{uv}^{k-1}}) + \frac{\Delta}{w_{uv}^{k-1}} \frac{w_{uv}^k}{W_v^k}] \\ &= \frac{w_{uv}^{k-1}}{W_v^{k-1}}[\frac{w_{uv}^k}{w_{uv}^{k-1}} + \frac{w_{uv}^k}{w_{uv}^{k-1}} \frac{\Delta}{W_v^k}] = \frac{w_{uv}^{k-1}}{W_v^{k-1}} \frac{w_{uv}^k}{w_{uv}^{k-1}} \frac{W_v^{k-1}}{W_v^k} = \frac{w_{uv}^k}{W_v^k} \end{aligned}$$

For any in-neighbor u' of v other than u,

$$\begin{aligned} pp_{u'v}^k &= pp_{u'v}^{k-1} + pp_{uv}^{k-1} \frac{\Delta}{w_{uv}^{k-1}} \frac{w_{u'v}^k}{W_v^k} \\ &= \frac{w_{u'v}^{k-1}}{W_v^{k-1}} + \frac{w_{uv}^{k-1}}{W_v^{k-1}} \frac{\Delta}{w_{uv}^{k-1}} \frac{w_{u'v}^k}{W_v^k} = \frac{w_{u'v}^k}{W_v^{k-1}} \frac{W_v^k + \Delta}{W_v^k} = \frac{w_{u'v}^k}{W_v^k} \end{aligned}$$

By treating v as also an in-neighbor of v itself and thus w_v is w_{vv}, we can prove the case when the weight update is on w_v. □

Suppose we have M poll samples in the polling sketch maintained. The expected number of poll samples needed to be retrieved is $\frac{MI_v^{t-1}}{n} \ll M$ for an update $(u, v, +/-, \Delta, t)$, where I_v^{t-1} is the influence spread of v at time $t - 1$. Only a small fraction of the retrieved poll samples need to be updated. Specifically, the expected number of poll samples updated is $\frac{MI_v^{t-1}\Delta}{nW_v^t} \ll M$ for a weight increase update $(u, v, +, \Delta, t)$, and $\frac{MI_v^{t-1}\Delta}{nW_v^{t-1}} \ll M$ for a weight decrease update $(u, v, -, \Delta, t)$. Clearly the cost of incremental maintenance is much less than re-generating M poll samples from scratch.

3.3.2 Updating under the IC Model

The idea of updating poll samples under the IC model is similar to [84]. We briefly introduce the idea in this section.

Rather than a simple path, a random poll sample in the IC model is a random connected component. Fig. 3.2 illustrates an example. Suppose the start point (the randomly picked vertex at the beginning of a poll) of a poll sample is v_1, then each vertex in this poll sample can be reversely reachable from v_1 via live edges.

For a random poll sample, we not only record the vertices in it but also all live edges among those vertices. We categorize live edges into two classes, namely BFS edges and cross edges. When a poll sample is being generated by reversely propagating from the start point in a breadth-first search manner, if a live edge (v_i, v_j) makes v_i propagated for the first time, (v_i, v_j) is labeled as a BFS edge; otherwise it is labeled as a cross edge. For each vertex in a poll sample, we use an adjacent list to store all live edges pointing to it. We also treat every vertex as a string and keep all vertices in a poll sample in a prefix tree for fast retrieving a vertex and the address of its adjacent list of live edges. The major

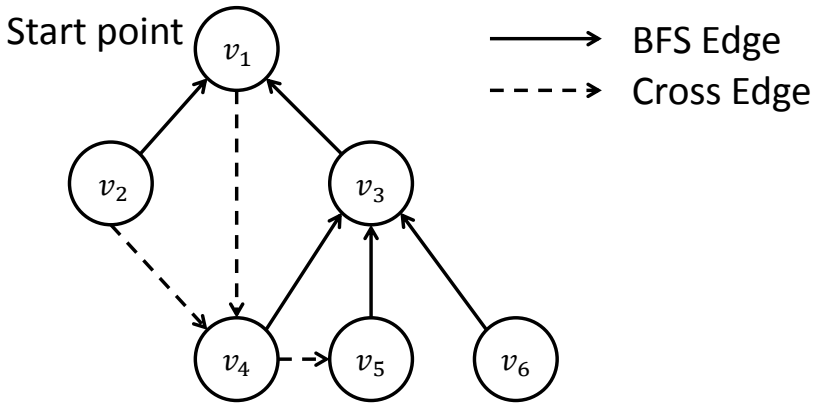


Figure 3.2: A random poll sample of the IC model is a random connected component.

difference of our data structure for storing a poll sample to the one in [84] is we do not store the propagation probabilities on live edges in a poll sample, while [84] does. We only store propagation probabilities in the graph data structure. This is obviously an improvement in space because the propagation probability of an edge is only stored once in our method.

Like the LT model, for the IC model, we also maintain an inverted index on all random poll samples so that we can access all poll samples containing a vertex. Since in the "live-edge" process of the IC model, every edge is picked independently, when there is an update $(u, v, +/-, \Delta, t)$ at time t, status ("live" or "dead") of edges other than (u, v) in poll samples stay the same. Thus, we have the following incremental maintenance,

1. If the update is a weight increase $(u, v, +, \Delta, t)$, we retrieve all poll samples passing v using the inverted index. For each poll sample retrieved, if (u, v) is not a live edge of it, we add (u, v) as a live edge to it with probability $\frac{\Delta}{1-w_{uv}^{t-1}}$. After adding (u, v), if u does not belong to this poll sample at time $t-1$, we further extend this poll sample by reversely propagating from u in a breadth-first search manner.
2. If the update is a weight decrease $(u, v, -, \Delta, t)$, we retrieve all poll samples passing v. If a retrieved poll sample contains a live edge (u, v), with probability $\frac{\Delta}{w_{uv}^{t-1}}$ we remove (u, v). If (u, v) is removed, we traverse from the start point v_1 via live edges other than (u, v) of this poll sample to find all vertices reversely reachable from v_1 and all live edges among them. Then, this poll sample is updated to one containing only those vertices and live edges we find.

Similar to the LT model, after updating the poll samples, we also update the inverted index. Clearly, our incremental maintenance ensures that, for each edge (u, v) at time t, if v is a vertex of a poll sample, the probability that (u, v) is a live edge of this poll sample is w_{uv}^t. So the same as the LT model, our incremental maintenance ensures that the probability of a poll sample h appearing in our polling sketch is always $\Pr(h \mid G^t)$ at any time t.

Theorem 2. *At any time t, after our incremental maintenance of the random poll samples under the IC model as described in this section, the probability of a poll sample h appearing in our polling sketch is always* $\Pr(h \mid G^t)$.

In our incremental maintenance, we need to find out if an edge (u, v) is a live edge in a poll sample. Suppose the number of vertices in a poll sample is L. Because normally the length of a vertex id is a constant, given an edge (u, v), using the prefix tree we can find the address of v's adjacent list in $O(1)$ time. Then a linear search is performed to find out if (u, v) is a live edge. In practice propagation probabilities are often small and $\sum_{u \in N^{in}(u)} w_{uv}$ is often a small constant. Therefore, in practice the average complexity of the linear search is $O(1)$ and in total we only need $O(1)$ time to decide if (u, v) is a live edge in a poll sample. Moreover, the space complexity of the poll sample is $O(L)$ in practice since every vertex only has a constant number of live edges pointing to it. Similar to the LT model, maintaining M poll samples and the inverted index under the IC model takes $O(M\bar{I} + n)$ space in expectation, where $\bar{I}$ is the average individual influence.

For the second situation when a live edge (u, v) is deleted, it is not always necessary to traverse from the start point, which takes $O(L)$ time if there are L vertices in the poll sample. It is easy to see that removing cross edges does not change the connectivity of vertices in a poll sample. Thus, if the removed live edge is labeled as a cross edge, we do not need to further update the poll sample.

Similar to LT model, under IC model, if we have M poll samples in the polling sketch maintained, the expected number of poll samples needed to be retrieved is $\frac{MI_v^{t-1}}{n} \ll M$ for an update $(u, v, +/-, \Delta, t)$ and only a small fraction of the retrieved poll samples need to be updated. The expected number of poll samples containing a live edge (u, v) is $\frac{MI_v^{t-1} w_{uv}^{t-1}}{n}$ and the expected number of poll samples that do not contain (u, v) as a live edge is $\frac{MI_v^{t-1}(1-w_{uv}^{t-1})}{n}$. Therefore, when there is an update on the edge (u, v), no matter it is weight increase or weight decrease, the expected number of poll samples needed to be updated is $\frac{MI_v^{t-1}\Delta}{n} \ll M$. Clearly the cost of incremental maintenance is much less than re-generating M poll samples from scratch.

3.4 Tracking Threshold-based Influential Vertices

A natural problem setting of finding influential vertices is to find all vertices whose influence spread is at least T, where T is a user-specified threshold. In this section, we discuss how to use random poll samples to approximate the desired result.

Before our discussion, we clarify that our problem is not Heavy Hitters [27] even when we treat the influence spread of a vertex as the "frequency/popularity" of an element. First, the definitions of "frequency" are different and have dramatically different properties. In Heavy Hitters, a stream of items is a multiset of elements and the frequency of an element is its multiplicity over the total number of items. Thus, the sum of frequencies of all elements

is 1, which means there are at most $1/\phi$ elements with frequency passing a threshold ϕ. In our problem, if we define the "frequency" of a vertex v as I_v/n, the value of $\sum_{v\in V}\frac{I_v}{n}$ is not necessarily 1. Actually one can easily prove that computing $\sum_{v\in V} I_v$ is #P-hard because computing I_v is #P-hard. As a result, normalizing I_v is difficult. Thus, given any influence threshold $T < n$, we cannot have an upper bound on the number of vertices that have influence greater than T. Also, the input of our problem is a stream of edge updates but not a stream of insertion/deletion of vertices (elements). Moreover, the influence of a vertex is not a simple aggregation of weights on the associated edges. In terms of technical solutions, it is hard to use a sublinear space to convert an update of edge weight to a list of insertions/deletions of vertices. As illustrated in Section 3.3, we need both the graph and poll samples to decide which vertices should be increased/decreased in frequency by an edge update. This is very different from the settings of Heavy Hitters where only a sublinear space is allowed, while the graph itself already takes space $\Omega(n)$. We also need to access a number of poll samples, while in Heavy Hitters only counters of elements are allowed to be kept in memory.

Due to the #P-hardness of computing influence spread under the LT model [21], it is not likely that we can find in polynomial time the exact set of vertices whose influence spread is at least T. Thus, we turn to algorithms that allow controllable small errors. Specifically, we ensure that the recall of the set of vertices found by our algorithm is 100% and we tolerate some false positive vertices. Moreover, the influence spread of those false positive vertices should take a high probability to have a lower bound that is not much smaller than T. We set the lower bound to $T - \epsilon n$, where ϵ controls the error.

According to Corollary 0.1, the larger M, the more accurate the unbiased estimator $n\frac{\mathcal{D}(u)}{M}$. Thus, the intuition of deciding M is to make sure that, for each u, $n\frac{\mathcal{D}(u)}{M}$ is large enough when $I_u \geq T$, and small enough when $I_u \leq T - \epsilon n$.

We first show that $n\frac{\mathcal{D}(u)}{M}$ is not likely to be too much smaller than T if $I_u \geq T$ and M is large enough.

Lemma 1. *With M random poll samples, if $I_u \geq T$, with probability at least $1-\exp(-\frac{M\epsilon^2 n}{8T})$, $n\frac{\mathcal{D}(u)}{M} \geq T - \frac{\epsilon n}{2}$.*

Proof. If $I_u \geq T$, we have

$$\begin{aligned}\Pr\{n\frac{\mathcal{D}(u)}{M} \leq T - \frac{\epsilon n}{2}\} &= \Pr\{n\frac{\mathcal{D}(u)}{M} \leq (1 - \frac{I_u - T + \frac{\epsilon n}{2}}{I_u})I_u\} \\ &\leq \exp\Big\{-\frac{M(I_u - T + \frac{\epsilon n}{2})^2}{2nI_u}\Big\}\end{aligned}$$

$\frac{(I_u - T + \frac{\epsilon n}{2})^2}{I_u}$ is non-decreasing with respect to I_u when $I_u \geq T$. Thus,

$$\Pr\{n\frac{\mathcal{D}(u)}{M} \leq T - \frac{\epsilon n}{2}\} = \exp(-\frac{M\epsilon^2 n}{8T})$$

□

Similarly, if $I_u \leq T - \epsilon n$, the probability that $n\frac{\mathcal{D}(u)}{M}$ is abnormally large is pretty small when M is large.

Lemma 2. *With M random poll samples, if $I_u \leq T - \epsilon$, with probability at least $1 - 2\exp(-\frac{M\epsilon^2 n}{12T})$, $n\frac{\mathcal{D}(u)}{M} \leq T - \frac{\epsilon n}{2}$.*

Proof. We prove that if $I_u \leq T - \epsilon n$, $\Pr\{n\frac{\mathcal{D}(u)}{M} - I_u \geq \frac{\epsilon n}{2}\} \leq 2\exp(-\frac{M\epsilon^2 n}{12T})$. Note that $n\frac{\mathcal{D}(u)}{M} - I_u \leq \frac{\epsilon n}{2}$ is a sufficient condition for $n\frac{\mathcal{D}(u)}{M} \leq T - \frac{\epsilon n}{2}$ when $I_u \leq T - \epsilon n$.

First, suppose $T \geq \frac{3\epsilon n}{2}$, which means $\frac{\epsilon n}{2} \leq T - \epsilon n$. There are two possible cases.

Case 1: $\frac{\epsilon n}{2} \leq I_u \leq T - \epsilon n$. Then,

$$\begin{aligned}\Pr\{|n\frac{\mathcal{D}(u)}{M} - I_u| \geq \frac{\epsilon n}{2}\} &= \Pr\{|M\frac{\mathcal{D}(u)}{M} - \frac{MI_u}{n}| \geq \frac{\epsilon M}{2}\} \\ &\leq 2\exp\{-\frac{1}{3}\frac{MI_u}{n}\frac{\epsilon^2 n^2}{4I_u^2}\} \leq 2\exp(-\frac{M\epsilon^2 n}{12T})\end{aligned}$$

Case 2: $I_u \leq \frac{\epsilon n}{2}$. Then,

$$\begin{aligned}\Pr\{n\frac{\mathcal{D}(u)}{M} - I_u \geq \frac{\epsilon n}{2}\} &= \Pr\{M\frac{\mathcal{D}(u)}{M} - \frac{MI_u}{n} \geq \frac{\epsilon M}{2}\} \\ &\leq \exp\{-\frac{1}{(2+\frac{2}{3})\frac{\epsilon n}{2I_u}}\frac{MI_u}{n}\frac{\epsilon^2 n^2}{4I_u^2}\} \\ &\leq \exp\{-\frac{3M\epsilon}{16}\} \leq 2\exp(-\frac{M\epsilon^2 n}{12T})\end{aligned}$$

Second, if $T \leq \frac{3\epsilon n}{2}$, for all $I_u \leq T - \epsilon n$, $I_u \leq \frac{\epsilon n}{2}$. Then, all $I_u \leq T - \epsilon n$ fall into Case 2 above and the lemma still holds. □

Because $\exp(-\frac{M\epsilon^2 n}{8T}) \leq 2\exp(-\frac{M\epsilon^2 n}{12T})$, by applying Boole's inequality (that is, the Union Bound), with probability at least $1 - 2n \cdot \exp(-\frac{M\epsilon^2 n}{12T})$, every $n\frac{\mathcal{D}(u)}{M}$ satisfies the conditions in Lemmas 1 and 2. Therefore, we have the following theorem on the sample size M for finding vertices whose influence spread is at least T.

Theorem 3. *By setting the number of random poll samples $M = \frac{12T}{n\epsilon^2}\ln\frac{2n}{\delta}$, with probability at least $1 - \delta$ the following conditions hold for every vertex u.*

1. *If $I_u \geq T$, then $n\frac{\mathcal{D}(u)}{M} \geq T - \frac{\epsilon n}{2}$*
2. *If $I_u < T - \epsilon n$, then $n\frac{\mathcal{D}(u)}{M} < T - \frac{\epsilon n}{2}$*

One nice property of M in Theorem 3 is that, given n, T, ϵ and δ, M is a constant. Therefore, when we track vertices of influence spread at least T in a dynamic network, no matter how the network changes, the sample size M remains the same. Moreover, based on Theorem 3, every time when a query of influential vertices (where the threshold is T) is issued, we return $S = \{u \mid \frac{\mathcal{D}(u)}{M} < \frac{T}{n} - \frac{\epsilon}{2}\}$ as the set of influential vertices.

Algorithm 3 Sampling Sufficient Poll Samples for Top-K Influential Individuals with Absolute Errors

Input: $G = \langle V, E, w\rangle$, ϵ, δ and $\mathcal{R}$ which is a set of random poll samples
Output: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

while $|\mathcal{R}_1| < \frac{48\times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$ **do**
  Sample a random poll sample and add to $\mathcal{R}_1$
end while
$x \leftarrow \frac{|\mathcal{R}_1|\epsilon^2}{48 \ln \frac{2n}{\delta}}$
while $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \geq x - \epsilon$ **do**
  Sample a random poll samples and add to $\mathcal{R}_1$
  $x \leftarrow \frac{|\mathcal{R}_1|\epsilon^2}{48 \ln \frac{2n}{\delta}}$
end while
Adjust the size of $\mathcal{R}_2$ to make $|\mathcal{R}_2| = |\mathcal{R}_1|$
return $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

3.5 Tracking Top-k Influential Vertices with Absolute Errors

Another useful problem setting is to find the top-k influential vertices, where k is a user-specified parameter.

Denote by I^k the influence spread of the k-th most influential vertex. Extracting top-k influential individual vertices equals extracting all vertices whose influence spread is at least I^k. Again, due to the #P-hardness of influence computation, we probably have to tolerate errors in the result when designing algorithms. Similar to Section 3.4, we hope the result returned by our algorithm contains all real top-k vertices, and for each false-positive vertex returned, its influence spread is no smaller than $I^k - \epsilon n$ with a high probability.

Unlike the task in section 3.4, we do not know the threshold I^k in advance, which makes the top-k tack more challenging. Thus, the intuition of our idea to solve the problem is that, if we have enough samples, we can bound I^k within a small range. Thus, in the following of this section, we first give a method of how to decide a proper sample size, then introduce how to extract influential vertices based on the current polling sketch for the top-k task. At last, we prove the effectiveness and efficiency of our method.

3.5.1 Deciding a Proper Sample Size

We first split the polling sketch $\mathcal{R}$ into two disjoint parts, $\mathcal{R}_1$ and $\mathcal{R}_2$. Thus, $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Then we use $\mathcal{R}_1$ to decide the size of $\mathcal{R}$, and $\mathcal{R}_2$ to extract influential vertices. The reason of doing so will be illustrated in Section 3.5.3.

Denote by $\mathcal{D}_1^*$ the greatest $\mathcal{D}_1(u)$ value in the polling sketch. Our method uses $\mathcal{D}_1^*$ as the signal to decide if the current sample size is proper or not. Specifically, Algorithm 3 describes how to ensure that we have enough poll samples, and Algorithm 4 helps us delete

Algorithm 4 Deleting Redundant Poll Samples for Top-K Influential Individuals with Absolute Errors

Input: $G = \langle V, E, w\rangle$, ϵ, δ and $\mathcal{R}$ which is a set of random poll samples
Output: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

while $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} < x - \epsilon \wedge |\mathcal{R}_1| > \frac{48\times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$ **do**
  $h \leftarrow$ the last poll sample of $\mathcal{R}_1$
  Delete h from $\mathcal{R}_1$
  if $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \geq x - \epsilon \vee |\mathcal{R}_1| < \frac{48\times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$ **then**
    Add h back to $\mathcal{R}_1$
    break
  end if
end while
Adjust the size of $\mathcal{R}_2$ to make $|\mathcal{R}_2| = |\mathcal{R}_1|$
return $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

Algorithm 5 Collecting Top-k Influential Individual Vertices with Absolute Errors

Input: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ maintained by Algorithm 3 and Algorithm 4
Output: A set of influential vertices S

return $S \leftarrow \{u \mid \frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} \geq \frac{\mathcal{D}_2^k}{|\mathcal{R}_2|} - \frac{\epsilon}{2}\}$

redundant poll samples. In Section 3.7, we will introduce how to efficiently maintain the value of $\mathcal{D}_1^*$ such that we can retrieve $\mathcal{D}_1^*$ in $O(1)$ time.

Suppose there is an infinite series of poll samples $\{h_1, h_2, h_3, ...\}$, where every h_i is independently and randomly generated. What our algorithms do can be considered as keep adding poll samples to the polling sketch $\mathcal{R}_1$ from $\{h_1, h_2, h_3, ...\}$, until the first time when $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} < x - \epsilon$, where $x = \frac{|\mathcal{R}_1|\epsilon^2}{48 \ln \frac{2n}{\delta}}$.

3.5.2 Extracting Influential Vertices

We use $\mathcal{R}_2$ to collect influential vertices for the top-k task. Denote by $\mathcal{D}_2^k$ the k-th greatest $\mathcal{D}_2(u)$ for all $u \in V$. We return all vertices u such that $\mathcal{D}_2(u)$ is greater or just slightly smaller than $\mathcal{D}_2^k$. Algorithm 5 describes our method for collecting influential vertices.

3.5.3 Proofs of Effectiveness and Efficiency

We prove the effectiveness and efficiency of our algorithms in this section. We show that by maintaining $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ using Algorithm 3 and Algorithm 4, $\mathcal{R}_2$ has enough poll samples to let Algorithm 5 extract influential vertices with a good quality guarantees. We also analyze that with high probability, the size of $\mathcal{R}$ is not too large.

First, if we have enough poll samples in $\mathcal{R}_2$, $n\frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|}$ will not deviate from its expectation I_u too much for all $u \in V$. Denote by I^* the greatest individual influence of all vertices.

Lemma 3. *If there are* $M_2 = |\mathcal{R}_2| \geq \frac{48I^*}{n\epsilon^2} \ln \frac{2n}{\delta}$ *poll samples in* $\mathcal{R}_2$*, then with probability at least* $1-\delta$*,* $|n\frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} - I_u| \leq \frac{\epsilon n}{4}$*, for all* $u \in V$*.*

Proof. Suppose we have M_2 poll samples in $\mathcal{R}_2$. We need to consider two possible cases to apply the Martingale Inequalities in Corollary 0.1.

Case 1: If $I_u \geq \frac{\epsilon n}{4}$

$$\Pr\{|n\frac{\mathcal{D}_2(u)}{M_2} - I_u| \geq \frac{\epsilon n}{4}\} \leq 2\exp\{-\frac{1}{3}\frac{\epsilon^2 n^2}{16 I_u^2}\frac{M_2 I_u}{n}\} \leq 2\exp(-\frac{M_2 n\epsilon^2}{48I^*})$$

Case 2: If $I_u \leq \frac{\epsilon n}{4}$

$$\begin{aligned}\Pr\{|n\frac{\mathcal{D}_2(u)}{M_2} - I_u| \geq \frac{\epsilon n}{4}\} &= \Pr\{n\frac{\mathcal{D}_2(u)}{M_2} - I_u \geq \frac{\epsilon n}{4}\} \\ &\leq \exp(-\frac{3}{8}\frac{\epsilon n}{4I_u}\frac{M_2 I_u}{n}) \\ &= \exp(-\frac{3M_2\epsilon}{32}) \leq 2\exp(-\frac{M_2 n\epsilon^2}{48I^*})\end{aligned}$$

When $M_2 = \frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$, $2\exp(-\frac{M_2 n\epsilon^2}{48I^*}) = \frac{\delta}{n}$. Applying the Union bound, we have that with probability at least $1-\delta$, for all $u \in V$, $|n\frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} - I_u| \leq \frac{\epsilon n}{4}$. □

Based on Lemma 3, we prove that if $\mathcal{R}_2$ has enough poll samples, then the set of influential vertices extracted by Algorithm 5 has a provable quality guarantee.

Lemma 4. *If there are* $M_2 \geq \frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$ *poll samples in* $\mathcal{R}_2$*, then with probability at least* $1-\delta$ *the set of influential vertices* $S = \{u \mid \frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} \geq \frac{\mathcal{D}_2^k}{|\mathcal{R}_2|} - \frac{\epsilon}{2}\}$ *returned by Algorithm 5 can achieve: (1) For all* $u \in V$*, if* $I_u \geq I^k$*, then* $u \in S$*; and (2) For all* $u \in V$*, if* $I_u < I^k - \epsilon n$*, then* $u \notin S$*.*

Proof. First, according to Lemma 3, with probability at least $1-\delta$, we have $|n\frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} - I_u| \leq \frac{\epsilon n}{4}$ for all $u \in V$.

When $|n\frac{\mathcal{D}_2(u)}{|\mathcal{R}_2|} - I_u| \leq \frac{\epsilon n}{4}$ for all $u \in V$, we can infer that $\frac{I^k}{n} - \frac{\epsilon}{4} \leq \frac{\mathcal{D}_2^k}{M_2} \leq \frac{I^k}{n} + \frac{\epsilon}{4}$. That is because there are at least k vertices u such that $\frac{\mathcal{D}_2(u)}{M_2} \geq \frac{I^k}{n} - \frac{\epsilon}{4}$ and there are at most k vertices v such that $\frac{\mathcal{D}_2(v)}{M_2} \geq \frac{I^k}{n} + \frac{\epsilon}{4}$.

If $I_u \geq I^k$, we have $\frac{\mathcal{D}_2(u)}{M_2} \geq \frac{I_u}{n} - \frac{\epsilon}{4} \geq \frac{I^k}{n} - \frac{\epsilon}{4} \geq \frac{\mathcal{D}_2^k}{M_2} - \frac{\epsilon}{2}$. Thus, when $I_u \geq I^k$, $u \in S$.

If $I_u \leq I^k - \epsilon n$, we have $\frac{\mathcal{D}_2(u)}{M_2} \leq \frac{I_u}{n} + \frac{\epsilon}{4} \leq \frac{I^k}{n} - \frac{3\epsilon}{4} \leq \frac{\mathcal{D}_2^k}{M_2} - \frac{\epsilon}{2}$. Thus, when $I_u < I^k - \epsilon n$, $u \notin S$. □

Now the problem is if $\mathcal{R}_2$ has at least $\frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$ poll samples. We prove that if the size of the polling sketch $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ is maintained by our Algorithm 3 and Algorithm 4, with high probability, the number of poll samples in $\mathcal{R}_2$ is not only at least $\frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$ poll samples, but also very close to $\frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$.

Lemma 5. *After applying Algorithm 3 to adjust the sample size of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, with probability at least $1 - o(\frac{1}{n^{23}})$, $|\mathcal{R}_2| \geq \frac{48I^*}{n\epsilon^2} \ln \frac{2n}{\delta}$.*

Proof. Since $|\mathcal{R}_2| = |\mathcal{R}_1|$, we only need to prove that when $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} < x - \epsilon$ for the first time, $\Pr\{|\mathcal{R}_1| < \frac{48x}{\epsilon^2} \ln \frac{2n}{\delta}\} \leq o(\frac{1}{n^{23}})$.

Suppose u is a vertex with the maximum influence, which means $I_u = I^*$. If $\mathcal{R}_1$ has $\frac{48x}{\epsilon^2} \ln \frac{2n}{\delta} < \frac{48I^*}{n\epsilon^2} \ln \frac{2n}{\delta}$ poll samples, then $xn < I^*$. Applying Corollary 0.1, we have

$$\begin{aligned}
\Pr\{\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \leq x - \epsilon \mid xn < I^*\} &\leq \Pr\{\frac{\mathcal{D}_1(u)}{|\mathcal{R}_1|} \leq x - \epsilon\} \\
&= \Pr\{\frac{\mathcal{D}_1(u)}{|\mathcal{R}_1|} \leq (1 - \frac{\epsilon}{x})x\} \\
&\leq \Pr\{n\frac{\mathcal{D}_1(u)}{|\mathcal{R}_1|} \leq (1 - \frac{\epsilon}{x})I^*\} \\
&\leq \exp(-\frac{(\frac{\epsilon}{x})^2|\mathcal{R}_1|I^*}{2n}) \leq \exp(-\frac{(\frac{\epsilon}{x})^2|\mathcal{R}_1|x}{2})
\end{aligned}$$

Since Algorithm 3 requires that $|\mathcal{R}_1| \geq \frac{48\times 4\epsilon}{\epsilon^2} \ln \frac{2n}{\delta}$, $\exp(-\frac{(\frac{\epsilon}{x})^2|\mathcal{R}_1|x}{2}) \leq (\frac{\delta}{2n})^{24}$.

Algorithm 3 fails to collect enough poll samples for $\mathcal{R}_1$ if the condition $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \leq x - \epsilon$ holds when $xn < I^*$. Apparently, Algorithm 3 checks if $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \leq x - \epsilon$ for at most $\frac{48}{\epsilon^2} \ln \frac{2n}{\delta}$ times. Therefore, applying the union bound, the failure probability of Algorithm 3 is at most $\frac{48}{\epsilon^2} \ln \frac{2n}{\delta} \times (\frac{\delta}{2n})^{24} = o(\frac{1}{n^{23}})$. □

Combining Lemma 5 and Lemma 4, we have the effectiveness of Algorithm 5 shown in Theorem 4.

Theorem 4 (Effectiveness of Algorithm 5)**.** *With probability at least $1 - \delta - o(\frac{1}{n^{23}})$, the set of influential vertices S returned by Algorithm 5 can achieve: (1) For all $u \in V$, if $I_u \geq I^k$, then $u \in S$; and (2) For all $u \in V$, if $I_u < I^k - \epsilon n$, then $u \notin S$.*

Lemma 6. *After applying Algorithm 4 to adjust the sample size of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, with probability at least $1 - o(\frac{1}{n^{14}})$, $|\mathcal{R}_2| \leq \frac{48 \max(4\epsilon n, I^* + 2\epsilon n)}{n\epsilon^2} \ln \frac{2n}{\delta}$.*

Proof. Similar to the proof of Lemma 5, we only need to prove that with high probability, $|\mathcal{R}_1| \leq \frac{48 \max(4\epsilon n, I^* + 2\epsilon n)}{n\epsilon^2} \ln \frac{2n}{\delta}$. To prove that, it is sufficient to show that if $\mathcal{R}_1$ has $\frac{48(I^* + 2\epsilon n)}{n\epsilon^2} \ln \frac{2n}{\delta}$ poll samples, the test if $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \leq x - \epsilon$ probably will pass.

Let $\epsilon' = \frac{\epsilon}{x}$. Note that Algorithm 4 ensures that $x \geq 4\epsilon$ and $\epsilon' \leq \frac{1}{4}$. Thus, $\frac{1-\epsilon'}{2} \leq 1 - 2\epsilon'$. Let $M_1 = \frac{48(I^* + 2\epsilon n)}{n\epsilon^2} \ln \frac{2n}{\delta}$. We have two possible cases for all $u \in V$.

Case 1: If $\frac{(1-\epsilon')xn}{2} \leq I_u \leq (1-2\epsilon')xn$

$$\begin{aligned}\Pr\{\frac{\mathcal{D}_1(u)}{M_1} \geq x-\epsilon\} &= \Pr\{n\frac{\mathcal{D}_1(u)}{M_1} \leq [1+\frac{(1-\epsilon')xn-I_u}{I_u}]I_u\} \\ &\leq \exp(-\frac{1}{3}\frac{[(1-\epsilon')xn-I_u]^2}{I_u^2}\frac{M_1 I_u}{n}) \\ &\leq \exp(-\frac{\epsilon'^2 M_1 x}{3(1-2\epsilon')}) \leq \exp(-16\ln\frac{2n}{\delta}) = (\frac{\delta}{2n})^{16}\end{aligned}$$

Case 2: If $I_u \leq \frac{(1-\epsilon')xn}{2}$

$$\begin{aligned}\Pr\{\frac{\mathcal{D}_1(u)}{M_1} \geq (1-\epsilon')x\} &= \Pr\{n\frac{\mathcal{D}_1(u)}{M_1} \leq [1+\frac{(1-\epsilon')xn-I_u}{I_u}]I_u\} \\ &\leq \exp(-\frac{3}{8}\frac{[(1-\epsilon')xn-I_u]}{I_u}\frac{M_1 I_u}{n}) \\ &\leq \exp(-\frac{3(1-\epsilon')M_1 x}{16}) \\ &\leq \exp(\frac{9\ln\frac{2n}{\delta}}{2\epsilon'^2}) \leq \exp(-18\ln\frac{2n}{\delta}) = (\frac{\delta}{2n})^{18}\end{aligned}$$

Applying the Union Bound, we have that, with probability at least $1-n(\frac{\delta}{2n})^{16}$, $\mathcal{D}_1^* \leq x-\epsilon$ when $\mathcal{R}_1$ has $\frac{48(I^*+2\epsilon n)}{n\epsilon^2}\ln\frac{2n}{\delta}$ poll samples. □

Combining Lemma 5 and Lemma 6, we have the following Theorem to bound the sample size of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ maintained by Algorithm 3 and Algorithm 4. Note that sample size of $\mathcal{R}$ reflects the efficiency of our method, since the time complexity of our method is proportional to the number of poll samples maintained.

Theorem 5 (Sample Size Maintained by Algorithm 3 and Algorithm 4)**.** *If we use Algorithm 3 and Algorithm 4 to maintain the sample size of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, with probability $1-o(\frac{1}{n^{14}})$, we have $|\mathcal{R}| = \frac{96x}{\epsilon^2}\ln\frac{2n}{\delta} \geq \frac{96I^*}{n\epsilon^2}\ln\frac{\delta}{2n}$ random poll samples, and $xn \leq \max(4\epsilon n, I^*+2\epsilon n)$.*

Remark 1. One may ask why do we need $\mathcal{R}_2$ because why not apply Theorem 4 on $\mathcal{R}_1$ to extract influential vertices. The answer is that probabilistic support (implication) is not transitive[1] [96]. Even by Lemma 5 we know that $\mathcal{R}_1$ has at least $\frac{48I^*}{n\epsilon^2}\ln\frac{2n}{\delta}$ poll samples with high probability, we cannot establish the conditions in Lemma 3 for $\mathcal{R}_1$. To apply Lemma 3 on $\mathcal{R}_1$, we should not have prior knowledge about $\mathcal{D}_1^*$ in $\mathcal{R}_1$ because the sample spaces are actually different. Lemma 3 holds for $\mathcal{R}_2$, whose sample space is all possible $M_2 = |\mathcal{R}|_1$ random poll samples $\{h_1, h_2, ..., h_{M_2}\}$, while the sample space of $\mathcal{R}_1$ requires that $\frac{\mathcal{D}_1^*}{|\mathcal{R}_1|} \geq x-\epsilon$.

[1]Suppose A implies B with probability $1-\delta_1$, and B implies C with probability $1-\delta_2$. A flawed argument using the transitivity is that by applying the union bound, A implies C with probability $1-\delta_1-\delta_2$.

Remark 2. A nice property of Algorithm 3 and Algorithm 4 is that the sample size maintained is proportional to the value of I^* as proved in Theorem 5. This is different from other studies [89, 90, 87] dealing with similar issues in mining frequent itemsets from sampled transactions [2]. Instead of estimating the greatest frequency of itemsets, [89, 90, 87] just use a trivial upper bound 1 as the maximum frequency. In a real dataset, no matter it is a transaction database or a social network, the maximum itemset frequency or the highest influence spread is normally much smaller than 1 or n, the total number of vertices. Thus, the sample size in [89, 90, 87] is actually much greater than needed. Our method that estimates I^* first exploits of the unique property (the value of I^*) of every dataset.

3.6 Tracking Top-k Influential Vertices with Relative Errors

One drawback of the algorithm described in Section 3.5 is that the error ϵ controlled by the algorithm is an absolute error. Since the k-th greatest influence spread I^k is not known to us in advance, it is possible that we do not set ϵ properly to make ϵn very close to or even greater than I^k. Moreover, in a dynamic social network, the value of I^k often changes as the network updates. In such cases, controlling a predefined absolute error ϵ in the algorithms in Section 3.5 may be meaningless to us sometimes. Thus, in this section, we develop algorithm that controls a relative error ϵ, which means the smallest influence spread in the returned set of users is at least $I^k(1-\epsilon)$.

Similar to Section 3.5, in this section, we first describe how to decide a proper sample size and how to extract influential users given the current polling sketch, then we prove the effectiveness and efficiency of our method. We also illustrate the advantages of our new algorithm to the algorithm in Section 3.5.

3.6.1 Deciding a Proper Sample Size

Similar to Section 3.5, we also divide the polling sketch $\mathcal{R}$ into two disjoint partitions $\mathcal{R}_1$ and $\mathcal{R}_2$. We use $\mathcal{R}_1$ to control the sample size of $\mathcal{R}$ and use $\mathcal{R}_2$ to extract influential vertices.

In our method, $\mathcal{D}_1^k$, the k-th greatest $\mathcal{D}_1(u)$ for $u \in V$, is used as the signal to decide if the current sample size is proper or not. How to efficiently maintain and retrieve the value of $\mathcal{D}_1^k$ will be illustrated in Section 3.7. Algorithm 6 describes our algorithm for controlling the sample size of $\mathcal{R}$, where $\Upsilon_1^{\epsilon,\frac{\delta}{n}} = 1 + (1+\epsilon)\frac{4(e-2)\ln\frac{2}{\delta}}{\epsilon^2}$ as defined in Section 2.1.4. What Algorithm 6 does is to always maintain the invariant that $\mathcal{D}_1^k = \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$.

[2] One may link finding influential vertices with finding frequent itemsets remotely due to the intuition that a vertex frequent in many poll samples is likely influential.

Algorithm 6 Adjusting Sample Size for Top-K Influential Individuals with Relative Errors

Input: $G = \langle V, E, w \rangle$, ϵ, δ and $\mathcal{R}$ which is a set of random poll samples
Output: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

1: **while** $\mathcal{D}_1^k < \lceil \Upsilon_1^{\epsilon, \frac{\delta}{n}} \rceil$ **do**
2: Sample a random poll sample and add to $\mathcal{R}_1$
3: **end while**
4: **while** $\mathcal{D}_1^k > \lceil \Upsilon_1^{\epsilon, \frac{\delta}{n}} \rceil$ **do**
5: Delete the last poll sample from $\mathcal{R}_1$
6: **end while**
7: Adjust the size of $\mathcal{R}_2$ to make $|\mathcal{R}_2| = |\mathcal{R}_1|$
8: **return** $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

Algorithm 7 Collecting Top-k Influential Individual Vertices with Relative Errors

Input: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ maintained by Algorithm 6
Output: A set of influential vertices S

1: **return** $S = \{u \mid \mathcal{D}_2(u) \geq T = \frac{1-\epsilon}{1+\epsilon} \Upsilon_1^{\epsilon, \frac{\delta}{n}}\}$

3.6.2 Extracting Influential Vertices

Similar to Section 3.5.2, when extracting influential vertices from $\mathcal{R}$, we use $\mathcal{D}_2(u)$ to decide if u should be returned or not. Algorithm 7 shows our method.

3.6.3 Proofs of Effectiveness and Efficiency

We prove the effectiveness and efficiency of our algorithms in this section. We show that by maintaining $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ using Algorithm 6, $\mathcal{R}_2$ has enough poll samples to let Algorithm 7 extract influential vertices with a good quality guarantees. Unlike Section 3.5, we prove that with high probability, the smallest influence spread of vertices returned by Algorithm 7 is only smaller than I^k by a relative ratio. We also analyze the size of $\mathcal{R}$, and compare it to the sample size maintained by Algorithm 3 and Algorithm 4 in Section 3.5.

First, we prove that when $\mathcal{D}_1^k = \lceil \Upsilon_1^{\epsilon, \frac{\delta}{n}} \rceil$ for the first time, we could use $\frac{n \Upsilon_1^{\epsilon, \frac{\delta}{n}}}{|\mathcal{R}_1|}$ to get a good estimation of I^k, which is unknown to us in advance.

Lemma 7. *For $\mathcal{R}_1$ maintained by Algorithm 6, we have (1)* $\Pr\{\frac{I^k}{n} \geq \frac{\Upsilon_1^{\epsilon, \frac{\delta}{n}}}{(1+\epsilon)|\mathcal{R}_1|}\} \geq 1 - \frac{\delta}{2}$ *and (2)* $\Pr\{\frac{I^k}{n} \leq \frac{\Upsilon_1^{\epsilon, \frac{\delta}{n}}}{(1-\epsilon)|\mathcal{R}_1|}\} \geq 1 - \frac{\delta}{2}$.

Proof. Let $V_L = \{u \mid u \in V, I_u \leq I^k\}$ and $V_U = \{u \mid u \in V, I_u \geq I^k\}$. Since I^k is the k-th largest individual influence spread, we have $|V_L| \geq n - k + 1$ and $|V_U| \geq k$. Let $M_1 = |\mathcal{R}_1|$.

First, suppose $\mathcal{R}_1$ has $M_1 < \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$. For any vertex $u \in V_L$, $M_1 < \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k} \leq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I_u}$. Based on the Stopping Rule Theorem (Corollary 0.2), for any $u \in V_L$ we have

$$\Pr\{\mathcal{D}_1(u) < \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil \mid M_1 < \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}\} > 1 - \frac{\delta}{2n}$$

Applying the union bound, we have

$$\Pr\{\forall u \in V_L, \mathcal{D}_1(u) < \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil \mid M_1 < \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}\} > 1 - \frac{\delta}{2}$$

which means when $M_1 < \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$, $\mathcal{D}_1^k < \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$ because there are at most $|V \setminus V_L| \leq k-1$ vertices with degree at least $\lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$. Thus, when $\mathcal{D}_1^k = \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$ for the first time, with probability at least $1 - \frac{\delta}{2}$ we have $M_1 \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$. Therefore, for $\mathcal{R}_1$ maintained by Algorithm 6, we have $\Pr\{\frac{I^k}{n} \geq \frac{\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)|\mathcal{R}_1|}\} \geq 1 - \frac{\delta}{2}$.

Similarly, we can obtain that when $\mathcal{R}_1$ has $M_1 \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)I^k}$ poll samples, we have

$$\Pr\{\forall u \in V_U, \mathcal{D}_1(u) \geq \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil \mid M_1 \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)I^k}\} > 1 - \frac{\delta}{2}$$

which means when $\mathcal{R}_1$ has $M_1 \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)I^k}$ poll samples, $\mathcal{D}_1^k \geq \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$ because there are at least $|V_U| \geq k$ vertices with degree at least $\lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$. Thus, when $\mathcal{D}_1^k = \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$ for the first time, with probability at least $1 - \frac{\delta}{2}$ we have $M_1 \leq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)I^k}$. Therefore, for $\mathcal{R}_1$ maintained by Algorithm 6, we have $\Pr\{\frac{I^k}{n} \leq \frac{\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)|\mathcal{R}_1|}\} \geq 1 - \frac{\delta}{2}$. □

Note that $\Pr\{\frac{I^k}{n} \geq \frac{\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)|\mathcal{R}_1|}\} \geq 1 - \frac{\delta}{2}$ implies $\Pr\{|\mathcal{R}_1| \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}\} \geq 1 - \frac{\delta}{2}$. Recall that in our algorithm, we make $|\mathcal{R}_2| = |\mathcal{R}_1|$. Thus, $\Pr\{|\mathcal{R}_2| \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}\} \geq 1 - \frac{\delta}{2}$. We show that when $|\mathcal{R}_1| \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$, the estimated influence $\frac{n\mathcal{D}_2(v)}{|\mathcal{R}_2|}$ of each vertex v does not deviate from its expectation I_v too much.

Lemma 8. *For $|\mathcal{R}_2| \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$ poll samples in $\mathcal{R}_2$, with probability at least $1 - \frac{\delta}{2}$, we have (1) for every v such that $I_v \geq I^k$, $\frac{n\mathcal{D}_2(v)}{|\mathcal{R}_2|} \geq (1-\epsilon)I^k$; and (2) for every v such that $I_v \leq (1-2\epsilon)I^k$, $\frac{n\mathcal{D}_2(v)}{|\mathcal{R}_2|} \leq I_v + \epsilon I^k$.*

Proof. Let $M_2 = |\mathcal{R}_2|$. First, when $M_2 \geq \frac{n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)I^k}$, we have

$$M_2 \geq \frac{n(1+\epsilon)4(e-2)\ln\frac{2n}{\delta}}{(1+\epsilon)\epsilon^2 I^k} \geq \frac{8n\ln\frac{2n}{\delta}}{3\epsilon^2 I^k}$$

Applying the Martingale inequalities in [103] (see Appendix), we have

1. If $I_v \geq I^k$, $\Pr\{\frac{n\mathcal{D}_2(v)}{M_2} \geq (1-\epsilon)I^k\} \leq (\frac{\delta}{2n})^{4/3} \leq \frac{\delta}{2n}$; and
2. If $I_v \leq (1-2\epsilon)I^k$, $\Pr\{\frac{n\mathcal{D}_2(v)}{M_2} \leq I_v + \epsilon I^k\} \leq \frac{\delta}{2n}$.

By applying the Union bound, the Lemma is proved. □

Based on Lemma 7 and Lemma 8, we have the following theorem that shows the effectiveness of our algorithm.

Theorem 6 (Effectiveness of Algorithm 7)**.** *With probability at least* $1-2\delta$*, Algorithm 7 returns a set of vertices* S *such that (1) all real top-k vertices are included, and (2)* $\min_{u\in S} I_u \geq (1-\frac{4\epsilon}{1+\epsilon})I^k \geq (1-4\epsilon)I^k$.

Proof. Based on Lemma 7 and Lemma 8, when $\mathcal{D}_1^k = \lceil \Upsilon_1^{\epsilon,\frac{\delta}{n}} \rceil$ and $M_2 = M_1$, with probability at least $1-2\delta$, the following conditions hold.

1. $\frac{\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)M_1} \leq \frac{I^k}{n} \leq \frac{\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1-\epsilon)M_1}$;
2. For every v such that $I_v \geq I^k$, $\frac{\mathcal{D}_2(v)}{M_2} \geq (1-\epsilon)\frac{I^k}{n}$;
3. For every v such that $I_v \leq (1-2\epsilon)I^k$, $\frac{\mathcal{D}_2(v)}{M_2} \leq \frac{I_v}{n} + \epsilon\frac{I^k}{n}$.

Under these 3 conditions, obviously, if $I_v \geq I^k$ then $D_2(v) \geq \frac{(1-\epsilon)M_2 I^k}{n} \geq \frac{1-\epsilon}{1+\epsilon}\Upsilon_1^{\epsilon,\frac{\delta}{n}} = T$. If $I_v \leq \frac{nT}{M_2} - \epsilon I^k \leq (1-2\epsilon)I^k$, then $\frac{\mathcal{D}_2(v)}{M_2} \leq \frac{I_v}{n} + \epsilon\frac{I^k}{n} \leq \frac{T}{M_2}$. Thus, if $I_v \leq \frac{nT}{M_2} - \epsilon I^k$, then $\mathcal{D}_2(v) \leq T$.

Now we prove $\frac{nT}{M_2} - \epsilon I^k$ is not much smaller than I^k, which means when we use T as a filtering threshold, the influence spread of any false positive vertex is close to the true threshold I^k. We have

$$\begin{aligned}\frac{nT}{M_2} - \epsilon I^k &= \frac{n(1-\epsilon)\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{(1+\epsilon)M_1} - \epsilon I^k \\ &\geq [\frac{(1-\epsilon)^2}{1+\epsilon} - \epsilon]I^k \\ &= (1-\frac{4\epsilon}{1+\epsilon})I^k \geq (1-4\epsilon)I^k\end{aligned}$$

Our algorithm returns the set of vertices $S = \{u \mid \mathcal{D}_2(u) \geq T\}$. Summarize the above analysis, we have that with probability at least $1-2\delta$, (1) if $I_u \geq I^k$, $u \in S$, and (2) $\min_{u\in S} I_u \geq (1-\frac{4\epsilon}{1+\epsilon})I^k \geq (1-4\epsilon)I^k$. □

Note that we prove $\min_{u \in S} I_u \geq (1-4\epsilon)I^k$ is just for showing there is a relative error bound. In practice, we use $\frac{4\epsilon}{1+\epsilon}$ to calculate the upper bound of relative error, since it is tighter than 4ϵ.

We also can bound the sample size of $\mathcal{R}$ maintained by Algorithm 6 by using Lemma 7.

Theorem 7 (Sample Size Maintained by Algorithm 6)**.** *If we use Algorithm 6 to maintain the sample size of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, with probability $1-\delta$, we have $\frac{2n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{I^k(1+\epsilon)} \leq |\mathcal{R}| \leq \frac{2n\Upsilon_1^{\epsilon,\frac{\delta}{n}}}{I^k(1-\epsilon)}$, where $\Upsilon_1^{\epsilon,\frac{\delta}{n}} = 1 + (1+\epsilon)\frac{4(e-2)\ln\frac{2n}{\delta}}{\epsilon^2}$.*

Theorem 7 show that with high probability, the sample size of the polling sketch $\mathcal{R}$ maintained by Algorithm 6 is $\Theta(\frac{n\ln\frac{n}{\delta}}{\epsilon^2 I^k})$. It is worth noting that, according to Dagum *et al.* [28], even when we know which vertex is the k-th most influential individual, to obtain an (ϵ,δ)-estimation[3] of I^k, at least $\Theta(\frac{\ln\frac{1}{\delta}}{\epsilon^2 I^k}\max\{n-I^k, \epsilon I^k\})$ poll samples are needed. Considering that normally in a real social network $I^k \ll n$, the minimum number of poll samples to achieve an (ϵ,δ)-estimation of I^k is $\Theta(\frac{n\ln\frac{1}{\delta}}{\epsilon^2 I^k})$, which is only smaller than our sample size $\Theta(\frac{n\ln\frac{n}{\delta}}{\epsilon^2 I^k})$ by at most a factor of $\ln n$.

Comparison to Section 3.5 For the top-k tracking algorithm in Section 3.5, we set the absolute error $\epsilon_1 n$ to the same value of the error in our method, which is less than $4\epsilon I^k$ according to Theorem 6 (suppose magically we know the value of I^k beforehand). Then the number of poll samples is $\Theta(\frac{I^*}{I^k}\frac{n\ln\frac{n}{\delta}}{\epsilon^2 I^k})$, where I^* is the maximum individual influence. It is obvious that $I^* \geq I^k$ and in real social networks, the gap between I^* and I^k can be large. Thus, our algorithm is normally more efficient than the top-k tracking algorithm in Section 3.5, when the errors controlled by the two methods are the same in absolute value.

3.7 Maintaining Vertices Ranking Dynamically

In this section, we introduce how to maintain all vertices always sorted by their degrees in a polling sketch $\mathcal{R}$. We also devise an algorithm to efficiently maintain and retrieve the value of $\mathcal{D}^k$, the k-the greatest degree of vertices in $\mathcal{R}$. Applying our algorithm on $\mathcal{R}_1$ in the algorithms in Section 3.5 and Section 3.6 can help us efficiently decide if the current sample size is proper or not.

To maintain all vertices sorted, the major idea is to group vertices with the same degrees in $\mathcal{R}$ together. We adopt the data structure for maximum vertex cover in a hyper graph [10]. Fig. 3.3 shows the data structure, which is a doubly linked list where every node is a doubly linked list. We call such a data structure a Linked List.

Vertices with the same degree in $\mathcal{R}$ are grouped together and stored in a doubly linked list like in Fig. 3.3. Moreover, for those vertices, we create a head node which is the start of

[3] $\hat{I}^k$ is an (ϵ,δ)-estimation of I^k if $\Pr\{|\hat{I}^k - I^k| \leq \epsilon I^k\} \geq 1-\delta$.

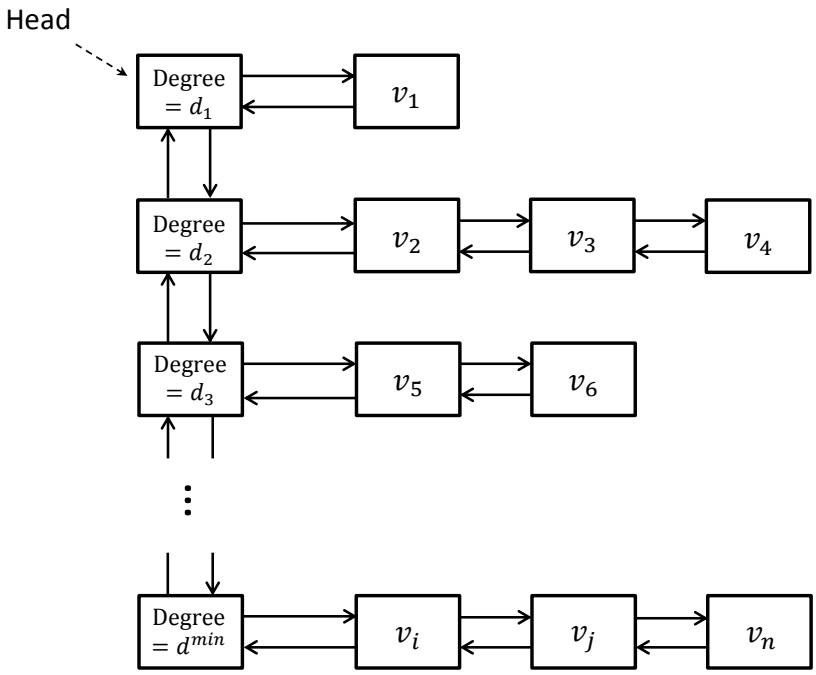


Figure 3.3: Linked List Structure, where $d_1 > d_2 > d_3 > ... > d^{min}$

the linked list containing all vertices with the same given degree. Apparently, the number of head nodes is the number of distinctive values of $\mathcal{D}(u)$ in $\mathcal{R}$. We also maintain all head nodes sorted in a doubly linked list. For each $u \in V$, we maintain its address in the doubly linked lists structure and the corresponding head node.

When a poll sample is updated, a new poll sample is generated or an existing poll sample is deleted, $\mathcal{D}(u)$ changes at most by 1 for each $u \in V$. Thus, every time when $\mathcal{D}(u)$ is updated (increased or decreased by 1), we only need $O(1)$ time to find the head node of the linked list u should be in (if such a head node does not exist now, we can create it and insert it into the doubly linked list of head nodes in $O(1)$ time) and insert it to the next of the head in $O(1)$ time. If after an update, a head node has no vertices after it, we delete it from the doubly linked list of head nodes in $O(1)$ time.

Besides maintaining all vertices sorted by their degrees in $\mathcal{R}$, we also need to maintain the value of $\mathcal{D}^k$ against the updates of the network, because in our top-k tracking algorithms, $\mathcal{D}^k$ is used as the signal[4] to decide if the current sample size is proper.

We record H_k, the head node of the doubly linked list containing vertices whose degrees in $\mathcal{R}$ are $\mathcal{D}^k$. We also need a bias b which indicates there are $k - b$ vertices u such that $\mathcal{D}(u) > \mathcal{D}^k$. Suppose due to an update, $\mathcal{D}(u)$ is increases by 1. Let $\mathcal{D}(u)^{old}$ be the value before the update and $\mathcal{D}(u)^{new}$ the value after the update. Denote by $H_k.degree$ the degree of vertices that H_k is their head node and $H_k.num$ the number of such vertices. Let $H_k.up$

[4]In Section 3.5 the signal is $\mathcal{D}^*$. Note that $\mathcal{D}^* = \mathcal{D}^k$ when we set $k = 1$.

be the head node above it and $H_k.down$ the head node below it. To update H_k and b, we have two cases depending on $\mathcal{D}(u)^{old}$.

1. If $\mathcal{D}(u)^{old} \neq H_k.degree$, we do not need to update H_k or b; and
2. If $\mathcal{D}(u)^{old} = H_k.degree$, there are two subcases according to b. If $b = 1$ before the update, we set H_k as $H_k.up$, and set b to the number of vertices in the linked list whose head is the updated H_k. If $b > 1$ before the update, we do not update H_k but we decrease b by 1.

Similarly, if $\mathcal{D}(u)$ of a vertex u decreases by 1, to update H_k and b, we have three cases depending on $\mathcal{D}(u)^{old}$.

1. If $\mathcal{D}(u)^{old} < H_k.degree$ or $\mathcal{D}(u)^{old} > H_k.degree + 1$, we do not need to update H_k and b;
2. If $\mathcal{D}(u)^{old} = H_k.degree + 1$, we do not update H_k but we increase b by 1; and
3. If $\mathcal{D}(u) = H_k.degree$, there are two subcases. If $b = H_k.num$ before the update, we set H_k as $H_k.down$ and set $b = 1$. If $b < H_k.num$ before the update, we do not update H_k or b.

Clearly, the updates on H_k and b only take $O(1)$ time when $\mathcal{D}(u)$ changes by 1 for a vertex u. After the maintenance, $\mathcal{D}^k = H_k.degree$.

When a poll sample is updated/inserted/deleted, $\mathcal{D}(u)$ changes at most by 1 for each u. We need $\Omega(1)$ time to update its inverted index and only $O(1)$ time to update the linked list data structure, H_k and b. Thus, maintaining the Linked List and the value of $\mathcal{D}^k$ do not increase the complexity of updating poll samples. Moreover, every time when we need to know the value of $\mathcal{D}^k$, we just need $O(1)$ time to retrieve it.

3.8 Experiments

In this section, we report a series of experiments on five real networks to verify our algorithms and our theoretical analysis. The experimental results demonstrate that our algorithms are both effective and efficient.

3.8.1 Experimental Settings

We ran our experiments on five real network data sets that are publicly available online (`http://konect.uni-koblenz.de/networks/`, `http://www.cs.ubc.ca/~welu/` and `http://konect.uni-koblenz.de`). Table 4.3 shows the statistics of the five data sets.

To simulate dynamic networks, for each data set, we randomly partitioned all edges exclusively into 3 groups: E_1 (85% of the edges), E_2 (5% of the edges) and E_3 (10% of the edges). We used $B = \langle V, E_1 \cup E_2 \rangle$ as the base network. E_2 and E_3 were used to simulate a stream of updates.

Table 3.2: The statistics of the data sets.

Network	#Vertices	#Edges	Average degree
wiki-Vote	7,115	103,689	14.6
Flixster	99,053	977,738	9.9
soc-Pokec	1,632,803	30,622,564	18.8
flickr-growth	2,302,925	33,140,018	14.4
Twitter	41,652,230	1,468,365,182	35.3

For the LT model, for each edge (u, v) in the base network, we set the weight to be 1. For each edge $(u, v) \in E_3$, we generated a weight increase update $(u, v, +, 1)$ (timestamps ignored at this time). For each edge $(u, v) \in E_2$, we generated one weight decrease update $(u, v, -, \Delta)$ and one weight increase update $(u, v, +, \Delta)$ where Δ was picked uniformly at random in $[0, 1]$. We randomly shuffled those updates to form an update stream by adding random time stamps. For each data set, we generated 10 different instances of the base network and update stream, and thus ran the experiments 10 times. Note that for the 10 instances, although the base networks and update streams are different, the final snapshots of them are identical to the data set itself.

For the IC model, we first assigned propagation probabilities of edges in the final snapshot, i.e. the whole graph. We set $w_{uv} = \frac{1}{\text{in-degree}(v)}$, where in-degree$(v)$ is the number of in-neighbors of v in the whole graph. Then, for each edge (u, v) in the base network, we set w_{uv} to $\frac{1}{\text{in-degree}(v)}$. For each edge $(u, v) \in E_3$, we generated a weight increase update $(u, v, +, \frac{1}{\text{in-degree}(v)})$ (timestamps ignored at this time). For each edge $(u, v) \in E_2$, we generated one weight decrease update $(u, v, -, \Delta\frac{1}{\text{in-degree}(v)})$ and one weight increase update $(u, v, +, \Delta\frac{1}{\text{in-degree}(v)})$ where Δ was picked uniformly at random in $[0, 1]$. We randomly shuffled those updates to form an update stream by adding random time stamps. For each dataset we also generated 10 instances.

For the parameters of tracking vertices of influence at least T, we set $\epsilon = 0.0002$, $\delta = 0.001$, and $T = 0.001 \times n$ for the first four data sets. We set $\epsilon = 0.001$, $\delta = 0.001$, and $T = 0.005 \times n$ for the twitter data set. For the top-k influential vertices tracking task where we control an absolute error, we set $k = 50$, $\delta = 0.001$, and $\epsilon = 0.0005$ for first four data sets. We set $k = 100$, $\delta = 0.001$, and $\epsilon = 0.0025$ for the twitter data set. The reason we have different parameter settings for the twitter data is that it has more influential vertices than other networks. For the top-k influential vertices tracking task where we control a relative error, we set $k = 50$, $\delta = 0.001$, and $\epsilon = 0.1$ which means the relative error rate controlled by our algorithm is roughly 36.5% according to Theorem 6.

All algorithms were implemented in Java and ran on a Linux machine of an Intel Xeon 2.00GHz CPU and 1TB main memory.

Table 3.3: Recall and Maximum Error of the Thresholding Task. The errors are measured in absolute influence value. "w.h.p." is short for "with high probability".

	wiki-Vote		
	Theoretical Value (w.h.p.)	Ave.±SD (LT)	Ave.±SD (IC)
Recall	100%	100%±0	100%±0
Max. Error	$0.0002 * 7115 = 1.423$	0.758±0.033	0.814±0.013
	Flixster		
	Theoretical Value (w.h.p.)	Ave.±SD (LT)	Ave.±SD (IC)
Recall	100%	100%±0	100%±0
Max. Error	$0.0002 * 99053 = 19.81$	10.81±0.46	11.79±0.85

3.8.2 Thresholding Task

We report the experimental results of the thresholding task.

Effectiveness

First, we assess the effectiveness. A challenge in evaluating the effectiveness of our algorithms is that the ground truth is hard to obtain. The existing literature of influence maximization [54, 19, 44, 104, 103, 26] always use the influence spread estimated by 20,000 times Monte Carlo (MC) simulations as the ground truth. However, such a method is not suitable for our tasks, because the ranking of vertices really matters here. Even 20,000 times MC simulations may not be able to distinguish vertices with close influence spread. As a result, the ranking of vertices may differ much from the real ranking. Moreover, the effectiveness of our algorithms has theoretical guarantees while 20,000 times MC simulations is essentially a heuristic. It is not reasonable to verify an algorithm with a theoretical guarantee using the results obtained by a heuristic method without any quality guarantees.

In our experiments, we only used wiki-Vote and Flixster to run MC simulations and compare the results to those produced by our algorithms. We used 2,000,000 times MC simulations as the (pseudo) ground truth in the hope we can get more accurate results. According to our experiments, even so many MC simulations may generate slightly different rankings of vertices in two different runs but the difference is acceptably small.

Table 3.3 reports the recall of the sets of influential vertices returned by our algorithms and the maximum errors of the false positive vertices in absolute influence value. Ave.±SD represents the average value and the standard deviation of a measurement on 10 instances. Our methods achieved 100% recall every time as guaranteed theoretically. Moreover, the real errors in influence were substantially smaller than the maximum error bound provided by our theoretical analysis. One may ask why we do not report the precision here. We argue that precision is indeed not a proper measure for our tasks when 100% recall is required. Since we can only estimate influence spreads of vertices via a sampling method due to the exact computation being #P-hard, if two vertices have close influence spreads, say $I_u = 100$

and $I_v = 99$, it is hard for a sampling method to tell the difference between I_u and I_v. Thus, if there are many vertices whose influence spreads are just slightly smaller than the threshold, it is hard to achieve a high precision when ensuring 100% recall. Moreover, with a high probability, our method guarantees that influence spreads of false positive vertices are not far away from the real threshold. Such small errors are completely acceptable in many real applications.

For the three large data sets, we did not run 2,000,000 times MC simulations to obtain the pseudo ground truth since the MC simulations are too costly. Instead, we compare the similarity between the results generated by different instances. Recall that the final snapshots of the 10 instances are the same. If the sets of influential vertices at the final snapshots of the 10 instances are similar, at least our algorithms are stable, that is, insensitive to the order of updates. To measure the similarity between two sets of influential vertices, we adopted the Jaccard similarity.

Fig. 3.4 shows the results where I1, ..., I10 represent the results of the first, ..., tenth instances, respectively. We also ran the sampling algorithm directly on the final snapshot, that is, we computed the influential vertices directly from the final snapshot using sampling without any updates. The result is denoted by ST. The results show that the outcomes from different instances are very similar, and they are similar to the outcome from ST, too. The minimum similarity in all cases is 87%.

Scalability

Fig. 3.5 shows the average running time with respect to the number of updates processed. The average is taken on the running times of the 10 instances. The time spent when the number of updates is 0 reflects the computational cost of running the sampling algorithm on the base network.

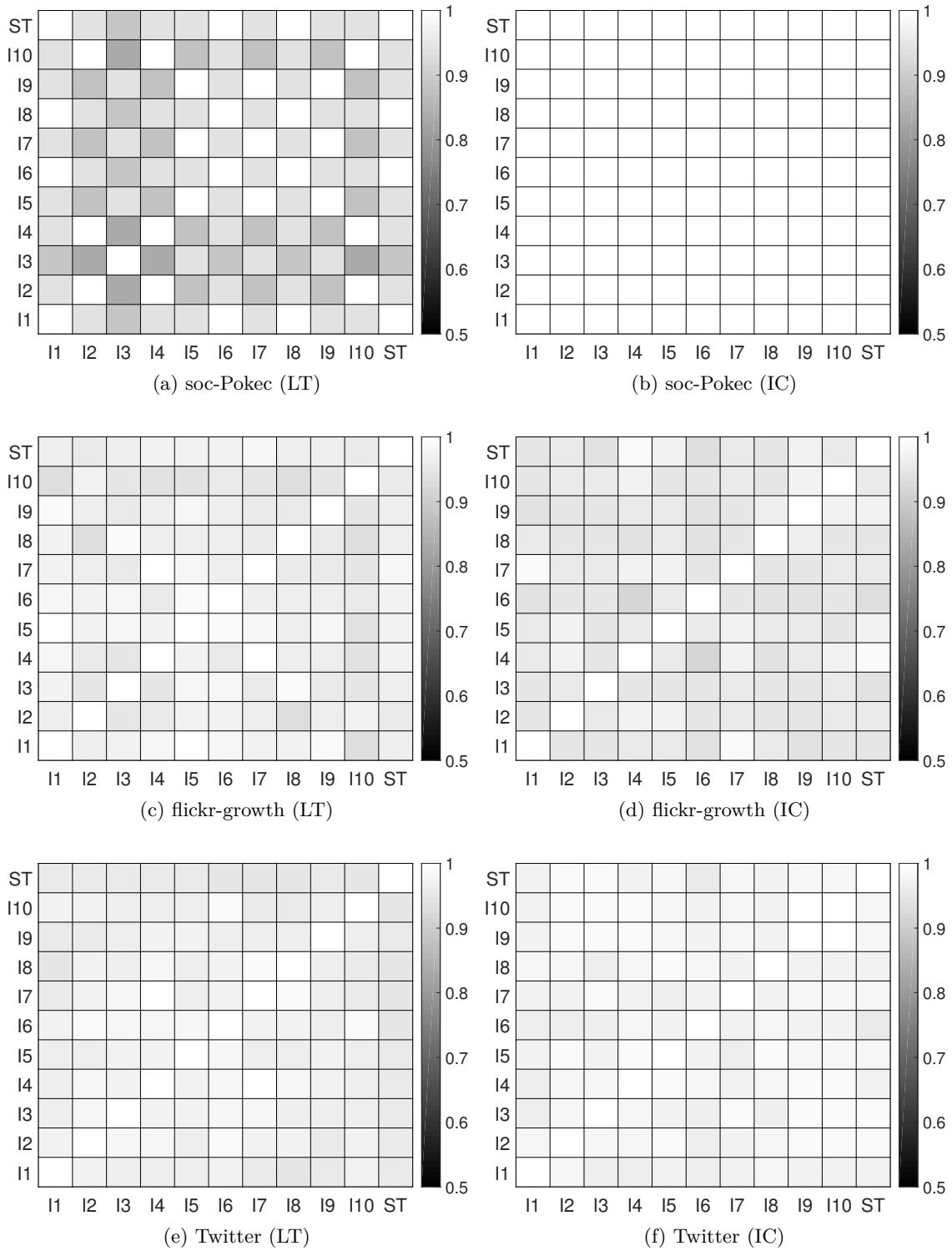
For the LT model, our algorithm scales up roughly linearly. For the IC model, the running time increases more than linear. This is due to our experimental settings. For the LT model, the sum of propagation probabilities from all in-neighbors of a vertex is always 1, while in the IC model, at the beginning the sum of propagation probabilities from all in-neighbors is roughly 0.9 but becomes 1 finally. Thus, the spreads of vertices change more dramatically in the IC model than in the LT model.

3.8.3 Top-K Task

We report the experimental results of the task of tracking top-k influential vertices.

Effectiveness

We still use the two small datasets, wiki-Vote and Flixster to verify the provable quality guarantees of our algorithms. Table 3.4 reports the recall of the sets of influential vertices



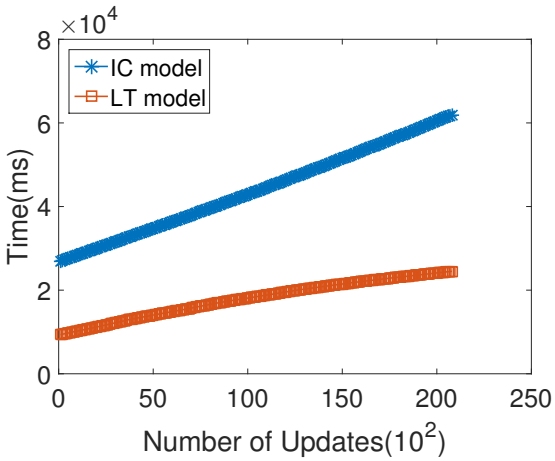
(a) soc-Pokec (LT) (b) soc-Pokec (IC)

(c) flickr-growth (LT) (d) flickr-growth (IC)

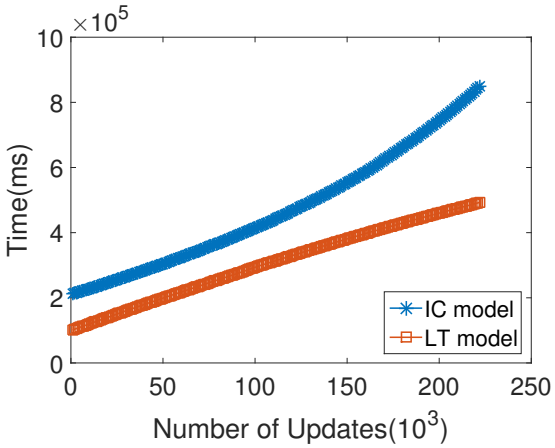
(e) Twitter (LT) (f) Twitter (IC)

Figure 3.4: Similarity among results in different instances. (Thresholding Task)

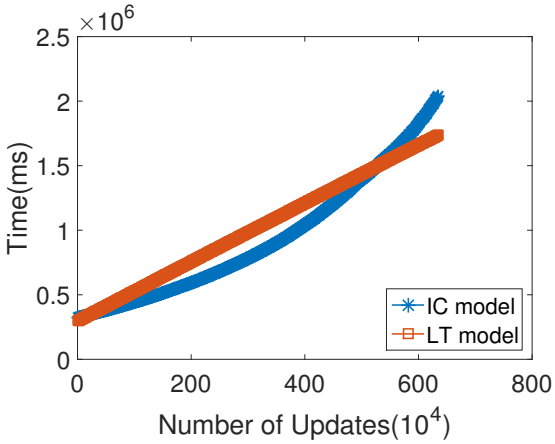
returned by our algorithms and the maximum errors of the false positive vertices in ab-



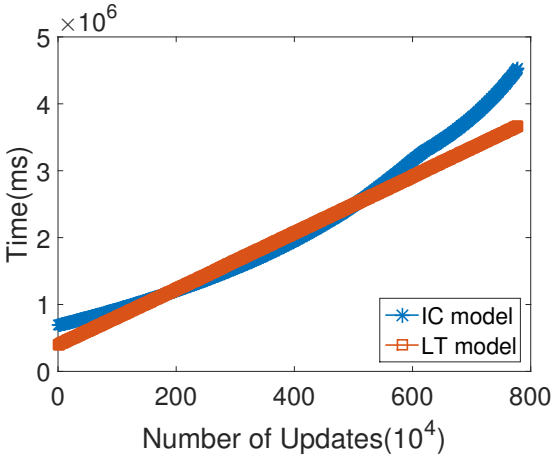
(a) wiki-Vote



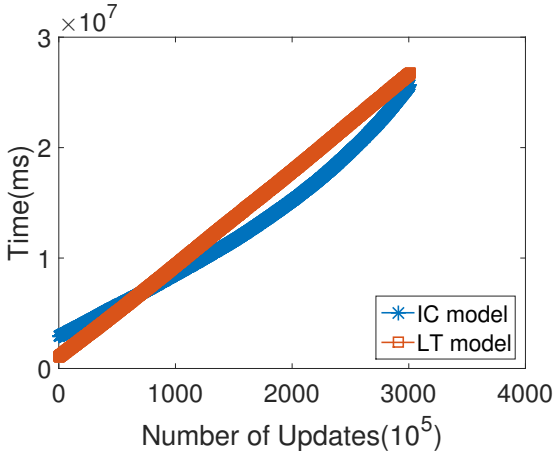
(b) Flixster



(c) soc-Pokec



(d) flickr-growth



(e) Twitter

Figure 3.5: Scalability. (Threshold)

Table 3.4: Recall and Maximum Error of the Top-k Task. "AE" denotes the method controlling absolute errors and "RE" denotes the method controlling relative errors. "w.h.p." is short for "with high probability".

	wiki-Vote		
	Theoretical Value (w.h.p.)	Ave.±SD (LT)	Ave.±SD (IC)
Recall (AE)	100%	100%±0	100%±0
Max. Error (AE)	$0.0005 * 7115 = 3.558$	1.854±0.080	1.872±0.090
Recall (RE)	100%	100%±0	100%±0
Max. Error Rate (RE)	36.5%	18.4%±0.97%	18.5%±1.17%
	Flixster		
	Theoretical Value (w.h.p.)	Ave.±SD (LT)	Ave.±SD (IC)
Recall (AE)	100%	100%±0	100%±0
Max. Error (AE)	$0.0005 * 99053 = 49.53$	27.77±0.87	27.17±0.56
Recall (RE)	100%	100%±0	100%±0
Max. Error Rate (RE)	36.5%	20.2%±0.90%	19.9%±0.51%

solute influence value. Again, our methods achieved 100% recall every time as guaranteed theoretically, and the real errors in influence were substantially smaller than the maximum error bound provided by our theoretical analysis.

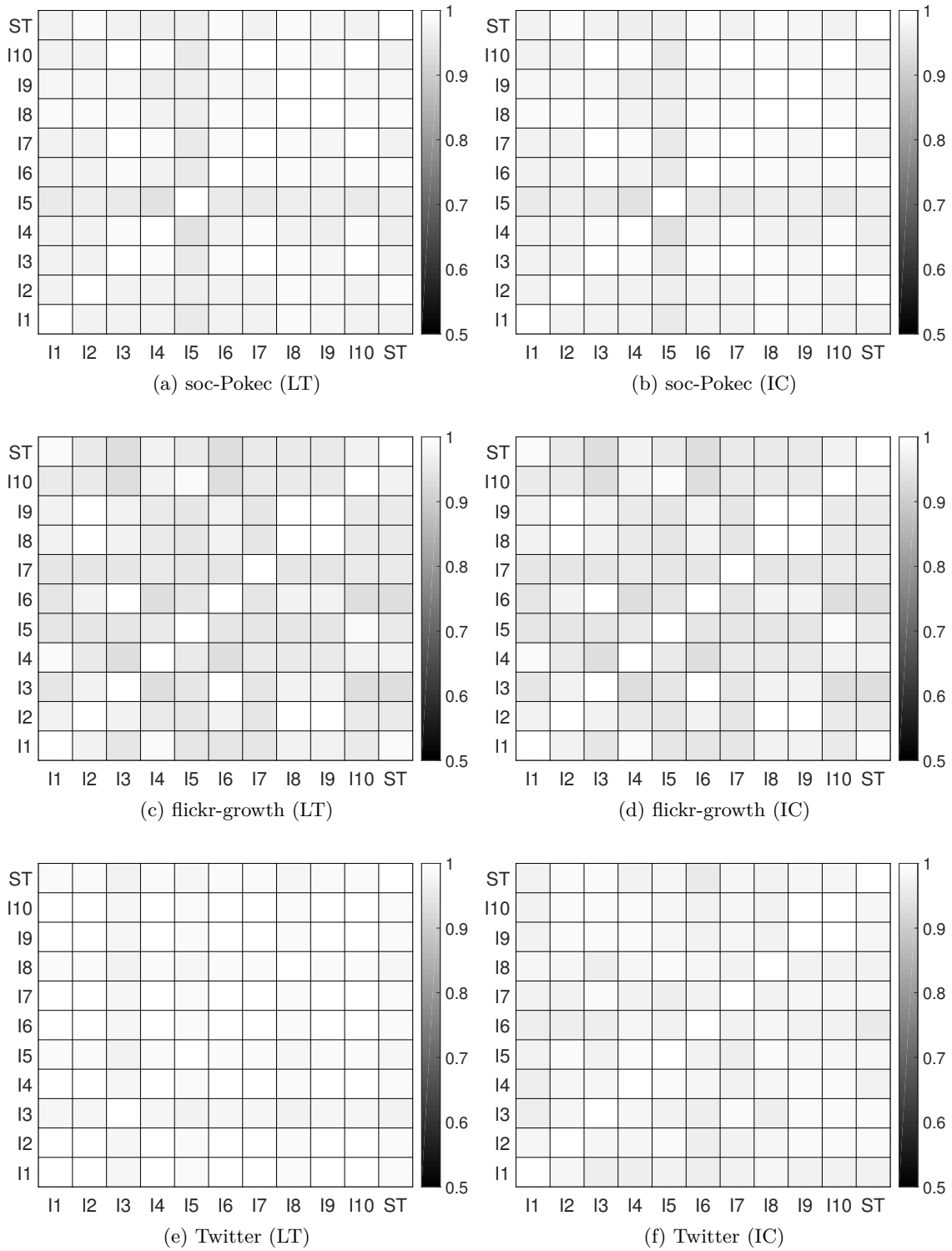
We also use the three large datasets to test Jaccard similarities between results obtained by running our algorithms on different instances. Fig. 3.6 and Fig. 3.7 show the results. Again, the outcomes from different instances are very similar, because the minimum similarity in all case is above 90%.

Efficiency

In Fig. 3.8, we report the average processing time of our algorithms with respect to the number of edge updates, where the average is taken over the results of the 10 instances. "RE IC" (RE is short for relative error) is our method in Section 3.6 under the IC model, while "AE IC" (AE is short for absolute error) stands for the method in Section 3.5 under the IC model.

In all cases, our methods controlling relative errors handles the whole update stream in substantially shorter time than our methods controlling absolute errors. Our algorithms under the LT model scales up roughly linearly. Under the IC model the running time sometimes increases more than linear. Still, this is caused by our experimental settings of the LT model and the IC model as illustrated in Section 3.8.2.

We also demonstrate the limitations of controlling $\max_{u \in S} I^k - I_u$ as an absolute error, where S is the set of vertices mined by Algorithm 5. Fig. 3.9 shows how the value I^k varies over time, and so does the theoretically maximum error $\max_{u \in S} \frac{I^k - I_u}{n}$ over time. The value of $\frac{I^k}{n}$ is estimated by $\frac{\mathcal{D}_1^k}{|\mathcal{R}_1|}$.



(a) soc-Pokec (LT)

(b) soc-Pokec (IC)

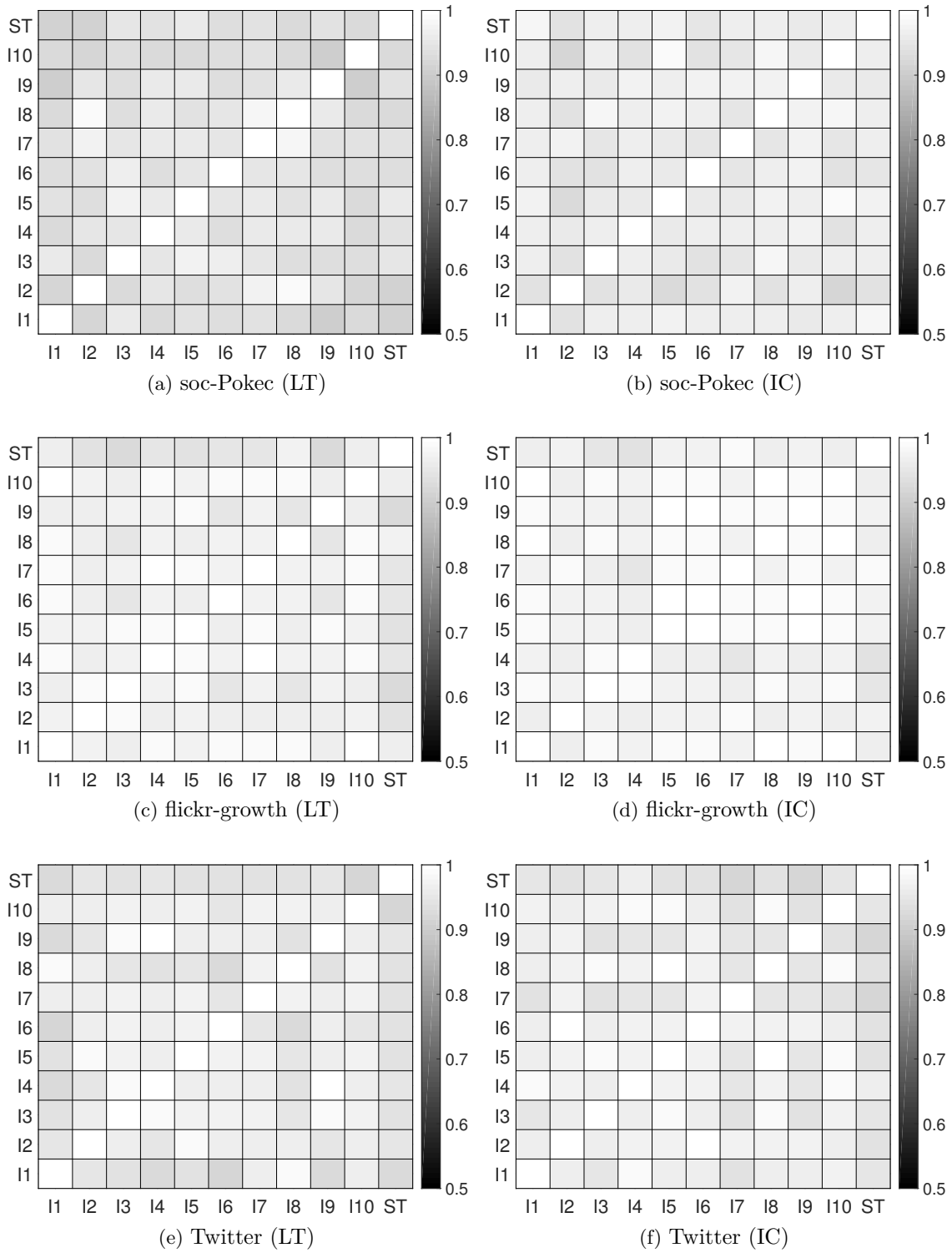
(c) flickr-growth (LT)

(d) flickr-growth (IC)

(e) Twitter (LT)

(f) Twitter (IC)

Figure 3.6: Similarity among results in different instances. (Top-k Task with Absolute Errors)

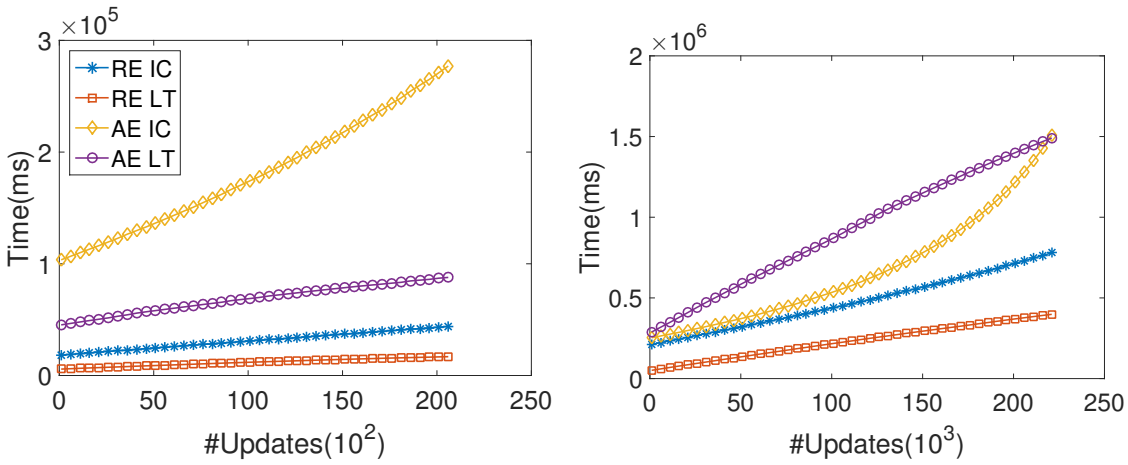


(a) soc-Pokec (LT) (b) soc-Pokec (IC)

(c) flickr-growth (LT) (d) flickr-growth (IC)

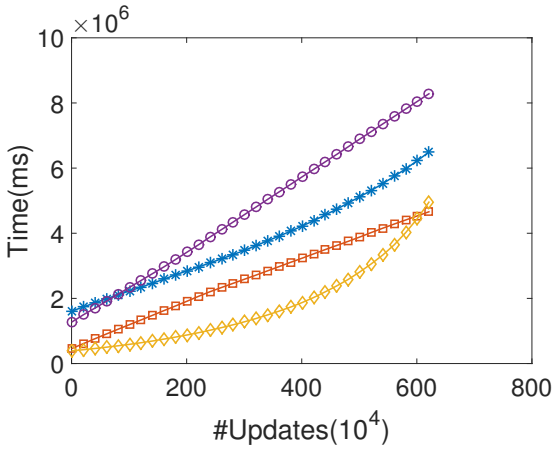
(e) Twitter (LT) (f) Twitter (IC)

Figure 3.7: Similarity among results in different instances. (Top-k Task with Relative Errors)

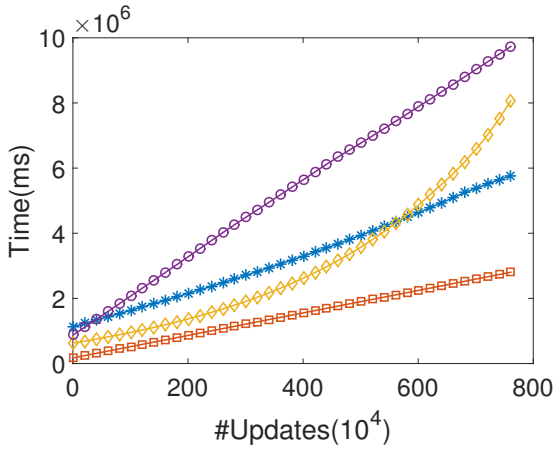


(a) wiki-Vote

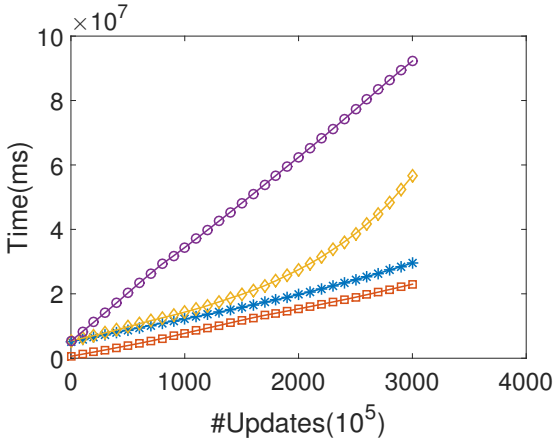
(b) Flixster



(c) soc-Pokec



(d) flickr-growth



(e) Twitter

Figure 3.8: Scalability (Tracking Top-k Influential Vertices).

The maximum error of the algorithm RE is either only a little bigger or smaller than the maximum error of the algorithm AE. Moreover, we find that the value of $\frac{I^k}{n}$ varies over

Table 3.5: Running time (s) on static networks.

Dataset	#Updates	LT		IC	
		Total	ST	Total	ST
wiki-Vote	2.1×10^4	11.3	2.3	29.3	8.2
Flixster	2.2×10^5	266	28	522	85
soc-Pokec	6.4×10^6	3165	311	4461	735
flickr-growth	7.8×10^6	1908	201	3223	935
Twitter	3.0×10^8	15369	375	19803	4770

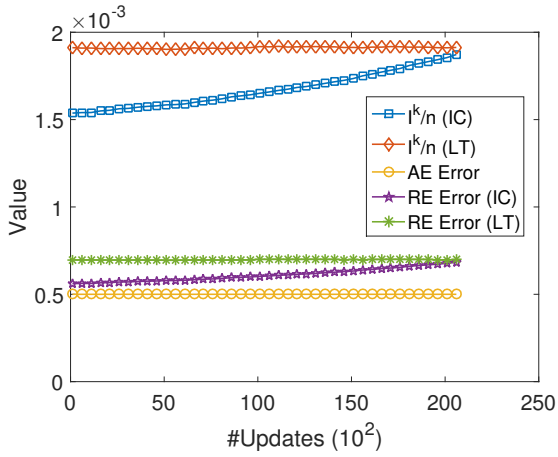
time, especially under the IC model. In Fig. 3.9 (c), (d) and (e), sometimes the error of the algorithm AE is even greater than $\frac{I^k}{n}$, which makes the result meaningless because all vertices are returned as influential vertices. This demonstrates the advantage of our method controlling relative errors over our method controlling absolute errors.

3.8.4 Comparison with Simple Heuristics and Static Algorithms

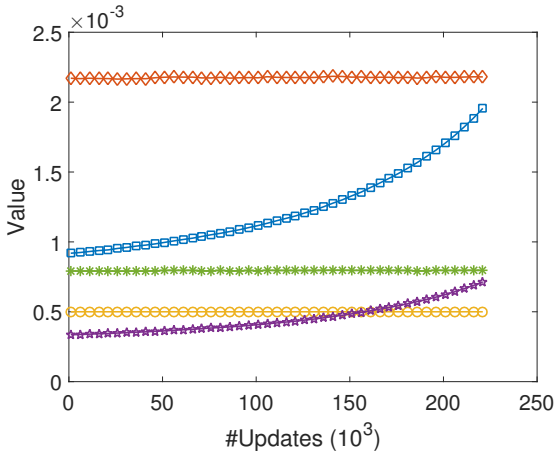
One may wonder if we can apply simple heuristics to extract influential vertices in a social network. Thus, we adopted two simple heuristics, degree and PageRank, as baselines to compared to our algorithms w.r.t. effectiveness. These two simple heuristics just use vertices' degrees in the social network and PageRank values as proxies of influence spread. Note that these two heuristics cannot solve the threshold based influential vertices mining problem because they do not know the real or the approximate influence spread of each vertex.

To compare our algorithms with degree and PageRank heuristics, we report the recall of the top ranked vertices obtained by each method on wiki-Vote and Flixster data sets in Fig 3.10. Vertices ranking by 2,000,000 times Monte Carlo simulations is regarded as the (pseudo) ground truth. The measure $Recall@N$ is calculated by $\frac{TP_N}{N}$, where TP_N is the number of vertices ranked top-N by both our algorithms and the ground truth. The results show that the rankings of the top vertices generated by our algorithms constantly have very good quality, while the two heuristics sometimes perform well but sometimes return really poor rankings. Moreover, performance of a heuristic algorithm is not predictable.

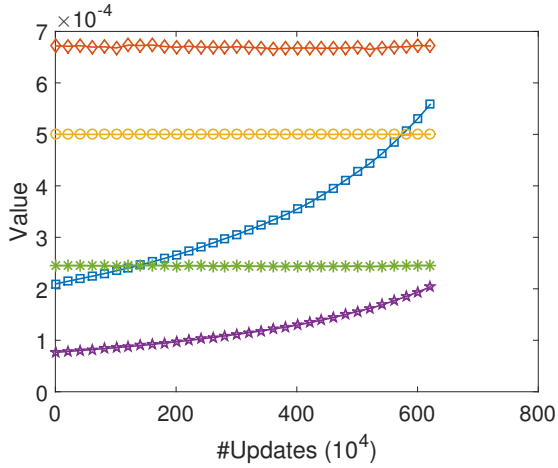
By conducting the task of tracking top-k influential vertices with relative errors, we test running time of the static algorithm (ST) that runs our sampling algorithm on every snapshots of a dynamic network. Table 3.5 compares the running time of ST on the last snapshot, and the total time (denoted by "Total") that our algorithm generates a polling sketch on the base network and deals with the whole update stream. In Table 3.5, the running time of Total is at most 40 times longer than that of ST. Thus, if we re-generate a polling sketch from scratch every time when the network updates, we probably can only deal with tens of updates within the same time as Total spends on all updates. However, the number of total updates is huge, tens of thousands or even hundreds of millions. This



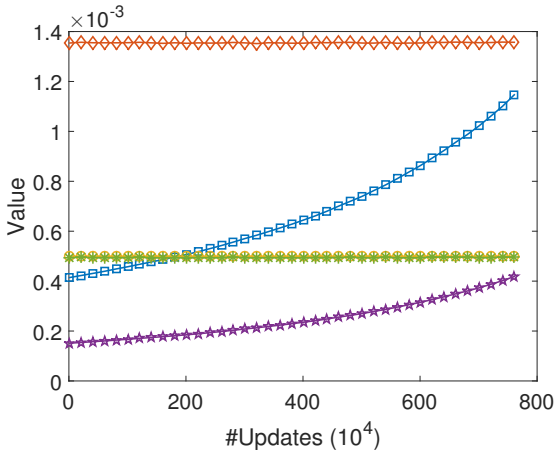
(a) wiki-Vote



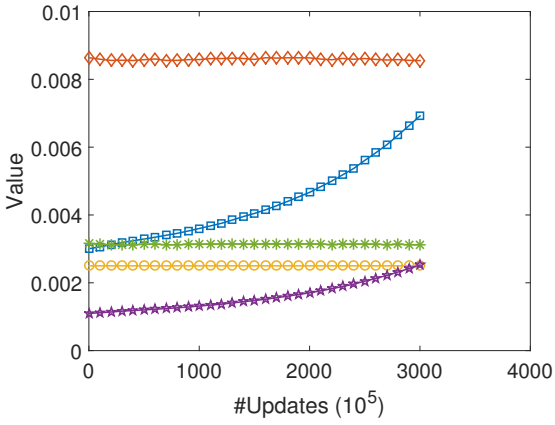
(b) Flixster



(c) soc-Pokec

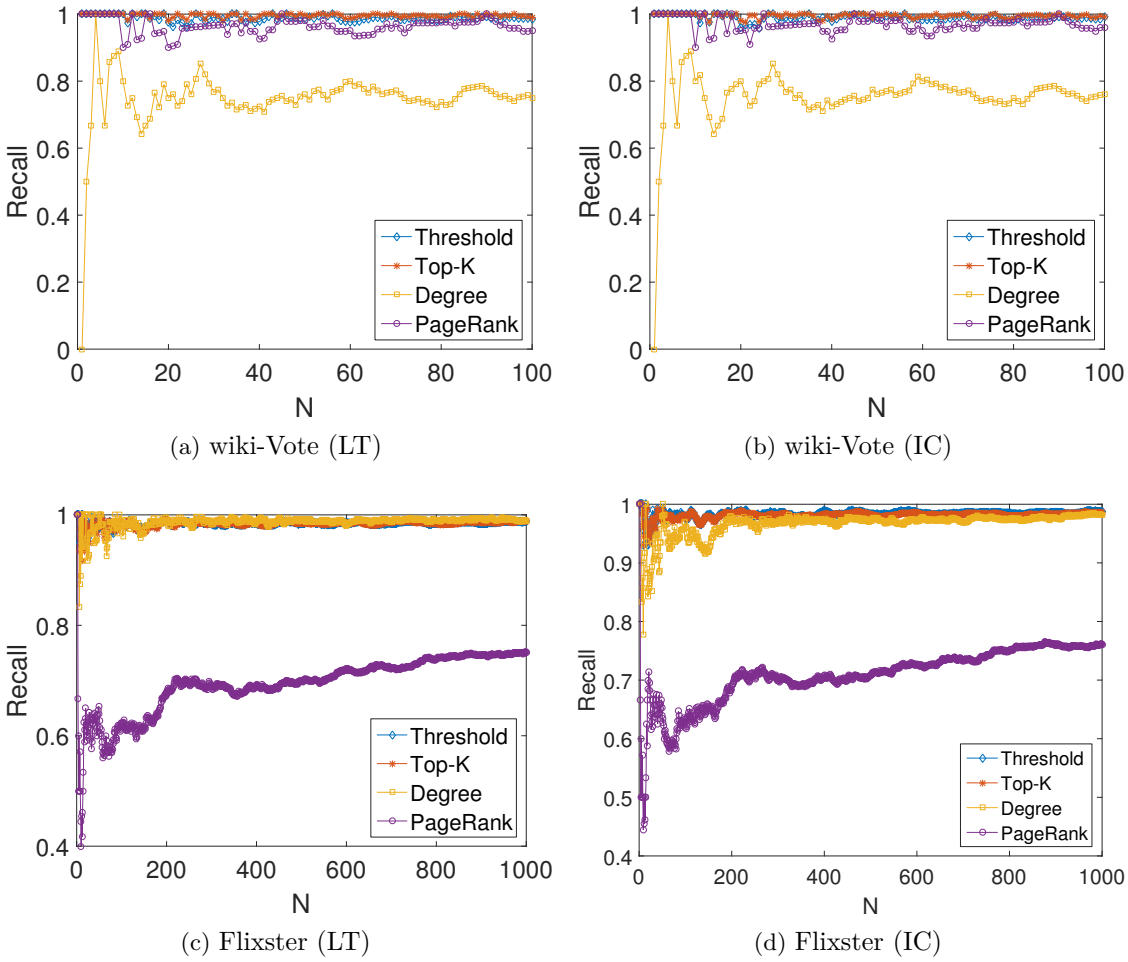


(d) flickr-growth



(e) Twitter

Figure 3.9: Average $\frac{I^k}{n}$ over time.



(a) wiki-Vote (LT)

(b) wiki-Vote (IC)

(c) Flixster (LT)

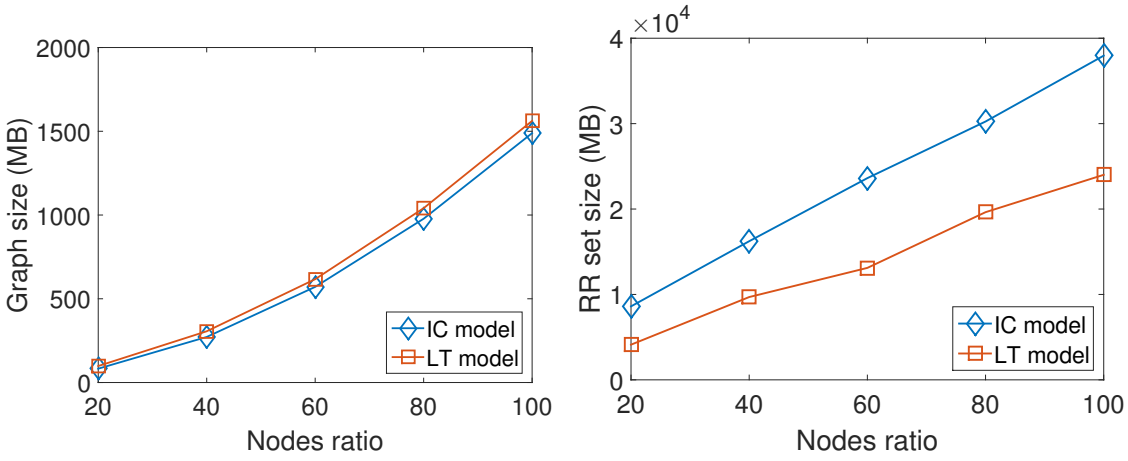
(d) Flixster (IC)

Figure 3.10: *Recall@N*

indicates that the non-incremental algorithm (re-generate a polling sketch from scratch when the network updates) is not competitive at all.

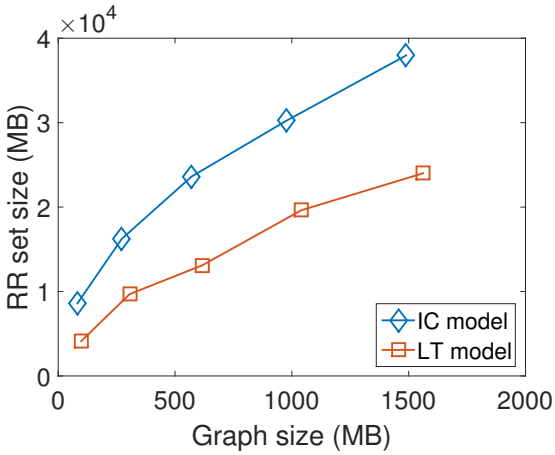
3.8.5 Memory Usage with respect to Input Size

We also report the memory usage of our algorithm against the increase of the input graph size. Since the task of mining top-k influential vertices with absolute errors usually requires the most memory usage, we only report results of this task. We used the second largest data set, flickr-growth network, to generate some smaller networks. Specifically, we sampled 20%, 40%, 60% and 80% vertices and extract the induced subgraphs. For each sample rate, we sampled 10 subgraphs and for each subgraph we generated a base network and an update stream as we described in Section 3.8.1. We ran the top-k influential vertices mining algorithm on those generated data. Fig. 3.11 reports the average memory storing the input graph and the average peak memory usage of the poll samples against the sample rate. The results show that the size of sampled graph increases super-linearly while the memory of



(a) Graph Size

(b) Polling SKetch Size



(c) Poll Samples w.r.t Graph

Figure 3.11: Memory Usage

poll samples increases roughly linearly as the sample rate increases. Fig. 3.11 also shows that the average peak memory used by the poll samples increases sub-linearly as the input graph size increases.

3.9 Conclusions

In this chapter, we proposed novel, effective and efficient polling-based algorithms for tracking influential individual vertices in dynamic networks under the Linear Threshold model and the Independent Cascade model. We modeled dynamics in a network as a stream of edge weight updates. We devised an efficient incremental algorithm for updating poll samples against network changes. For two interesting settings of influential vertices tracking, namely, tracking vertices with influence above a given threshold and tracking top-k influential vertices, we derived the number of poll samples we need to approximate the exact

set of influential vertices. We reported a series of experiments on five real networks and demonstrated the effectiveness and efficiency of our algorithms.

There are a few interesting directions for future work. For example, can we apply similar techniques to other influence models such as the Continuous-Time Diffusion Model [35]? Since the Continuous-Time Diffusion model has an implicit time constraint, how to efficiently update poll samples according to the time constraint is a critical challenge. Parallelizing our methods in large distributed systems is also an interesting future direction. Moreover, how to deal with influence maximization queries in a dynamic social network is worth exploring, since the theoretically sound method [84] is not practical due to the large number of poll samples it needs.

Chapter 4

Budget Allocation for Influence Maximization

Imagine we are introducing a new product through a social network, where we know for each user in the network the function of purchase probability with respect to discount. Then, what discount should we offer to those social network users so that, under a predefined budget, the adoption of the product is maximized in expectation? Although influence maximization has been extensively explored, surprisingly, this appealing practical problem still cannot be answered by the existing influence maximization methods. In this chapter, we tackle the problem systematically. We formulate the general continuous influence maximization problem, investigate the essential properties, and develop a general coordinate descent algorithmic framework as well as the engineering techniques for practical implementation. Our investigation does not assume any specific propagation model and thus is general and principled. At the same time, using the most popularly adopted triggering models as a concrete example, we demonstrate that more efficient methods are feasible under specific propagation models. Our extensive empirical study on four benchmark real world networks with synthesized purchase probability curves clearly illustrates that continuous influence maximization can improve influence spread significantly with very moderate extra running time comparing to the classical influence maximization methods.

4.1 Introduction

Influence maximization [33, 54] is a critical technique in many social network applications, such as viral marketing. The intuition is that, by targeting on only a small number of vertices (called seed vertices), it is possible to trigger a large cascade of information spreading in a social network. Technically, in a social network, influence maximization tries to identify a set of vertices such that if the selected vertices are committed to spread a piece of information to their neighbors, such as adopting a product, the expected spread in the social network

is maximized. There have been abundant studies on various models and computational methods for influence maximization, which are reviewed in Chapter 2.

Imagine a company is introducing a new product through a social network by providing discounts to users in the network in the hope of maximizing the influence spread. The total discount is constrained by a budget defined by the company. It is well known that different users in a social network may have a different capability in spreading influence. Consequently, the company naturally wants to offer different users different discounts. It is reasonable to assume that the more discount a user is offered, the more likely the user may adopt the product and spread the influence to her neighbors, which is also known in marketing research as the *purchase probability curve* being monotonic with respect to discount [58]. At the same time, different users may have different purchase probability curves. Given a budget and the users' purchase probability curves, what discounts should the company offer to the users so that the expected influence spread is maximized? Apparently, this is an interesting question that is asked again and again in various applications where influence maximization is used. At the same time, unfortunately the existing influence maximization techniques cannot answer the question.

Motivated by the practical demands, we investigate the questions about what discounts we should offer to social network users. In general, given a social network, a budget, and, for each user in the network, the seed probability function on discount (corresponding to the purchase probability curve with respect to discount in the above motivation example), the continuous influence maximization problem is to find the optimal configuration, which consists of a discount rate for each user, that maximizes the influence spread in expectation. We make several contributions in this chapter.

First, to the best of our knowledge, we are the first to systematically study the problem of continuous influence maximization that utilizes users' purchase probability curves, which has significant applications in practice. We show that the continuous influence maximization problem is a generalization of influence maximization, which focuses on discrete configurations. Consequently, we investigate the hardness of the problem, and analyze several essential properties of the problem. We do not assume any specific propagation model, and thus all properties explored are general.

Second, we develop a general coordinate descent framework for the general continuous influence maximization problem. Again, this algorithm does not assume any specific propagation model. Such a coordinate descent algorithmic framework helps us prove some interesting connections of the CIM problem to the traditional influence maximization problem.

Third, we devise practically efficient implementations of our CIM algorithm for specific propagation models. We consider triggering models [54], which contain most popularly used propagation models like the Independent Cascade model and the linear threshold model in literature. We make an analogy of polling-based influence maximization algorithm [10]

and machine learning, and develop effective algorithms also based on polling that avoid the "overfitting" issue.

Last, we report an extensive empirical evaluation using four benchmark real social network data sets with synthesized purchase probability curves. The largest data set has almost 4 million vertices and 70 million edges. The experiment results clearly show that continuous influence maximization can significantly improve influence spread. At the same time, the extra running time remains moderate.

The rest of the chapter is organized as follows. We formulate the problem of continuous influence maximization in Section 4.2. In Section 4.3, we investigate the properties of the expectation of influence spread. We present the general coordinate descent framework and discuss how to find a good initial configuration for running the coordinate descent framework in Section 4.4. We study in Section 4.5 the relationship between continuous influence maximization developed in this chapter and the existing influence maximization problem. In Section 4.6, we develop algorithms under the triggering models and carefully analyze the "overfitting" issue and how to avoid it. We report an extensive empirical evaluation in Section 4.7, and conclude the chapter in Section 4.8.

4.2 Problem Definition

A social network is a graph $G = \langle V, E \rangle$, where V is a set of users and E is a set of relationships between users. Denote by $n = |V|$ the number of users and $m = |E|$ the number of relationships, that is, edges.

An *influence propagation model* (*propagation model* for short) describes the process of how influence is cascaded in a social network. Two most widely used propagation models are the independent cascade model and the linear threshold model [54]. In an influence cascade process, a cascade is started by a small number of users, whom we call *seed users* (or *seeds* for short). We call the set of seed users the *seed set*, denoted by S. Every propagation model has an *influence function* $I : 2^V \to R$, where $I(S)$ is the expected size of the cascade triggered by the seed set S and is also called the *influence spread* of S. Usually, $I(S)$ is assumed monotonic and submodular [54, 73], which capture the intuition about influence spreading.

In this chapter, we are interested in customizing a discount for every user in a social network to maximize influence cascading. With a discount of 0%, a user has to pay the full price. With a discount of 100%, the product is free for the user. Please note that the notion of discount here can also be used to model in general the cost that we would like to pay to a user to turn the user into a seed.

Technically, a user $u \in V$ is associated with a *seed probability function* $p_u : [0, 1] \to [0, 1]$, which models the probabilistic distribution that u is attracted to become a seed user given a discount between 0% to 100%. Denote by c_u the discount we offer to u. Then, $p_u(c_u)$ is the

probability that u becomes a seed user given such a discount. In this chapter, we assume that a seed probability function $p_u(\cdot)$ has the following properties: (1) $p_u(0) = 0$; (2) $p_u(1) = 1$; (3) $p_u(c_u)$ is monotonic with respect to c_u; and (4) $p_u(c_u)$ is continuously differentiable. Conditions (1) and (2) are also assumed in the classical influence maximization problem.

The existing marketing research [12, 98] estimates user purchase probability, where the focus is on the adoption rate of the whole population rather than each individual, and estimations are on specific goods. In reality, a user's purchasing behavior may change over time and on different types of goods [24]. Thus, the best way to decide a user's seed probability function (purchase probability curve) is to learn from data. Since seed probability functions can take many different forms, it is crucial to design a general marketing method that can handle all kinds of such functions.

We assume that different users become seed users independently. Denote by an n-dimensional vector $C = (c_1, ..., c_n)$ a *configuration* of discounts assigned to all users in G. It is clear that, unlike the situation in the influence maximization problem, the seed set S in our problem setting is probabilistic. Given a social network $G = \langle V, E\rangle$ and a configuration C, the probability that a subset $S \subseteq V$ of users is the seed set is

$$Pr(S; V, C) = \prod_{u \in S} p_u(c_u) \prod_{v \in V \setminus S} (1 - p_v(c_v)) \tag{4.1}$$

For a specific propagation model with an influence function $I(S)$, the *expected influence spread* is

$$UI(C) = \sum_{S \in 2^V} Pr(S; V, C) I(S) \tag{4.2}$$

Now we define the *continuous influence maximization* problem (*CIM* for short) studied in this chapter as follows. Given a social network $G = \langle V, E\rangle$, a budget B, a seed probability function $p_u(c_u)$ for every user u, and an propagation model with an influence function $I(S)$, find the configuration C that is the optimal solution to the following continuous optimization problem.

$$\begin{aligned} \text{maximize} \quad & UI(C) \\ \text{s.t.} \quad & 0 \leq c_u \leq 1, \forall u \in V \\ & \sum_{u \in V} c_u \leq B \end{aligned} \tag{4.3}$$

We call a configuration satisfying the constraints in Eq. 4.3 a *feasible* configuration. Please note that the budget B here is a safe budget in general. When discounts here are used to model the costs committed to users, the budget models the total cost. When discounts are explained as discount rates, the budget is the worst-case budget. We leave the exploration of other forms of the budget constraint to future work, such as the expected budget under the discount rate explanation.

Table 4.1: Frequently used notations.

Notation	Description
$G = \langle V, E \rangle$	Social network G, where V is the set of users and E is the set of relationships
$n = \|V\|$	The number of vertices in G
$m = \|E\|$	The number of edges in G
$p_u(c_u)$	The probability that u becomes a seed user if u is offered a discount c_u
$C = (c_1, c_2, ..., c_n)$	A configuration, where c_u $(0 \leq c_u \leq 1)$ is the discount offered to user u
$Pr(S; V, C)$	The probability of a subset of users $S \subseteq V$ is the seed set S (Eq. 4.1)
$UI(C)$	The expected influence spread caused by configuration C (Eq. 4.2)
$H = \{h_1, h_2..., h_{m_H}\}$	A polling sketch which contains m_H poll samples

The classical influence maximization problem is a special case of the continuous influence maximization problem, since it can be written in a similar way as follows.

$$\begin{aligned} \text{maximize} \quad & UI(C) \\ \text{s.t.} \quad & c_u = 0 \text{ or } c_u = 1, \forall u \in V \\ & \sum_{u \in V} c_u \leq B \end{aligned} \tag{4.4}$$

We call a configuration satisfying the constraints in Eq. 4.4 an *integer* configuration. Apparently, an integer configuration is also a feasible configuration.

In the rest of the chapter, for the sake of clarity, we also call the classical influence maximization problem *discrete influence maximization* (*DIM* for short). Table 4.1 summarizes the frequently used notations.

4.3 Expected Influence Spread

The CIM problem is to optimize the expected influence spread $UI(C)$. In this section, we discuss the computation of $UI(C)$ and the monotonicity of $UI(C)$, which prepare us for the solution development in the next sections.

4.3.1 Computing $UI(C)$

Given $G = \langle V, E \rangle$, an influence function $I(S)$, and a seed probability function $p_u(c_u)$ for every $u \in V$, how can we obtain $UI(C)$? It is known that, for many popular propagation models, computing $I(S)$ is #P-hard [19, 21]. What is the hardness of computing $UI(C)$?

Theorem 8 (Complexity)**.** *Given a configuration C, the problem of computing $UI(C) = \sum_{S\in 2^V} Pr(S;V,C)I(S)$ is #P-hard if computing $I(S)$ is #P-hard.*

Proof. We prove by a simple reduction from computing $I(S)$. For any S, we can make a configuration C such that $c_u = 1$ if $u \in S$ and $c_u = 0$ otherwise. Clearly we have $UI(C) = I(S)$. Thus, if computing $I(S)$ is #P-hard, so is computing $UI(C)$. □

Since $UI(C)$ is the expectation of $I(S)$ over the random variable S, we can use Monte Carlo simulations to estimate $UI(C)$. Because every user becomes a seed user independently, randomly generating a seed set S based on $Pr(S;V,C)$ is equivalent to simply adding each user u to S independently with probability $p_u(c_u)$. We have the following result.

Theorem 9 ((ϵ,δ) estimation)**.** *Suppose we have an influence spread oracle that can return the influence spread $I(S)$ of a given seed set S. By calling the influence spread oracle $\frac{n^2 \ln \frac{2}{\delta}}{2\epsilon^2(\sum_{u\in V} p_c(c_u))^2}$ times, we can have an (ϵ,δ) estimation [72] of $UI(C)$.*

Proof. Recall that $UI(C) = \sum_{S\in 2^V} Pr(S;V,C)I(S)$. $\forall S \in 2^V$, we have $0 \le I(S) \le n$. We can estimate $UI(C)$ by a Monte Carlo method. By applying the Hoeffding bound, we have

$$Pr(|\widehat{UI}(C) - UI(C)| \ge \epsilon UI(C)) \le 2e^{-\frac{2R^2\epsilon^2 UI^2(C)}{Rn^2}},$$

where R is the number of MC simulations. Since $UI(C) \ge \sum_{u\in V} p_u(c_u)$, to achieve the goal that $Pr(|\widehat{UI}(C) - UI(C)| \ge \epsilon UI(C)) \le \delta$, we can set $R \ge \frac{n^2 \ln \frac{2}{\delta}}{2\epsilon^2(\sum_{u\in V} p_u(c_u))^2}$. □

As mentioned before, computing $I(S)$ is #P-hard for some influence functions. The good news is that there exists a FPRAS[1] [72, 68] for estimating $I(S)$. We prove that if $I(S)$ can be estimated efficiently, so is $UI(C)$. Similar to influence maximization where the number of seeds B is assumed to be $\Omega(1)$, we assume that the expected number of seeds $\sum_{u\in V} p_u(c_u)$ is also $\Omega(1)$. Moreover, as we will analyze in Corollary 13.3, if the expected number of seeds $\sum_{u\in V} p_u(c_u)$ is too small, such a configuration cannot make a high influence and we are not interested in it if we try to maximize $UI(C)$.

Theorem 10 (FPRAS estimation)**.** *For an influence function $I(\cdot)$, if there is a FPRAS for estimating $I(\cdot)$, there is a FPRAS for estimating $UI(\cdot)$.*

Proof. If there is a FPRAS for estimating $I(S)$, in $O(\text{POLY}(\frac{n}{\epsilon'}\ln\frac{1}{\delta'}))$ time we can obtain an (ϵ',δ') approximation of $I(S)$. If we set $\delta' = \frac{\delta_1}{O(\text{POLY}(\frac{n}{\epsilon'}\ln\frac{1}{\delta_1}))}$, the time we need is $O(\text{POLY}(\frac{n}{\epsilon'}\ln\frac{O(\text{POLY}(\frac{n}{\epsilon'}\ln\frac{1}{\delta_1}))}{\delta_1})) = O(\text{POLY}(\frac{n}{\epsilon'}\ln\frac{1}{\delta_1}))$.

[1] An FPRAS (Fully Polynomial Randomized Approximation Scheme) is an algorithm which returns an (ϵ,δ) estimation of the desired value in time polynomial to n (size of input), $\frac{1}{\epsilon}$ and $\ln\frac{1}{\delta}$.

According to Theorem 9, if we can access the value of $I(S)$, we can have an (ϵ', δ_1) approximation of $UI(C)$ by calling the oracle $O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1}))$ times. Suppose $R = O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1}))$, and we have R randomly generated seed sets $\{S_1, ..., S_R\}$. Let $\widehat{UI}(C) = \sum_{i=1}^{R} I(S_i)$, $\widehat{I}(S_i)$ be the estimation of $I(S_i)$ obtained by the FPRAS in $O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1}))$ time, and $\widehat{\widehat{UI}}(C) = \sum_{i=1}^{R} \widehat{I}(S_i)$. Clearly, $Pr\{|\widehat{UI}(C) - UI(C)| > \epsilon' UI(C)\} < \delta_1$, and $Pr\{|\widehat{\widehat{UI}}(C) - \widehat{UI}(C)| > \epsilon' \widehat{UI}(C)\} < \delta_1$ (by Union Bound). Applying the union bound, we have that $Pr\{|\widehat{\widehat{UI}}(C) - UI(C)| < \epsilon'(2 + \epsilon') UI(C)\} > 1 - 2\delta_1$.

Setting $\epsilon'(2 + \epsilon') = \epsilon$ and $2\delta_1 = \delta$, which means $\epsilon' = \sqrt{1 + \epsilon} - 1$ and $\delta_1 = \frac{\delta}{2}$, we can obtain an (ϵ, δ) approximation of $UI(C)$ in $O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1})) \times O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1})) = O(\text{POLY}(\frac{n}{\epsilon'} \ln \frac{1}{\delta_1}))$ time. Note that $\frac{1}{\epsilon'} = \frac{1}{\sqrt{1+\epsilon}-1} = \frac{\sqrt{\epsilon+1}+1}{\epsilon} \leq \frac{3}{\epsilon} = O(\frac{1}{\epsilon})$, and $\ln \frac{1}{\delta_1} = O(\ln \frac{1}{\delta})$. So in $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ time we can have an (ϵ, δ) approximation of $UI(C)$. □

For the two most popular propagation models, namely the independent cascade model and the linear threshold model [54], computing $I(S)$ has FPRAS [55]. Thus, $UI(C)$ under these two models can be efficiently estimated.

Corollary 10.1. *Computing $UI(C)$ under both the Independent Cascade model and the Linear Threshold model admits a FPRAS, assuming $\sum_{u \in V} p_u(c_u) \in \Omega(1)$.*

In summary, the results in this subsection establish that, as long as $I(S)$ can be computed/estimated efficiently (that is, in polynomial time), $UI(C)$ can also be estimated efficiently. This strong relation makes comparing two different configurations C_1 and C_2 computationally feasible, since we can efficiently estimate $UI(C_1)$ and $UI(C_2)$ accurately.

4.3.2 Monotonicity of $UI(C)$

Eq. 4.3 contains an inequality constraint $\sum_{u \in V} c_u \leq B$. According to the assumption that $p_u(c_u)$ is monotonic with respect to c_u, we can show that the inequality constraint $\sum_{u \in V} c_u \leq B$ can be substituted by an equation constraint $\sum_{u \in V} c_u = B$.

Lemma 9. *Given configurations $C_1 = (c_1^1, ..., c_n^1)$ and $C_2 = (c_1^2, ..., c_n^2)$, if there exists u $(1 \leq u \leq n)$ such that $c_u^1 \geq c_u^2$, and $\forall v \in V \setminus \{u\}$, $c_v^1 = c_v^2$, then $UI(C_1) \geq UI(C_2)$.*

Proof. Because $p_u(c_u)$ is monotonic with respect to c_u, we have $p_u(c_u^1) \geq p_u(c_u^2)$. Thus, $p_u(c_u^1) - p_u(c_u^2) = \alpha \geq 0$. We have

$$\begin{aligned} UI(C) = &\sum_{S \in 2^{V \setminus \{u\}}} Pr(S; V \setminus \{u\}, C) I(S)[1 - p_u(c_u)] \; + \\ &\sum_{S \in 2^{V \setminus \{u\}}} Pr(S; V \setminus \{u\}, C) I(S \cup \{u\}) p_u(c_u) \end{aligned}$$

Therefore,

$$\begin{aligned}
&UI(C_1) - UI(C_2) \\
=&\sum_{S\in 2^{V\setminus\{u\}}} Pr(S; V\setminus\{u\}, C_1)I(S\cup\{u\})\alpha \quad - \sum_{S\in 2^{V\setminus\{u\}}} Pr(S; V\setminus\{u\}, C_2)I(S)\alpha \\
=&\sum_{S\in 2^{V\setminus\{u\}}} \alpha Pr(S; V\setminus\{u\}, C_2)[I(S\cup\{u\}) - I(S)]
\end{aligned}$$

Due to the monotonicity of $I(S)$, $I(S\cup\{u\}) - I(S) \geq 0$ and $UI(C_1) - UI(C_2) \geq 0$. Thus, we have $UI(C_1) \geq UI(C_2)$. □

For two configurations $C_1 = (c_1^1, ..., c_n^1)$ and $C_2 = (c_1^2, ..., c_n^2)$, we write $C_1 \succeq C_2$ if $\forall u$, $c_u^1 \geq c_u^2$. By the transitivity of $\geq$ and Lemma 9, we have the following immediately.

Theorem 11 (Monotonicity)**.** *If $C_1 \succeq C_2$, then $UI(C_1) \geq UI(C_2)$.*

According to Theorem 11, it is obvious that the optimal C for CIM (Eq. 4.3) must use up the budget B. Thus, CIM can be rewritten as follows.

$$\begin{aligned}
\text{maximize} \quad & UI(C) \\
\text{s.t.} \quad & 0 \leq c_u \leq 1, \forall u \in V \\
& \sum_{u\in V} c_u = B
\end{aligned} \tag{4.5}$$

4.4 A General Coordinate Descent Framework

In this section, we develop a coordinate descent algorithm to solve the continuous influence maximization problem. Our algorithm is general, since we do not compose any constraints on the specific form of the influence function $I(S)$ and the seed probability function $p_u(c_u)$. We only assume that $p_u(c_u)$ is monotonic and continuously differentiable.

4.4.1 Gradient

For a vertex $u \in V$, we can rewrite $UI(C)$ as follows.

$$\begin{aligned}
&UI(C) \\
=&\sum_{S\in 2^V \wedge u\in S} Pr(S; V, C)I(S) + \sum_{S\in 2^V \wedge u\notin S} Pr(S; V, C)I(S) \\
=&\sum_{S\in 2^{V\setminus\{u\}}} Pr(S; V\setminus\{u\}, C)I(S)[1 - p_u(c_u)] \\
&+ \sum_{S\in 2^{V\setminus\{u\}}} Pr(S; V\setminus\{u\}, C)I(S\cup\{u\})p_u(c_u) \\
=&p_u(c_u)\cdot \sum_{S\in 2^{V\setminus\{u\}}} Pr(S; V\setminus\{u\}, C)[I(S\cup\{u\}) - I(S)] + const
\end{aligned}$$

where *const* is a constant with respect to c_u. Given a graph $G = \langle V, E\rangle$ and influence function $I(S)$, for a vertex $u \in V$, $Pr(S; V \setminus \{u\}, C)$, $I(S)$ and $I(S \cup \{u\})$ are constants with respect to c_u. Therefore, using a vertex $u \in V$ we can rewrite the objective function $UI(C)$ into a linear function of $p_u(c_u)$, where c_u is the only variable.

Assuming that $p_u(c_u)$ is continuously differentiable, we can take the partial derivative of $UI(C)$ with respect to c_u, that is,

$$\frac{\partial UI(C)}{\partial c_u} = p_u^{'}(c_u) \sum_{S \in 2^{V \setminus \{u\}}} Pr(S; V \setminus \{u\}, C)[I(S \cup \{u\}) - I(S)] \tag{4.6}$$

In this way, we can compute the gradient of $UI(C)$ with respect to a specific configuration C. The gradient information will be used in the coordinate descent algorithm to be developed next.

4.4.2 A Coordinate Descent Algorithm

The coordinate descent algorithm is an iterative algorithm. In each iteration, we pick only two variables c_i and c_j, and fix the rest $n-2$ variables. We try to increase the value of the objective function $UI(C)$ by changing only the values of c_i and c_j.

As stated in Eq. 4.5, $\sum_{u \in V} c_u = B$. Thus, when we fix c_u for all $u \in V \setminus \{i, j\}$, $\sum_{u \in V \setminus \{i,j\}} c_u$ is a constant. Let $B' = B - \sum_{u \in V \setminus \{i,j\}} c_u = c_i + c_j$. In other words, $c_j = B' - c_i$. Combining the other constraints $0 \leq c_i \leq 1$ and $0 \leq c_j \leq 1$ in Eq. 4.5, we have an equivalent constraint $max(0, B' - 1) \leq c_i \leq min(1, B')$.

Thus, in each iteration, increasing $UI(C)$ can be achieved by solving the following optimization problem.

$$\begin{aligned} \text{maximize} \quad & UI(C) \text{ with respect to } c_i \text{ and } c_j = B' - c_i \\ \text{s.t.} \quad & max(0, B' - 1) \leq c_i \leq min(1, B') \end{aligned} \tag{4.7}$$

To solve the above optimization problem, we further rewrite $UI(C)$ by fixing c_u for all $u \in V \setminus \{i, j\}$ and setting $c_j = B' - c_i$. That is,

$$\begin{aligned} UI(C) = \sum_{S \in 2^{V \setminus \{i,j\}}} & Pr(S; V \setminus \{i, j\}, C)\Big\{ \\ & [1 - p_i(c_i)][1 - p_j(B' - c_i)]I(S) \\ & + [1 - p_i(c_i)]p_j(B' - c_i)I(S \cup \{j\}) \\ & + [1 - p_j(B' - c_i)]p_i(c_i)I(S \cup \{i\}) \\ & + p_i(c_i)p_j(B' - c_i)I(S \cup \{i, j\})\Big\} \end{aligned} \tag{4.8}$$

In Eq. 4.8, except for $p_i(c_i)$ and $p_j(B' - c_i)$, all terms can be regarded as constants. Therefore, we have a new form of $UI(C)$ with respect to c_i and $c_j = B' - c_i$ as follows.

$$\begin{aligned} UI(C) =&(A_1 + A_2 - A_3 - A_4)p_i(c_i)p_j(B' - c_i) \\ &+ (A_3 - A_1)p_i(c_i) + (A_4 - A_1)p_j(B' - c_i) + const, \end{aligned} \tag{4.9}$$

where

$$\begin{aligned} A_1 &= \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S) \\ A_2 &= \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{i,j\}) \\ A_3 &= \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{i\}) \\ A_4 &= \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{j\}) \end{aligned} \tag{4.10}$$

In Eq. 4.9, $UI(C)$ only has one variable c_i. We take the derivative of $UI(C)$, that is,

$$\begin{aligned} \frac{\mathrm{d}UI(C)}{\mathrm{d}c_i} =&(A_1 + A_2 - A_3 - A_4)[p_i^{'}(c_i)p_j(B' - c_i) \\ &- p_i(c_i)p_j^{'}(B' - c_i)] + (A_3 - A_1)p_i^{'}(c_i) \\ &- (A_4 - A_1)p_j^{'}(B' - c_i) \end{aligned} \tag{4.11}$$

Since $p_i(c_i)$ and $p_j(B'-c_i)$ are both continuously differentiable, the value $c_i \in [max(0, B'-1), min(B', 1)]$ that maximizes the objective function in Eq. 4.7 must be in one of the following three situations: (1) $c_i = max(0, B' - 1)$; (2) $c_i = min(B', 1)$; or (3) $c_i = x$, where $x \in (max(0, B' - 1), min(B', 1))$ and $\frac{\mathrm{d}UI(C)}{\mathrm{d}c_i}|_{c_i=x} = 0$. Thus, we only need to check these three types of points and set c_i to the one that results in the maximum value of $UI(C)$ with respect to c_i and $c_j = B' - c_i$.

Based on the above discussion, the pseudo-code of the coordinate descent algorithm for solving the continuous influence maximization problem is given in Algorithm 8.

We do not assume any specific seed probability function $p_u(c_u)$ and influence function $I(S)$. Thus, Algorithm 8 is a general framework for solving the continuous influence maximization problem.

In Line 3 of Algorithm 8, we do not specify which c_i and c_j should be picked. One heuristic that may help here is to use the partial derivative $\frac{\partial UI(C)}{\partial c_u}$ as an indicator. For example, we can pick a variable with a large partial derivative and another variable that has a small partial derivative.

The convergence of Algorithm 8 is guaranteed by the following observations. First, $UI(C) \leq n$, where n is the number of vertices in the social network. Second, after each

Algorithm 8 The Coordinate Descent Algorithm

Input: Budget B, social network G, seed probability function $p_u(c_u)$, $\forall u \in V$, and influence function (propagation model) $I(S)$

Output: Configuration C

1: Initialize C such that C satisfies constraints in Eq. 4.5
2: **while** not converge **do**
3: Pick two variables c_i and c_j
4: $B' \leftarrow c_i + c_j$
5: Find all $x \in (max(0, B' - 1), min(B', 1))$ that $\frac{\mathrm{d}UI(C)}{\mathrm{d}c_i}|_{c_i=x} = 0$
6: $c_i \leftarrow \mathrm{argmax}_{c_i \in \{x, max(0,B'-1), min(1,B')\}} UI(C)$
7: $c_j \leftarrow B' - c_i$
8: **end while**
9: **return** C

iteration in Algorithm 8, the updated configuration C ensures that the value of $UI(C)$ is at least as good as the previous one, that is, the value of $UI(C)$ is non-decreasing.

Algorithm 8 approaches a stationary configuration as the limit, which is a necessary condition for finding local optima [11]. Since the objective function $UI(C)$ is not necessarily convex or concave, even when the stationary point is a local optima, it may not be the global optima (note that for non-concave functions, even finding a local maxima is NP-hard). At the same time, because in each iteration the value of our objective cannot be decreased, when taking a configuration C and applying Algorithm 8, we can always find a configuration C' no worse than C.

4.4.3 Finding a Good Initial Configuration: Unified Discount Configuration

To run Algorithm 8 effectively, we need a good initial configuration. A practical engineering strategy to design discounts is to offer a unified discount to some users in a social network. That means, for each vertex u in G, c_u is either a predefined value c or 0. When c is fixed, finding the optimal configuration C is to find the optimal set of users to offer each of them discount c. Suppose we choose a set S of users to offer discounts, denote by $Pr(S'; S, c)$ the probability of generating a seed set S' when the unified discount is c, that is,

$$Pr(S'; S, c) = \prod_{u \in S'} p_u(c) \prod_{v \in S - S'} (1 - p_v(c)) \tag{4.12}$$

We define $UI(S; c)$ as the expected influence spread when we offer each user in S a unified discount c. That is,

$$UI(S; c) = \sum_{S' \in 2^S} Pr(S'; S, c) I(S') \tag{4.13}$$

We observe the following nice property of $UI(S;c)$ when c is fixed.

Theorem 12 (Monotonicity and submodularity)**.** *If $I(S)$ is monotonic and submodular with respect to S, then $UI(S;c)$ is also monotonic and submodular with respect to S.*

Proof. The monotonicity can be immediately proved using Theorem 11. Next, we show the submodularity of $UI(S;c)$.

Suppose we have two sets S_1 and S_2 such that $S_1 \cup \{v\} = S_2$. Let u be a vertex such that $u \notin S_2$. Then,

$$\begin{aligned}
&UI(S_1 \cup \{u\}; c) - UI(S_1; c) \\
&= \sum_{S \subseteq S_1} Pr(S; S_1, c)\Big(p_u(c) I(S \cup \{u\}) + (1 - p_u(c)) I(S)\Big) - \sum_{S \subseteq S_1} Pr(S; S_1, c) I(S) \\
&= p_u(c) \sum_{S \subseteq S_1} Pr(S; S_1, c)\Big(I(S \cup \{u\}) - I(S)\Big)
\end{aligned}$$

We also have

$$\begin{aligned}
&UI(S_2 \cup \{u\}; c) - UI(S_2; c) \\
&= \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\Big(p_v(c) p_u(c) I(S \cup \{u, v\}) + p_v(c)(1 - p_u(c)) I(S \cup \{v\}) + \\
&\qquad\qquad p_u(c)(1 - p_v(c)) I(S \cup \{u\}) + (1 - p_u(c))(1 - p_v(c)) I(S)\Big) - \\
&\quad \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\Big(p_v(c) I(S \cup \{v\}) + (1 - p_v(c)) I(S)\Big) \\
&= \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\Bigg(p_u(c) p_v(c)\Big(I(S \cup \{u, v\}) - I(S \cup \{v\})\Big) + \\
&\qquad\qquad p_u(c)(1 - p_v(c))\Big(I(S \cup \{u\}) - I(S)\Big)\Bigg)
\end{aligned}$$

Due to the submodularity of $I(S)$, $I(S \cup \{u, v\}) - I(S \cup \{v\}) \leq I(S \cup \{u\}) - I(S)$. Therefore,

$$\begin{aligned}
&UI(S_1 \cup \{u\}; c) - UI(S_1; c) \\
&\leq \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\Bigg(p_u(c) p_v(c)\Big(I(S \cup \{u\}) - I(S)\Big) + \\
&\qquad\qquad p_u(c)(1 - p_v(c))\Big(I(S \cup \{u\}) - I(S)\Big)\Bigg) \\
&= p_u(c) \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\Big(I(S \cup \{u\}) - I(S)\Big) \\
&= UI(S_1 \cup \{u\}; c) - UI(S_1; c)
\end{aligned}$$

That is, we prove

$$UI(S_1 \cup \{u\}; c) - UI(S_1; c) \geq UI(S_2 \cup \{u\}; c) - UI(S_2; c).$$

By a simple induction, we can show that, if $S \subseteq T$ and $u \notin T$, $UI(S \cup \{u\}; c) - UI(S; c) \geq UI(T \cup \{u\}; c) - UI(T; c)$, which means $UI(S; c)$ is submodular with respect to S. □

The monotonicity and submodularity of $UI(S; c)$ with respect to S imply that, when c is fixed, we can apply a greedy algorithm to find a set of users S_c to offer discounts which can cause influence spread at least $(1 - \frac{1}{e})$ times of the influence spread caused by the optimal set of users S_c^*. In such a case, when the propagation model and seed probability function for each user are given, some efficient influence maximization algorithms [65, 104, 103] can be applied here.

One important problem is what unified discount rate c should we use. Since we have to use up all budget due to the monotonicity of $UI(C)$, to find the optimal c, we only need to consider situations when $c = \frac{B}{\lceil B \rceil}, \frac{B}{\lceil B \rceil + 1}, \ldots, \frac{B}{n}$, which means we only need to try $O(n)$ different values of c. To make the process of searching the optimal c even faster, we only try τ different values in $\{\frac{B}{\lceil B \rceil}, \frac{B}{\lceil B \rceil + 1}, \ldots, \frac{B}{n}\}$, where τ is a predefined constant. Algorithm 9 shows the Unified Discount heuristic. The τ different values of c tested by Algorithm 9 distributed roughly evenly in the range $[\frac{B}{n}, \frac{B}{\lceil B \rceil}]$. The i-th possible value of c tested is around $\frac{i}{\tau}$. We usually set τ as a small constant like 20. In the following of this chapter, we will see that setting a small τ can help avoid the "overfitting" issue. Moreover, our experimental results will show that a small τ like 20 is enough to achieve a good performance in practice.

Algorithm 9 The Unified Discount Heuristic

Input: Budget B, social network G, seed probability function $p_u(c_u)$, $\forall u \in V$, influence function (propagation model) $I(S)$ and τ
Output: Configuration C

$C \leftarrow (0, 0, \ldots, 0)$
for $i \leftarrow 1$ to τ **do**
  $c' \leftarrow i/\tau$ and $k \leftarrow B/c$
  $c \leftarrow B/k$
  Apply the greedy algorithm to find S_c
  Make $C' = (c'_1, c'_2, \ldots, c'_n)$, where $c'_i = c$ if $i \in S_c$ and $c'_i = 0$ otherwise
  if $UI(C') > UI(C)$ **then**
    $C \leftarrow C'$
  end if
end for
return C

4.4.4 The Pool Setting

In real applications, instead of a complete purchase probability curve, sometimes we can only access to the purchase probabilities of a user when she is offered certain discounts in a prefixed **discount pool** $\mathcal{P}$ [116]. For example, for every user $u \in V$, we only have values of $p_u(c_u)$ when $c_u \in \mathcal{P} = \{5\%, 10\%, \ldots, 100\%\}$. We call this situation the **Pool Setting** of the CIM problem. Correspondingly, we call the situation that we have the complete purchase probability curves of all users the **Curve Setting** of the CIM problem.

Under the pool setting, we do not have the information of derivatives of purchase probability curves. We slightly modify Algorithm 8 by replacing Lines 5 and 6 by an exhaustive search over $c_u \in [max(0, B' - 1), min(1, B')] \cap \mathcal{P}$. Then, the termination condition in Line 2 indicating convergence becomes that we cannot find a pair of users i and j such that by adjusting discounts allocated to them $UI(C)$ can be further improved. To pick two variables c_i and c_j to optimize in each iteration, one simple way is to use the Round-Robin method where we try all possible $O(n^2)$ pairs of c_i and c_j in one batch, and keep iterating until the objective $UI(C)$ cannot be improved in a batch. In practice $O(n^2)$ may be too large to be feasible. We can only consider vertices whose initial discounts are positive in iterations. In such a case, in every batch we only need $O(|S|^2)$ pairs to optimize, where S contains all vertices with positive initial discounts. As we will see in Section 4.6.3, restricting the size of S is a good idea in practice because by doing so we can avoid the "overfitting" issue. Therefore, $|S|^2$ is usually not big in practice.

We also slightly modify the unified discount heuristic under the pool setting. Since we can only set the unified discount c as one element from the discount pool $\mathcal{P}$, it is possible that $\frac{B}{c}$ is not an integer so by offering the unified discount c we cannot use up the budget B. In such a case we first greedily allocate c to $\lfloor \frac{B}{c} \rfloor$ users, and then allocate the rest budget $c_l = B - \lfloor \frac{B}{c} \rfloor c < c$ to the user in the rest users (who do not receive the unified discount c) who can improve the influence spread the most [2].

Example 2. *Fig. 4.1 is a toy example illustrating the differences between integer configuration, unified discount configuration, and continuous configuration. In Figure 4.1, the propagation model is the IC model and the propagation probabilities along edges are all set to* 0.1*. Suppose the seed probability functions for the vertices in this graph are all in the form of* $p_u(c_u) = 2c_u - c_u^2$*, that is, the users are sensitive to discount. When* $B = 1$*, the optimal seeding strategy for DIM is to choose vertex* v_1 *as the single seed, which leads to the best integer configuration is* $C_1 = (1, 0, 0, 0, 0)$ *and* $UI(C_1) = 1.4$*. If we apply the unified discount strategy by setting* τ *as 10, the best unified discount value is* 0.2*. Correspondingly the best unified discount configuration is* $C_2 = (0.2, 0.2, 0.2, 0.2, 0.2)$ *and* $UI(C_2) = 1.89216$*. If we apply the*

[2] We assume c_l is always in the discount pool $\mathcal{P}$, which in reality is easy to be satisfied. For example, if $\mathcal{P} = \{1\%, 2\%, ..., 100\%\}$, our assumption holds.

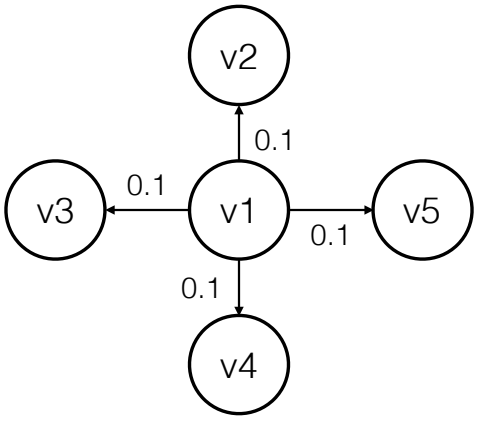


Figure 4.1: An example illustrating the differences among integer configuration, unified discount configuration and continuous configuration.

coordinate descent algorithm and set $C_2 = (0.2, 0.2, 0.2, 0.2, 0.2)$ *as the initial value of configuration, we get a better continuous configuration* $C_3 = (0.3831, 0.1542, 0.1542, 0.1542, 0.1542)$ *and* $UI(C_3) = 1.93533$.

4.5 CIM and DIM

In this section, we examine the relation between the continuous influence maximization problem (CIM) studied in this chapter and the classical and well studied (discrete) influence maximization problem (DIM). Surprisingly, our Coordinate Descent algorithm not only provides us an algorithmic framework for the CIM problem, but also helps us derive relations between CIM and DIM. All results in this section are obtained by utilizing our Coordinate Descent algorithm as an essential building block.

Our first result is that, when the influence function $I(S)$ and the seed probability function $p_u(c_u)$ satisfy certain conditions, the continuous influence maximization problem and the discrete influence maximization problem share the same optimal solution.

Theorem 13 (CIM and DIM)**.** *Given an influence function* $I(S)$ *that is monotonic and submodular, a budget* B *that is a positive integer, and a seed probability function* $p_u(\cdot)$ *for every vertex such that* $\forall u \in V, \forall c_u \in [0, 1]$, $p_u(c_u) \leq c_u$, *the optimal objectives of CIM and DIM are equivalent.*

Proof. The major idea of our proof is to show that, for an arbitrary feasible configuration C, there is an integer configuration C' such that $UI(C') \geq UI(C)$. As $p_u(c_u) \leq c_u$ stated in the theorem, we consider two cases.

First, we consider the situation where $\forall u \in V, p_u(c_u) = c_u$. In such a case, $UI(C)$ actually can be regarded as a multilinear extension of the submodular influence spread function [13]. For any feasible configuration C, in Line 3 of Algorithm 8, if C contains a component c_i

that is not an integer, since B is an integer, C must contains another component c_j $(i \neq j)$ such that c_j is not an integer, either. Therefore, we can always pick non-integers c_i and c_j. Then, we optimize over c_i and c_j by solving the optimization problem in Eq. 4.7. Due to Eq. 4.9, we have

$$\begin{aligned} UI(C) =&(A_1 + A_2 - A_3 - A_4)c_i(B' - c_i)+ \\ &(A_3 - A_1)c_i + (A_4 - A_1)(B' - c_i) \end{aligned}$$

which is a quadratic form of c_i and the coefficient of the quadratic term is $-(A_1 + A_2 - A_3 - A_4)$. Since $I(S)$ is submodular, we have $I(S \cup \{i, j\}) - I(S \cup \{j\}) \leq I(S \cup \{i\}) - I(S)$. Thus, $(A_1 + A_2 - A_3 - A_4) \leq 0$, which means $UI(C)$ is a convex function with respect to c_i. Therefore, the value of x that makes $\frac{\mathrm{d}UI(C)}{\mathrm{d}c_i}|_{c_i=x} = 0$ is the global minimum. Since we are interested in only the maximum, we can ignore the root of $\frac{\mathrm{d}UI(C)}{\mathrm{d}c_i} = 0$ completely. This means that, after optimization, c_i must be either $max(0, B' - 1)$ or $min(B', 1)$.

Let us examine the value of c_i and c_j after optimization under different situations of B'. There are two possible cases.

- If $B' \geq 1$, then $B' - 1 \leq c_i \leq 1$. After optimization over c_i and c_j, if $c_i = B' - 1$, then $c_j = 1$. If $c_i = 1$, then $c_j = B' - 1$.
- If $B' < 1$, then $0 \leq c_i \leq B'$. After optimization over c_i and c_j, if $c_i = 0$, then $c_j = B'$. If $c_i = B'$, then $c_j = 0$.

Thus, after optimization over c_i and c_j, at least one variable of c_i and c_j takes an integer value. In other words, after one iteration we eliminate at least one non-integer c_u. Apparently, after at most n iterations we can make all c_u's integers. Since in every iteration the objective function does not decrease, the final integer configuration C' can achieve an influence spread no smaller than the initial configuration. Therefore, we only need to consider integer configurations, which means CIM degenerates into DIM.

Second, we consider the situation when there exists at least one vertex u such that $p_u(c_u) < c_u$. In other words, we consider for each u, $p_u(c_u) \leq c_u$. Let $\bar{p}_u(c_u)$ be the seed probability function such that for each u, $\bar{p}_u(c_u) = c_u$. Denote by $\overline{UI}(C)$ be the influence spread using $\bar{p}_u(c_u)$ with respect to configuration C. For each u, due to the assumption that $\bar{p}_u(\cdot)$ is continuous, we have $\bar{c}_u$ such that $p_u(c_u) = \bar{p}_u(\bar{c}_u) \leq c_u = \bar{p}_u(c_u)$. Consider two configurations $C = (c_1, \ldots, c_n)$ and $\overline{C} = (\bar{c}_1, \ldots, \bar{c}_n)$. Clearly, $\overline{C} \preceq C$. Due to the monotonicity of $UI(C)$, we have $UI(C) = \overline{UI}(\overline{C}) \leq \overline{UI}(C)$. As the first case, we already prove that $\overline{UI}(C)$ has the same objectives in CIM and DIM when C is an integer configuration. Note that for any integer configuration C, $UI(C) = \overline{UI}(C)$. Thus, the theorem holds in this general case. □

Corollary 13.1. *Given an influence function $I(S)$ that is monotonic and submodular, an integer budget B, if $\forall u \in V, \forall c_u \in [0,1]$, $p_u(c_u) \leq c_u$, then there exists an integer configuration C that is optimal to CIM.*

Theorem 13 also immediately indicates the complexity of the continuous influence maximization problem.

Corollary 13.2. *If maximizing $I(S)$ is NP-hard, and $I(S)$ is monotonic and submodular, then given a social network $G = \langle V, E \rangle$, a budget B and the seed probability function $p_u(c_u)$ for each $u \in V$, maximizing $UI(C)$ over C is also NP-hard.*

To further understand the significance of Theorem 13, we notice that the seed probability function $p_u(c_u)$ represents how user u is sensitive to discount c_u. If $\forall c_u \in [0,1]$ $p_u(c_u) \leq c_u$, then user u is insensitive to discount. Theorem 13 indicates that, if all users in the network are insensitive to discount, then we would better give free products to some seed users, that is, setting $c_u = 1$, to trigger a sizeable cascade propagation.

Utilizing the proof of Theorem 13, we have an upper bound of $UI(C)$ given a configuration C. Define an l-seed set a set of l seeds.

Corollary 13.3 (Upper Bound)**.** *Given an influence function $I(S)$ that is monotonic and submodular, and a discount configuration C. Suppose $K = \lceil \sum_{u \in V} p_u(c_u) \rceil$, then $UI(C) \leq I(S_K^*)$, where S_K^* is the optimal K-seed set for DIM.*

Proof. We prove this corollary by playing a trick of substituting seed probability functions. Given C, we have the seed probability $p_u(c_u)$ for every $u \in V$. We use notations in Section 4.2, $UI(C)$ and $I(S)$, to denote the expected influence of a configuration C and the influence of a seed set S under the setting that $p_u(c_u)$ is the seed probability function of u for every $u \in V$. Let $C' = (p_1(c_1), p_2(c_2), \ldots, p_n(c_n))$ and with out loss of generality, assume $\sum_{u \in V} p_u(c_u)$ equals an integer K. Now, for every $u \in V$, we replace the seed probability function of u by a new one, $p'_u(c_u) = c_u$. Denote by $\widetilde{UI}(C')$ the expected influence of C', and $\tilde{I}(S)$ the influence spread of a seed set S, under the new seed probability function setting. Clearly, $\widetilde{UI}(C') = UI(C)$. Also, $\tilde{I}(S) = I(S)$ for every $S \subseteq V$. According to the proof of Theorem 13, $\widetilde{UI}(C')$ is not optimal under the budget constraint $B = \sum_{u \in V} p_1(c_1) = K$ and $\widetilde{UI}(C') \leq I'(S_K^*)$, where S_K^* is the optimal K-seed set for the DIM problem. Thus, $UI(C) = \widetilde{UI}(C') \leq I'(S_K^*) = I(S_K^*)$. □

Given a budget B, we can solve the following optimization problem to get the maximum expected number of seeds.

$$\begin{aligned} \text{maximize} \quad & \sum_{u \in V} p_u(c_u) \\ \text{s.t.} \quad & 0 \leq c_u \leq 1, \forall u \in V \\ & \sum_{u \in V} c_u = B \end{aligned} \tag{4.14}$$

Since a seed probability function $p_u(c_u)$ can be arbitrarily shaped, $\sum_{u\in V} p_u(c_u)$ can be non-concave and non-convex. By assuming that the maximum of Eq. 4.14 is K, where K is decided by seed probability functions of all vertices and B, we have a data dependent approximation ratio of a DIM algorithm and our CIM algorithm (Algorithm 8).

Theorem 14 (Data-dependent approximation)**.** *Given an influence function $I(S)$ that is monotonic and submodular, and an integer budget B. Let $K = \max_{C\in\mathcal{C}} \lceil \sum_{u\in V} p_u(c_u) \rceil$, where $\mathcal{C} = \{C \mid \mathbf{0} \preceq C \preceq \mathbf{1}, |C|_1 = B\}$. If S_B is a B-seed set that is α-optimal to the DIM problem with budget B, then $I(S_B) \geq \alpha\frac{B}{K} UI(C^*)$, where $UI(C^*) = \max_{C\in\mathcal{C}} UI(C)$. Moreover, if we take C_B, the corresponding configuration to S_B, as the initial value, then Algorithm 8 is also $\alpha\frac{B}{K}$ optimal.*

Proof. Denote by S_B^* and S_K^* the optimal B-seed set and the optimal K-seed set for DIM. By Corollary 13.3 and the monotonicity of $I(S)$, we can immediately get that $UI(C^*) \leq I(S_K^*)$. We can run the greedy algorithm on S_K^* to sort vertices in it. Suppose $S_K^* = \{u_1, u_2, \ldots, u_K\}$, where u_i is the i-th seed picked by the greedy algorithm. Let $S_K^B = \{u_1, \ldots, u_B\}$. Due to the submodularity of $I(S)$ and the greedy algorithm, we have $I(S_K^B) \geq \frac{B}{K} I(S_K^*)$. Obviously $I(S_B^*) \geq I(S_K^B)$. Thus, we have $I(S_B^*) \geq \frac{B}{K} I(S_K^*)$. Moreover, when we take C_B, the corresponding configuration of S_B^* as the initial value for coordinate descent iterations, we can find a configuration C° such that $UI(C^\circ) \geq UI(C_B) = I(S_B^*)$. Thus, Algorithm 8 is also $\alpha\frac{B}{K}$-optimal. □

Although under some conditions, CIM and DIM share the same optimal objectives, it can be shown that in some situations it is not the case, particularly when some users are sensitive to discount.

Example 3. *Consider a social network $G = \langle V, E\rangle$ where $E = \emptyset$. In other words, G is a graph of n isolated vertices. In the independent propagation model or the linear cascade model, if the budget $B = 1$, and for each vertex u, the seed probability function $p_u(c_u) = \sqrt{c_u}$, then the optimal solution for discrete influence maximization is to pick an arbitrary vertex u and the optimal influence spread is 1. However, the optimal solution to continuous influence maximization is to assign $\frac{1}{n}$ discount to each vertex, and the optimal influence spread is $\sqrt{n}$. Thus, not only the optimal solution to discrete influence maximization is not optimal to continuous influence maximization in some cases, but also such solutions to DIM can be arbitrarily bad for CIM as the size of the network becomes very large.*

It can be easily shown that CIM can always achieve an influence spread no smaller than DIM. Given a budget B, one can first run a DIM algorithm to find a seed set of $\lfloor B \rfloor$ seeds. Then, by taking the corresponding integer configuration C of the seed set as the initial configuration, after applying the coordinate descent algorithm, a configuration C' that $UI(C') \geq UI(C)$ can be found.

4.6 Implementation of CIM under Triggering Models

All our discussion above assumes that we have an oracle that can return the exact $I(S)$, the influence spread of a given seed set S. However, under many propagation models, computing $I(S)$ is usually hard and often we can only have an approximation of $I(S)$. This should be considered in implementing our CIM algorithm under a specific propagation model.

In this section, we provide specific algorithms for continuous influence maximization under triggering models [54], where the most widely used propagation models, such as the Independent Cascade (IC) model [20, 19, 54, 104], the Linear Threshold (LT) model [54, 21, 44] and the Continuous-Time diffusion model [35] are all instances. We give a polling based algorithm and discuss how to avoid the "overfitting" problem similar to the issue in machine learning.

4.6.1 Practical Challenges for the Coordinate Descent Algorithm

We first discuss challenges in implementing our CIM algorithm under specific propagation models. In the coordinate descent algorithm, we need to compute 3 coefficients, $(A_1 + A_2 - A_3 - A_4)$, $A_3 - A_1$, and $A_4 - A_1$. Similar to $UI(C)$, these three coefficients can only be estimated by sampling techniques if computing $I(S)$ is #P-hard. All the three coefficients can be very close to 0. Consequently, estimating them may be very challenging.

To tackle the challenge, a practical trick is not to solve $\frac{\mathrm{d}UI(C)}{\mathrm{d}c_i}|_{c_i=x} = 0$. Instead, we can estimate $UI(C)$ directly by trying all possible values of $c_i \in [max(0, B' - 1), min(B', 1)]$. This makes good sense in practice since a budget typically carries a minimum unit. For example, if we want the absolute error of C to be up to 0.01, at most we only need to try 101 different values of c_i because $min(B', 1) - max(0, B' - 1) \leq 1$.

However, even the above trick may face serious challenges in some situations. A key observation is that the gain obtained in an iteration of the coordinate descent algorithm is limited.

Theorem 15 (Gain in an iteration)**.** *In the coordinate descent algorithm, let the configuration before an iteration and that after the iteration be C and C_1, respectively. Then, after the iteration, we have*

$$\begin{aligned} &UI(C_1) - UI(C) \\ \leq &\operatorname*{argmax}_{u \in V} \sum_{S \in 2^{V \setminus \{u\}}} Pr(S; V \setminus \{u\}, C)\big(I(S \cup \{u\}) - I(S)\big) \end{aligned}$$

Proof. Suppose in this iteration c_i and c_j are picked for optimization. Clearly after optimization over c_i and c_j, if c_i and c_j change, then one of them increases and the other decreases, because the sum of c_i and c_j remains a constant. Without loss of generality, assume c_i increases by α and $p_i(c_i + \alpha) - p_i(c_i) = \beta$. Apparently, since $0 \leq p_i(c_i) \leq 1$ and $p_i(c_i)$ is monotonic, $\beta \leq 1$.

Let C_2 be a new configuration such that $C_2(j) = C_1(j) + \alpha$ and all other entries of C_2 are identical to those in C_1. Due to the monotonicity of $UI(C)$, since $C_2 \succeq C_1$, we have $UI(C_2) \geq UI(C_1)$. Moreover,

$$UI(C_2) - UI(C) = \beta \sum_{S \in 2^{V-\{i\}}} Pr(S; V - \{i\}, C)\big(I(S \cup \{u\}) - I(S)\big).$$

Therefore,

$$\begin{aligned} &UI(C_1) - UI(C) \leq UI(C_2) - UI(C) \\ \leq &\sum_{S \in 2^{V-\{i\}}} Pr(S; V - \{i\}, C)\big(I(S \cup \{u\}) - I(S)\big) \\ \leq &\operatorname*{argmax}_{u \in V} \sum_{S \in 2^{V-\{u\}}} Pr(S; V - \{u\}, C)\big(I(S \cup \{u\}) - I(S)\big) \end{aligned}$$

□

Theorem 15 indicates that sometimes the gain after one iteration in the coordinate descent algorithm can be very small, because $I(S \cup \{u\}) - I(S)$ can be close to 0. If we calculate $UI(C)$ by Monte Carlo simulations, such sampling techniques may fail to detect a very small difference between two highly similar configurations. This suggests that algorithms like some traditional influence maximization algorithms [54, 65], where the optimization and estimation (by Monte Carlo simulations) are conducted alternatively, are not suitable for implementing the CIM algorithm. Thus, we turn to algorithms that generate a sketch structure that can be used to estimate everything we need in the optimization process before optimization. Specifically, we adopt a polling-based algorithm that will be illustrated in the following of this section.

4.6.2 Polling-based CIM Algorithm

Based on the challenges of implementing CIM algorithms discussed above, we turn to algorithms that generate a sketch structure that can be used to estimate everything we need in the optimization process before optimization. Specifically, we adopt the polling sketch to approximate influence spreads of configurations and we conduct optimization on the polling sketch.

We first show how to approximate $UI(C)$ by a polling sketch.

Theorem 16. *Given a graph $G = \langle V, E\rangle$, a triggering model I where a propagation can be simulated and the seed probability functions of all vertices, we generate a polling sketch H with m_H poll samples generated according to G and I. Then for a configuration C, $\frac{n * f_H(C)}{m_H}$ is an unbiased estimation of $UI(C)$, where $f_H(C) = \sum_{h \in H} f_h(C)$ and $f_h(C) = 1 - \prod_{u \in h}(1 - p_u(c_u))$.*

Proof. By the definition of $UI(C)$, we have

$$\frac{UI(C)}{n} = \frac{1}{n}\sum_{S\subseteq V} Pr(S;V,C)I(S) = \sum_{S\subseteq V} Pr(S;V,C)\frac{I(S)}{n}$$

For a triggering model, by randomly generating a poll sample h via a Monte Carlo simulation of the reverse propagation process, we have that $A(S,h)$ is an unbiased estimation of $\frac{I(S)}{n}$, where $A(S,h) = 1$ if $S \cap h \neq \emptyset$, and $A(S,h) = 0$ otherwise. So $E[A(S,h)] = \frac{I(S)}{n}$ and

$$\frac{UI(C)}{n} = \sum_{S\subseteq V} Pr(S;V,C)E[A(S,h)]$$

For the expectation $E[A(S,h)]$, the randomness is over the random hyper edge h. Denote by $Pr(h)$ the probability of generating a poll sample h. So $E[A(S,h)] = \sum_{h\subseteq V} Pr(h)A(S,h)$, and we have

$$\begin{aligned}\frac{UI(C)}{n} &= \sum_{S\subseteq V} Pr(S;V,C)\sum_{h\subseteq V} Pr(h)A(S,h)\\ &= \sum_{h\subseteq V} Pr(h)\sum_{S\subseteq V} Pr(S;V,C)A(S,h)\end{aligned}$$

The sum $\sum_{S\subseteq V} Pr(S;V,C)A(S,h)$ can be regarded as the expectation of $A(S,h)$ when h is fixed and S is randomly generated according to $Pr(S;V,C)$. It is obvious that when $S \cap h = \emptyset$, $Pr(S;V,C)A(S,h) = 0$, and when $S \cap h \neq \emptyset$, $A(S,h) = 1$. So

$$\sum_{S\subseteq V} Pr(S;V,C)A(S,h) = \sum_{S\subseteq V, S\cap h\neq\emptyset} Pr(S;V,C)$$

The right-hand side is the probability that the randomly generated set S has at least one vertex in h. Recall that $Pr(S;V,C) = \prod_{u\in S} p_u(c_u)\prod_{v\in V\setminus S}(1-p_v(c_v))$, which means each u belongs to S independently. Thus, obviously,

$$\begin{aligned}\sum_{S\subseteq V} Pr(S;V,C)A(S,h) &= \sum_{S\subseteq V, S\cap h\neq\emptyset} Pr(S;V,C)\\ &= 1 - \prod_{u\in h}(1-p_u(c_u))\end{aligned}$$

Combining the above results, we have

$$\frac{UI(C)}{n} = \sum_{h\subseteq V} Pr(h)[1 - \prod_{u\in h}(1-p_u(c_u))]$$

The right-hand side is the expectation of $1 - \prod_{u \in h} (1 - p_u(c_u))$, when h is randomly generated according to $Pr(h)$. Thus,

$$\frac{UI(C)}{n} = E[1 - \prod_{u \in h} (1 - p_u(c_u))]$$

By the linearity of expectation, we have

$$E[\frac{n * \sum_{h \in H} [1 - \prod_{u \in h} (1 - p_u(c_u))]}{m_H}] = UI(C).$$

□

According to Theorem 16, we build a polling sketch H of m_H poll samples. Then we solve the following optimization problem.

$$\begin{aligned} \text{maximize} \quad & f_H(C) = \sum_{h \in H} f_h(C) = \sum_{h \in H} [1 - \prod_{u \in h} (1 - p_u(c_u))] \\ \text{s.t.} \quad & 0 \leq c_u \leq 1, \forall u \in V \\ & \sum_{u \in V} c_u \leq B \end{aligned} \tag{4.15}$$

$f_H(C)$ is an unbiased estimation of $UI(C)$. In Section 4.4.2, we proved that the CIM problem (Eq. 4.3) is NP-hard in general, we now prove that so is the optimization problem in Eq. 4.15.

Theorem 17. *Given a polling sketch H and the seed probability function $p_u(c_u)$ for each $u \in V$, the optimization problem in Eq. 4.15 is NP-hard in general.*

Proof. We prove the NP-hardness of the optimization problem in Eq. 4.15 by a reduction from the max B-set cover problem, which is known to be NP-hard. Consider an arbitrary instance of the max B-set cover problem, which consists of the set of all elements $E = \{e_1, \ldots, e_n\}$, and a collection of set $\mathcal{S} = \{S_1, \ldots, S_m\}$, where $S_i \subseteq V$ for all i, and an integer B. The objective is to choose B sets from $\mathcal{S}$ such that the cardinality of the union of these k sets is maximized. Denote by OPT the optimal cardinality.

We reduce such an instance to an instance of Eq. 4.15. We first create a polling sketch H as follows. We set the vertex set V as $\{1, \ldots, m\}$, and create n poll samples $\{h_1, \ldots, h_n\}$. Note that a poll sample essentially is a set of vertices. For each h_i and a vertex $u \in V$, we make $u \in h_i$ if $e_i \in S_u$. We also set the budget in Eq. 4.15 as B, and for each $u \in V$, we set the seed probability function $p_u(c_u) = c_u$. The reduction is obviously in polynomial time.

Clearly, for any $S \subseteq V$, $\mathcal{D}_H(S) = |\cap_{u \in S} S_u|$. Thus, we have $\max_{S \subseteq V, |S| = B} \mathcal{D}_H(S) = OPT$. Now all we need to prove is that the optimality of Eq. 4.15 is $\max_{S \subseteq V, |S| = B} \mathcal{D}_H(S)$. This is equivalent to prove that Eq. 4.15 in the instance constructed above has an optimal configuration C whose entries are all integers (either 0 or 1).

The proof is pretty like the proof of Theorem 13. For any valid configuration C such that $\sum_{u\in V} c_u = B$, if C contains non-integer entries, it must contain at least 2 non-integer entries. We pick two non-integer entries c_i and c_j, and fix all other entries. Suppose $c_i + c_j = B'$, then we can optimize over c_i to further improve the objective $f_H(C)$. It is easy to verify that $f_H(C)$ is a convex function with respect to c_i when satisfying $c_j = B' - c_i$, $0 \le c_i \le 1$ and $0 \le c_j \le 1$ (because the coefficient of the quadratic term is positive). Thus, similar to the construction in the proof of Theorem 13, after optimizing over c_i, either c_i or c_j becomes an integer and the objective $f_H(C)$ does not decrease. Therefore, there exist an integer configuration C such that $f_H(C)$ is optimal to Eq. 4.15. □

As we pointed out in Section 4.4.2, $UI(C)$ is not necessarily concave or convex. It is easy to verify that neither is $f_H(C)$. Thus, we seek for stationary points as illustrated in Section 4.4.2, which are necessary conditions of local maxima points (note that finding local maxima of non-concave functions is NP-hard in general [75]). Due to the linear contraints in Eq. 4.15, a configuration C is a stationary point if and only if it satisfies the Karush-Kuhn-Tucker (KKT) conditions [11]. By a simple derivation, we have that if C is a **KKT point** of Eq. 4.15, it should satisfy the following.

$$\nabla_u f_H(C) = \begin{cases} \ge \lambda & c_u = 1 \\ = \lambda & 1 > c_u > 0 \\ \le \lambda & c_u = 0 \end{cases} \quad \forall u \in V \tag{4.16}$$

where $\nabla_u f_H(C)$ is the partial derivative of $f_H(C)$ with respect to c_u and

$$\nabla_u f_H(C) = p_u(c_u)' \sum_{h\in H_u} \prod_{i\in h, i\neq u} (1 - p_i(c_i)) \tag{4.17}$$

The condition in Eq. 4.16 is equivalent to

$$\max_{u:c_u<1} \nabla_u f_H(C) \le \min_{v:c_v>0} \nabla_v f_H(C) \tag{4.18}$$

Our algorithmic framework Algorithm 8 can be used to find KKT points. In every iteration, we pick the discounts of two vertices c_u and c_v to optimize, where $u = \arg\max_{i:c_i<1} \nabla_i f_H(C)$ and $v = \arg\min_{j:c_j>0} \nabla_j f_H(C)$. But this method is time consuming, since in each iteration we need $O(n)$ time to find c_u and c_v to optimize. To avoid this, we adopt a Shrink-and-Expansion strategy which is used in finding dense subgraphs [67].

Algorithm 10 2-Coordinate Descent Shrink-and-Expansion Algorithm.

Input: A polling sketch H, and an initial configuration C
Output: C

1: $S \leftarrow \{u \mid c_u > 0\}$
2: **while true do**
3: Use the 2-Coordinate Descent algorithm and take C as the initial value to find a local KKT point C^{new} on S
4: $C \leftarrow C^{new}$
5: $S \leftarrow \{u \mid c_u > 0\}$
6: $t \leftarrow \min_{v \in S} \nabla_v f_H(C)$
7: $Z \leftarrow \{i \mid \nabla_i f_H(C) > t\}$
8: **if** $Z \neq \emptyset$ **then**
9: $S \leftarrow S \cup Z$
10: **else**
11: **break**
12: **end if**
13: **end while**
14: **return** C

To illustrate the Shrink-and-Expansion method, we first define a **local KKT point** on $S \subseteq V$ as a configuration C satisfying

$$c_u = 0 \quad if \quad u \notin S$$
$$\nabla_u f_H(C) = \begin{cases} \geq \lambda & c_u = 1 \\ = \lambda & 1 > c_u > 0 \\ \leq \lambda & c_u = 0 \end{cases} \quad \forall u \in S \tag{4.19}$$

Algorithm 10 describes the Shrink-and-Expansion algorithm. Line 3 is the Shrink stage, because after Line 3 the number of non-zero entries of C does not get bigger. Line 7 is the expansion stage, where we add vertices whose partial derivatives is greater than the threshold t with respect to S for iterations, and the threshold is set to the smallest partial derivative of vertices in S. If in the expansion stage, we do not have any vertices to add to S, clearly the current C satisfies the KKT condition in Eq. 4.16 so it is already a KKT point. If C is only a local KKT point on S but not a KKT point to Eq. 4.15, in the expansion stage we must add some vertices to S. Also, after one shrink stage and one expansion stage, the objective $f_H(C)$ is obviously non-decreasing. Thus, Algorithm 10 always converges and it converges to a KKT point.

Theorem 18 (KKT Point)**.** *Algorithm 10 converges to a KKT configuration C.*

The advantage of Algorithm 10 is that normally only a small number of vertices are involved in computation, especially when the initial configuration C does not have many non-zero entries. Thus, comparing to directly running line 3 of Algorithm 10 to find a KKT

point C, which is also a local KKT point on the all vertices set V, Algorithm 10 is more efficient.

To find a good initial configuration C, we first define

$$f_H(S;c) = \sum_{S' \in 2^S} Pr(S';S,c) \sum_{h \in H} [1 - \prod_{u \in S' \cap h} (1 - p_u(c))]$$

It is not difficult to verify that when c is fixed, similar to $UI(S;c)$, $f_H(S;c)$ is also monotonic and submodular with respect to S. Thus, we can totally apply the Unified Discount heuristic (Algorithm 9) described in section 4.4.3.

4.6.3 Deciding Sample Size to Avoid "Overfitting"

Our algorithm needs to build a polling sketch H. How many poll samples do we need to achieve a good result? What are the risks if we only sample a small number of random poll samples? We answer these questions by making an analogy of machine learning.

An Analogy between Influence Maximization and Machine Learning

We first make an analogy between influence maximization based on the polling method and Empirical Risk Minimization (ERM) machine learning [94].

In machine learning, we have the instance domain $\mathcal{X}$, where an element $x \in \mathcal{X}$ is an instance. Denote by $\Pr(x)$ the probability of obtaining x as a random sample from $\mathcal{X}$. If the domain $\mathcal{X}$ is continuous, then $\Pr(x)$ is the probability density of x. The ultimate goal is as follows

$$\begin{aligned} \text{maximize} \quad & \sum_{x \in \mathcal{X}} \Pr(x) g_x(\eta) \\ \text{s.t.} \quad & \eta \in \mathcal{H} \end{aligned} \tag{4.20}$$

where η is a model and $\mathcal{H}$ is the hypothesis set containing all the models we can use. $g_x(\eta)$ is a function measuring how good is a model η on an instance x. For example, for a binary classification task, suppose we try to find an optimal linear classifier, and let $x = (x_1, \ldots, x_n)$ where $x_n \in \{0, 1\}$ is the class label of x. Then $g_x(\eta) = 1$ if $(\sum_{i=1}^{n-1} \eta_i x_i + \eta_n) * x_n > 0$ and $g_x(\eta) = 0$ otherwise.

However, solving Eq. 4.20 is difficult since we normally cannot iterate every instance $x \in \mathcal{X}$. Instead, we can draw a set $\mathcal{T}$ that contains a number of i.i.d. samples, where a sample x is drew according to the probability $\Pr(x)$. We call $\mathcal{T}$ the training set. Adopting the Empirical Risk Minimization (ERM) rule in machine learning, to solve Eq. 4.20, we

Table 4.2: IM & CIM from machine learning perspective.

	IM	CIM
Instances	All possible poll samples	All possible poll samples
Training Samples	m_H samples poll samples	m_H samples poll samples
Hypothesis set	$\mathcal{S} = \{S \mid S \subseteq V, \lvert S\rvert = B\}$	$\mathcal{C} = \{C \mid \mathbf{0} \preceq C \preceq \mathbf{1}, \lvert C\rvert_1 = B\}$
Objective	$I(S)$	$UI(C)$
Objective on training data	$n * \mathcal{D}_H(S)/m_H$	$n * f_H(C)/m_H$

solve the following optimization problem.

$$\begin{aligned} \text{maximize} \quad & \sum_{x \in \mathcal{T}} g_x(\eta) \\ \text{s.t.} \quad & \eta \in \mathcal{H} \end{aligned} \tag{4.21}$$

We call $\sum_{x \in \mathcal{X}} \Pr(x) g_x(\eta)$ in Eq. 4.20 the **objective** and $\sum_{x \in \mathcal{T}} g_x(\eta)$ in Eq. 4.21 the **objective on training data** of a machine learning task.

We write the problems of IM and CIM as machine learning problems, as shown in Table 4.2. For both IM and CIM, the expectation of the objective on training data equals the real objective for any hypothesis. Similar to machine learning, our task in IM or CIM is to choose a good hypothesis with high objective value from the hypothesis set ($\mathcal{S}$ or $\mathcal{C}$), by utilizing the training samples and the objective function on the training data.

One key point of machine learning is, how many training samples do we need to achieve a provable performance guarantee [94]? If we do not have sufficient training samples, we may get overfitting, which means we may choose a hypothesis whose expected objective value on the whole data set is much smaller than its objective value on the training samples.

Deciding the sample size (number of poll samples) is also the key point of state-of-the-art influence maximization algorithms based on polling method [10, 104, 103]. Similar to machine learning, if the random poll samples (training samples) are not sufficient, we may find a seed set with poor quality and get "overfitting" (the estimated influence of the seed set on the sampled polling sketch is much greater than it real influence spread).

One major idea of these algorithms is to sample enough poll samples to achieve a **uniform convergence** [94] on $\mathcal{S}$ such that

$$\Pr\{\exists S \in \mathcal{S}, \lvert \frac{\mathcal{D}_H(S)}{m_H} - \frac{I(S)}{n} \rvert > \epsilon\} \leq \delta \tag{4.22}$$

When this uniform convergence is achieved, the value $\frac{n * \mathcal{D}_H(S)}{m_H}$ is a good estimation of $I(S)$ for any B-sized seed set S. Thus, by maximizing $\frac{n * \mathcal{D}_H(S)}{m_H}$, we can find a good seed set with near-optimal influence spread (the approximation ratio is decided by ϵ). To achieve Eq. 4.22, a necessary condition is to sample enough poll samples such that for an arbitrary B-sized

seed set S,

$$\Pr\{|\frac{\mathcal{D}_H(S)}{m_H} - \frac{I(S)}{n}| > \epsilon\} \leq \delta / \binom{n}{B} \tag{4.23}$$

Once Eq. 4.23 is satisfied, by applying the union bound, Eq. 4.22 holds because the number of different B-seed sets (i.e., classifiers) is $\binom{n}{B}$.

Sample Size for CIM

We discuss how to decide a proper sample size (number of random poll samples) for the CIM problem. One difficulty is that optimizing $f_H(C)$ over a number of poll samples is NP-hard as illustrated by Theorem 17. Moreover, $f_H(C)$ is often neither convex nor concave, and so far we can only find a KKT point but we do not know how big the gap between a KKT point and the optimal configuration is. Therefore, unlike influence maximization, where we can get a constant approximation factor $(1 - 1/e - \epsilon)$ (ϵ is decided by the sample size) with high probability, for the CIM problem, it is hard to figure out the gap between a solution obtained by optimizing $f_H(C)$ over a given number of poll samples and the optimal solution. Thus, we seek for a weaker goal that, for the configuration C obtained by optimizing C, with high probability, $\frac{f_H(C)}{m_H}$ is close to $\frac{UI(C)}{n}$. Note that, according to the machine learning setting of influence maximization in Section 4.6.3, $\frac{f_H(C)}{m_H}$ is the accuracy of C on the training data and $\frac{UI(C)}{n}$ is the real accuracy of C. Formally, we define "overfitting" in CIM as

$$\frac{f_H(C)}{m_H} - \frac{UI(C)}{n} > \epsilon \tag{4.24}$$

where C is returned by Algorithm 10 with m_H poll samples as inputs. Our goal is to sample m_H poll samples such that

$$Pr\{|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon\} \geq 1 - \delta \tag{4.25}$$

Just achieving Eq. 4.25 may cause "underfitting" [94] , where although $|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon$, $\frac{UI(C)}{n}$ might not be good enough. This is because fixing the sample size m_H, the smaller $\frac{UI(C)}{n}$, the higher probability that $|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon$, according to the Chernoff bound [72]. Thus, just achieving Eq. 4.25 may lead to find a poor configuration.

We fix the underfitting issue by setting a good configuration C as the initial configuration of Algorithm 10. Taking Influence Maximization (IM) as the baseline, our goal is to find a configuration that is at least as good as the solution produced by the state-of-the-art IM algorithm. We run a state-of-the-art IM algorithm [10, 104, 103, 82] first to get a discrete configuration C_1, then we use the poll samples, apply the Unified Discount algorithm to find a better configuration C_2, and apply Algorithm 10 to find an even better configuration C_3. Here "better" is with respect to the objective value on poll samples. Clearly $f(C_3) \geq f(C_2) \geq f(C_1)$. Suppose $f(C_2) = f(C_1) + \delta_2 * m_H$ and $f(C_3) = f(C_1) + \delta_3 * m_H$. If

$|\frac{f(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon$ holds for $C = C_1, C_2, C_3$, then we have

$$\begin{aligned} &n(\frac{f(C_1)}{m_H} - \epsilon) \leq UI(C_1) \leq n(\frac{f(C_1)}{m_H} + \epsilon) \\ &UI(C_2) \geq UI(C_1) + n(\delta_2 - 2\epsilon) \\ &UI(C_3) \geq UI(C_1) + n(\delta_3 - 2\epsilon) \end{aligned} \tag{4.26}$$

Thus, if $\delta_3 \geq \delta_2 \geq 2\epsilon$, we have $UI(C_2) \geq UI(C_1)$ and $UI(C_3) \geq UI(C_1)$. If both δ_2 and δ_3 are no greater than 2ϵ, to avoid the risk of getting a configuration C such that $UI(C) \leq UI(C_1)$, we can just conservatively pick C_1 as our final configuration. Therefore, taking IM as the baseline, we can avoid the underfitting issue where our final configuration is at least as good as the solution of IM.

One necessary condition of Eq. 4.25 is the **uniform convergence** on $\mathcal{C}$, that is, we need to ensure

$$Pr\{\forall C \in \mathcal{C}, |\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon\} \geq 1 - \delta \tag{4.27}$$

where $\mathcal{C} = \{C \mid \mathbf{0} \preceq C \preceq \mathbf{1}, |C|_1 = B\}$ is our hypothesis set. Unlike influence maximization, where the hypothesis set $\mathcal{S}$ contains only $\binom{n}{B}$ hypotheses, $\mathcal{C}$ is uncountablely infinite. This seems to be a trouble for us to apply the union bound, but we show that we can still achieve the uniform convergence in Eq. 4.25 by sampling a finite number of poll samples.

Theorem 19 (Uniform Convergence)**.** *By sampling $m_H \geq \frac{3(n \ln 2 + \ln \frac{2}{\delta})}{\epsilon^2}$ poll samples, we have $Pr\{\forall C \in \mathcal{C}, |\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon\} \geq 1 - \delta$, where $\mathcal{C} = \{C \mid \boldsymbol{0} \preceq C \preceq \boldsymbol{1}, |C|_1 = B\}$.*

Proof. We show that $\frac{f_H(C)}{m_H}$ can be represented by a convex combination of a finite number of unbiased estimations of influence spread of seed sets. $f_H(C) = \sum_{h \in H}[1 - \prod_{u \in h}(1 - p_u(c_u))]$. In the proof of Theorem 16, we demonstrate that

$$\sum_{S \subseteq V} Pr(S; V, C) A(S, h) = 1 - \prod_{u \in h}(1 - p_u(c_u))$$

where $A(S, h) = 1$ if $S \cap h \neq \emptyset$, and $A(S, h) = 0$ otherwise. Thus,

$$\begin{aligned} f_H(C) &= \sum_{h \in H}[1 - \prod_{u \in h}(1 - p_u(c_u))] \\ &= \sum_{h \in H} \sum_{S \subseteq V} Pr(S; V, C) A(S, h) \\ &= \sum_{S \subseteq V} Pr(S; V, C) \sum_{h \in H} A(S, h) \end{aligned}$$

Note that $\sum_{h \in H} A(S, h)$ is actually $\mathcal{D}_H(S)$. So we have

$$\frac{f_H(C)}{m_H} = \sum_{S \subseteq V} Pr(S; V, C) \frac{\mathcal{D}_H(S)}{m_H}$$

Since $Pr(S;V,C)$ is the probability of generating a seed set S when the discount configuration is C, clearly $\frac{f_H(C)}{m_H}$ is a convex combination of $\frac{\mathcal{D}_H(S)}{m_H}$ over all possible S. In addition, $\frac{UI(C)}{n} = \sum_{S\subseteq V} Pr(S;V,C)\frac{I(S)}{n}$ and $\frac{UI(C)}{n}$ is a convex combination of $\frac{I(S)}{n}$ with the same coefficients. Recall that $\frac{\mathcal{D}_H(S)}{m_H}$ is an unbiased estimation of $\frac{I(S)}{n}$. If for all $S \subseteq V$, $|\frac{\mathcal{D}_H(S)}{m_H} - \frac{I(S)}{n}| \leq \epsilon$, then it is guaranteed that $|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon$.

We set $m_H = \frac{3(n\ln 2+\ln\frac{2}{\delta})}{\epsilon^2}$. Applying the Chernoff bound [72], we obtain that for a seed set $S \subseteq V$,

$$Pr\{|\frac{\mathcal{D}_H(S)}{m_H} - \frac{I(S)}{n}| \geq \epsilon\} \leq \frac{\delta}{2^n}$$

Applying the union bound, we have

$$Pr\{|\frac{\mathcal{D}_H(S)}{m_H} - \frac{I(S)}{n}| \leq \epsilon, \forall S \subseteq V\} \geq 1-\delta$$

Therefore, when $m_H = \frac{3(n\ln 2+\ln\frac{2}{\delta})}{\epsilon^2}$,

$$Pr\{|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon, \forall C \in \mathcal{C}\} \geq 1-\delta.$$

□

Similar to many machine learning methods, if we do not want to sample too many poll samples, we can play the regularization trick to narrow down the hypothesis set. In CIM, the regularization is to add a constraint to C such that the number of non-zero entries (or $|C|_0$) is no greater than a threshold N. Therefore, the hypothesis set becomes $\mathcal{C}_N = \{C \mid \mathbf{0} \preceq C \preceq \mathbf{1}, |C|_1 = B, |C|_0 \leq N\}$. Note that $|\mathcal{C}_N| = \sum_{i=B}^{N} \leq (\frac{en}{N})^N$. Similar to Theorem 19, we have the following corollary.

Corollary 19.1 (Regularization)**.** *By setting the number of poll samples* $m_H \geq \frac{3(N+N\ln\frac{n}{N}+\ln\frac{2}{\delta})}{\epsilon^2}$, *we have* $Pr\{\forall C \in \mathcal{C}_N, |\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}| \leq \epsilon\} \geq 1-\delta$, *where* $\mathcal{C}_N = \{C \mid \boldsymbol{0} \preceq C \preceq \boldsymbol{1}, |C|_1 = B, |C|_0 \leq N\}$.

When we have the regularization with parameter N, we slightly modify Algorithm 10. In the expansion stage (Line 7), if $|Z| > N - |S|$, we truncate Z by only keeping the top $N - |S|$ vertices with respect to their partial derivatives in Z, and in the next while loop, we only do the shrink stage and return the C obtained.

Theorem 19 and Corollary 19.1 provide necessary conditions for avoiding overfitting in CIM. These two bounds of m_H may be loose. Thus, in practice, we use a similar strategy as the state-of-the-art IM algorithm in [82], where we keep two hyper graphs H_1 and H_2 that both have m_H poll samples. We maximize $f_{H_1}(C)$ to get C on H_1, and estimate $\frac{UI(C)}{n}$ using H_2. Suppose $\lfloor f_{H_2}(C)\rfloor = L$. Note that $f_{H_2}(C) = \sum_{h\in H_2} f_h(C)$ where $f_h(C) = 1 - \prod_{u\in h}(1-p_u(c_u))$. We find the smallest index i_L of poll samples in H_2 such

Algorithm 11 Optimization-and-Estimation Strategy

Input: Two polling sketchs H_1 and H_2 that both have m_H poll samples
Output: C

1: Set M to $\frac{3(n\ln 2+\ln\frac{2}{\delta})}{\epsilon^2}$ (without regularization) or $\frac{3(N+N\ln\frac{n}{N}+\ln\frac{2}{\delta})}{\epsilon^2}$ (with regularization)
2: **while** $m_H \leq M$ **do**
3: Optimize $f_{H_1}(C)$ on H_1 to get C
4: $L \leftarrow \lfloor f_{H_2}(C) \rfloor$
5: Find the smallest index i_L such that $\sum_{i=1}^{i_L} f_{h_i}(C) \geq L$, where $h_1, h_2, \ldots, h_{i_L} \in H_2$
6: $e \leftarrow \max(|\frac{f_{H_1}(C)}{m_H} - \frac{L}{(1+\epsilon_1)i_L}|, |\frac{f_{H_1}(C)}{m_H} - \frac{L}{(1-\epsilon_1)i_L}|)$
7: **if** $e \leq \epsilon$ **then**
8: **break**
9: **else**
10: $H_1 \leftarrow H_1 \cup H_2$
11: Sample $|H_1| = 2m_H$ random poll samples to rebuild H_2
12: $m_H \leftarrow 2 * m_H$
13: **end if**
14: **end while**
15: **return** C

that $\sum_{i=1}^{i_L} f_{h_i}(C) \geq L$, where h_i is the i-th poll sample of H_2. Since $f_h(C)$ is a random variable (due to the randomness of h) distributed in $[0,1]$ and $E[f_h(C)] = \frac{UI(C)}{n}$, we can apply the Stop-Rule Theorem [28] and obtain that $\frac{L}{i_L}$ is an (ϵ_1, δ) estimation of $\frac{UI(C)}{n}$, where $1 + (1+\epsilon_1)\frac{4(e-2)\ln\frac{2}{\delta}}{\epsilon_1^2} = L$. Thus, with probability at least $1-\delta$, we have $\frac{UI(C)}{n} \in [\frac{L}{(1+\epsilon_1)i_L}, \frac{L}{(1-\epsilon_1)i_L}]$. Then we check if the maximum possible estimation error of $\frac{f_{H_1}(C)}{m_H}$, $\max(|\frac{f_{H_1}(C)}{m_H} - \frac{L}{(1+\epsilon_1)i_L}|, |\frac{f_{H_1}(C)}{m_H} - \frac{L}{(1-\epsilon_1)i_L}|)$, is less than ϵ. If it is, we are done. If not, we set $H_1 = H_1 \cup H_2$ and we rebuild H_2 by sampling $|H_1| = 2m_H$ random poll samples. Then we redo the above process until the maximum possible estimation error of $\frac{f_{H_1}(C)}{m_H}$ is less than ϵ, or $|H_1|$ exceeds the value of m_H in Theorem 19 or Corollary 19.1. We describe the Optimization-and-Estimation Strategy in Algorithm 11.

The input of Algorithm 11 can be obtained by the state-of-the-art algorithm SSA-fix [82, 51] for influence maximization, where SSA-fix exactly outputs two polling sketchs H_1 and H_2 that have the same size. The seed set returned by SSA-fix can be transformed to an integer configuration and be used to find a good initial configuration. Moreover, the result of SSA-fix can help us control a relative error (note that our goal Eq. 4.25 controls an absolute error). Suppose the integer configuration returned by SSA-fix is C_1, we just need to set $\epsilon = O(\frac{f_{H_1}(C_1)}{m_H})$ because it is guaranteed that $f_{H_1}(C) \geq f_{H_1}(C_1)$ where C is obtained in Line 3 in Algorithm 11. The coefficient hidden in $O(\frac{f_{H_1}(C_1)}{m_H})$ is decided by the relative error of SSA-fix, our desired relative error and the parameter δ in Eq. 4.25.

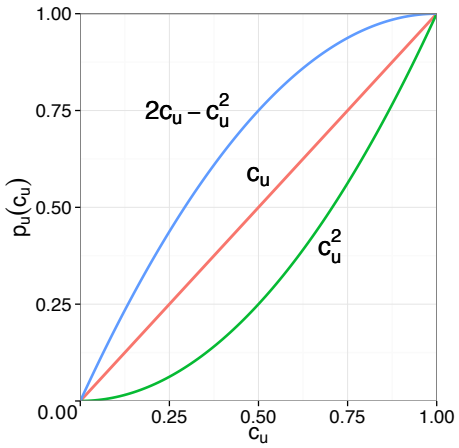


Figure 4.2: The seed probability functions used in the experiments.

Table 4.3: Datasets

Network	n	m	Average Degree
wiki-Vote	7,115	103,689	14.6
ca-AstroPh	18,772	396,220	21.1
com-dblp	317,080	2,099,732	6.6
com-LiveJornal	3,997,962	69,362,378	17.3

4.7 Empirical Evaluation

To examine the effectiveness and efficiency of our methods, in this section, we report experiments on four real networks with synthesized seed probability functions to test our proposed methods. The experiment results show that the continuous influence maximization strategy can significantly improve influence spread without incurring dramatic extra overheads compared to discrete influence maximization.

4.7.1 Experimental Settings

We ran our experiments on four real network data sets that are publicly available in SNAP (`http://snap.stanford.edu/data/index.html`). Table 4.3 shows the details of the four data sets. All networks are treated as directed graphs, which means if a network is undirected, every undirected edge (u, v) is processed as two directed edges (u, v) and (v, u).

In our experiments, we adopted the Independent Cascade (IC) model as the propagation model, which is the most widely used triggering model in literature [20, 19, 54, 104, 82]. Following the most popular settings of the IC model [20, 19, 54, 104, 82], we set the propagation probability of a directed edge (u, v) to $\frac{1}{in-degree(v)}$.

For seed probability functions, unfortunately we do not have access to any such real data sets for the purpose of experiments. Thus, we used synthesized seed probability functions.

Given a network $G = \langle V, E\rangle$, we randomly assigned $p_u(c_u) = 2c_u - c_u^2$ to a p_1 portion of vertices, which means that those users were sensitive to discount. Then a p_2 portion of vertices were assigned with $p_u(c_u) = c_u$, and a p_3 portion of vertices were assigned with $p_u(c_u) = c_u^2$. These users (vertices) were insensitive to discount. We maintain $p_1 + p_2 + p_3 = 1$. Figure 4.2 shows the curves of the two seed probability functions as well as the function $p_u(c_u) = c_u$ as the reference. In our experiments, we set $P = (p_1, p_2, p_3)$ as $(0.85, 0.1, 0.05)$, $(0.75, 0.15, 0.1)$ and $(0.65, 0.2, 0.15)$. For each P, we ran 20 independent experiments by randomly assigning seed probability functions to vertices. All results reported are the averages taken over the results of 20 independent experiments.

We also tested the **pool setting** described in Section 4.4.4, although the aim of this chapter is to settle the CIM problem under the curve setting. We still used the 3 curves $p_u(c_u) = 2c_u - c_u^2$, $p_u(c_u) = c_u$ and $p_u(c_u) = c_u^2$, and portion of each curve is set the same as described above. We set the discount pool $\mathcal{P} = \{1\%, 2\%, 3\%, \ldots, 100\%\}$.

We compare three algorithms: discrete influence maximization (IM), the unified discount heuristic for finding a good initial configuration (UC), and applying the coordinate descent algorithm based on the unified discount heuristic (UC+CD). Note that except for IM, all other algorithms are CIM algorithms since they can allocate non-integer discounts to users.

We adopted the state-of-the-art SSA-fix [82, 51] algorithm as the IM implementation. According to Theorem 14, for the result of IM, the better approximation it is to DIM, the better approximation it is to CIM. Thus, we implemented the IM algorithm by setting $\epsilon = 0.03$ and $\delta = \frac{1}{n}$, which means with probability at least $1 - \frac{1}{n}$, the result of IM is at least 60%-optimal to DIM. For our CIM algorithms UC and UC+CD, we used our Algorithm 11 in section 4.6.3 to avoid the "overfitting" issue. The two polling sketches generated by the SSA-fix [82, 51] algorithm are used as the input of Algorithm 11. We played the regularization trick and set $N = 20B$, which means we offer non-zero discounts to at most $20B$ users. The absolute error ϵ of $|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}|$ in Eq. 4.24 was set to 0.01. For our unified discount heuristic (UC), as introduced in at the end of section 4.6.2, we set $\tau = 20$ since $N = 20B$.

All algorithms were implemented in C# and ran on an Windows 10 computer with Intel(R) Core(TM) i7-3770 3.40GHz CPU, 32GB main memory.

4.7.2 Effectiveness

Fig. 4.3 and Fig. 4.4 show the influence spread of each algorithm under different settings of parameters. Note that in our Algorithm 11 described in Section 4.6.3, we have two polling sketchs. The second one is for testing the influence spread of a given configuration. By the Stop Rule Theorem [28], we find that the second polling sketch estimates influence spreads of configurations obtained by different algorithms accurately. With very high probability $(1 - O(\frac{1}{n}))$, the relative error of estimations never exceeds 2%. From the results we find that all CIM algorithms can significantly increase the expected influence spread compared to discrete influence maximization (IM).

It is not surprising that our CIM algorithms can achieve slightly higher influence spread under the curve setting than under the pool setting, since the curve setting fully utilizes the purchase probability curves. Moreover, the impact of adjusting the portions different purchase probability curves (the parameter P) is limited. Compared to $P = (0.85, 0.1, 0.05)$, when we set $P = (0.65, 0.2, 0.15)$ the influence spreads of UC and UC+CD only decrease by at most 2%.

Under the curve setting, the improvement of UC to IM is from 3% to 40%, the improvement of UC+CD to UC is from 1% to 9.5% and the improvement of UC+CD to IM is from 12% to 43%. Under the pool setting we have similar findings, where the improvement of UC to IM is also 3% to 40%, the improvement of UC+CD to UC is from 1% to 9.2% and the improvement of UC+CD to IM is from 11.7% to 42%. Thus, we can see that both UC and UC+CD can significantly improve influence spread compared to the baseline method IM.

Effectiveness of Setting $\tau = 20$ in UC

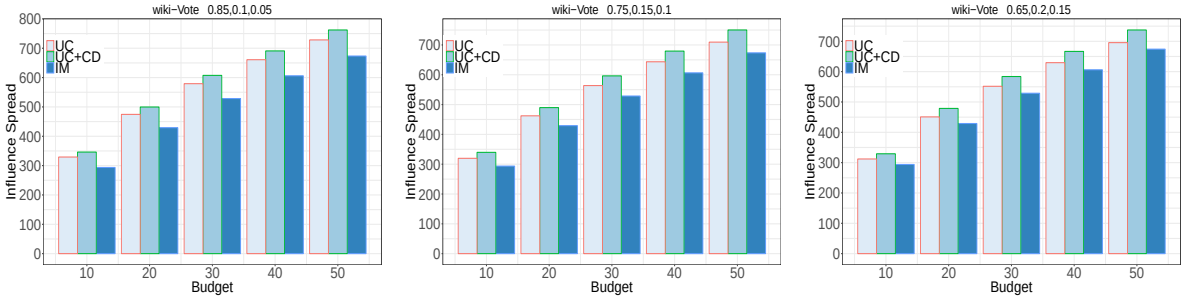
We also tested the effectiveness of setting $\tau = 20$ in the Unified Discount algorithm (UC). Table 4.4 shows the difference of using $\tau = 100$ and $\tau = 20$ as the search step when finding the best unified discount c. Here $P = (0.85, 0.1, 0.05)$. Results under other setting of P are skipped because they are all similar. The column "Reduction Percentage" means how much influence the best spread is decreased if we change τ from 100 to 20. From Table 4.4 we find that the reduction is tiny. In other words, the Unified Discount algorithm is insensitive to this parameter.

4.7.3 Efficiency

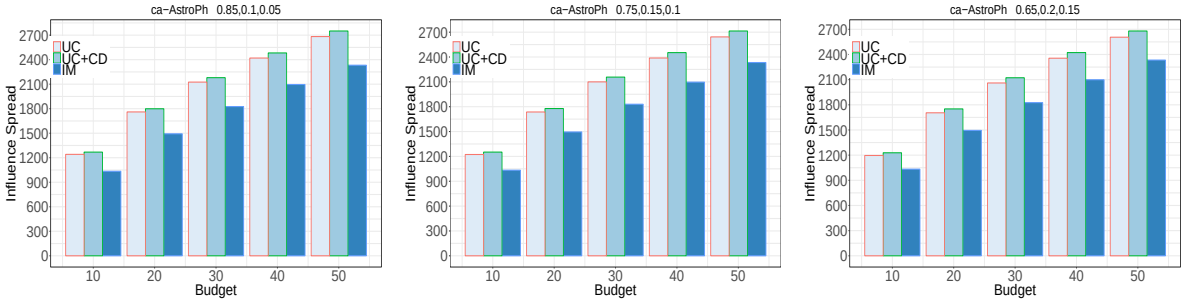
We also tested the efficiency and the scalability of algorithms compared and report the results in Fig. 4.5 and Fig. 4.6. The IM algorithm is always the fastest, since the other two algorithms need to run the IM algorithm (the SSA-fix [82, 51]) first and take the two polling sketches generated by IM as the input to run the Optimization-and-Estimation Strategy (Algorithm 11). However, the running time of different algorithms does not differ much. The most time-consuming algorithm UC+CD only takes around 1.1 to 3 times of the time spent by the most efficient algorithm IM. This is because the two random hyper graphs produced by the IM algorithm are always enough for avoiding the "overfitting" issue in the other CIM algorithms. Thus, every time we broke the while loop in Algorithm 11 in the first round.

4.8 Conclusions

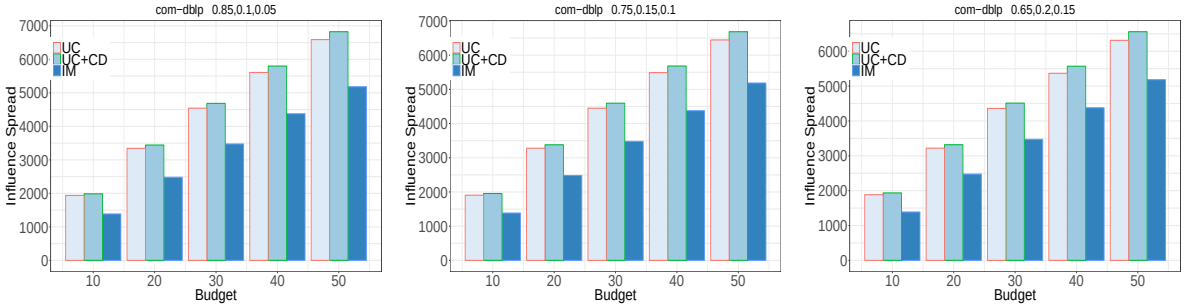
In this chapter, we propose to offer users in social networks discounts rather than free products to trigger social cascades. We model the continuous influence maximization problem. Some key properties of the continuous influence maximization problem are studied and a



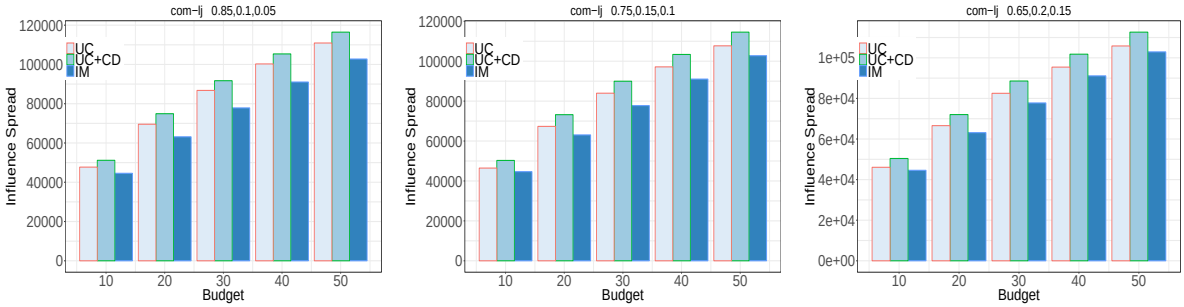
(a) wiki-Vote $P = (0.85, 0.1, 0.05)$ (b) wiki-Vote $P = (0.75, 0.15, 0.1)$ (c) wiki-Vote $P = (0.65, 0.2, 0.15)$



(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$ (e) ca-AstroPh $P = (0.75, 0.15, 0.1)$ (f) ca-AstroPh $P = (0.65, 0.2, 0.15)$



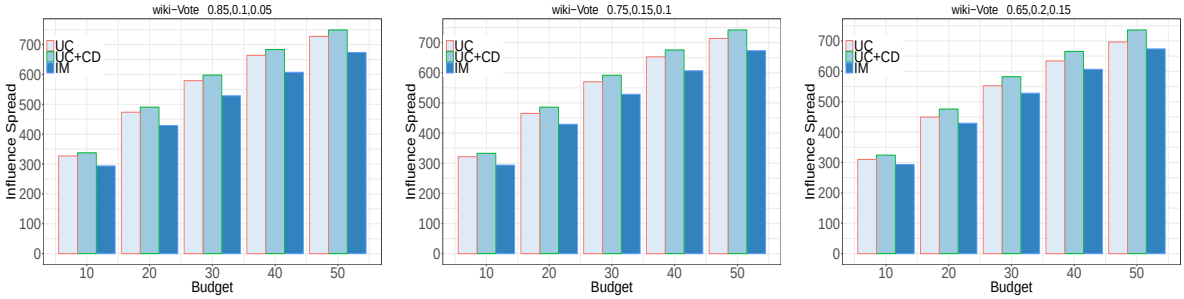
(g) com-dblp $P = (0.85, 0.1, 0.05)$ (h) com-dblp $P = (0.75, 0.15, 0.1)$ (i) com-dblp $P = (0.65, 0.2, 0.15)$



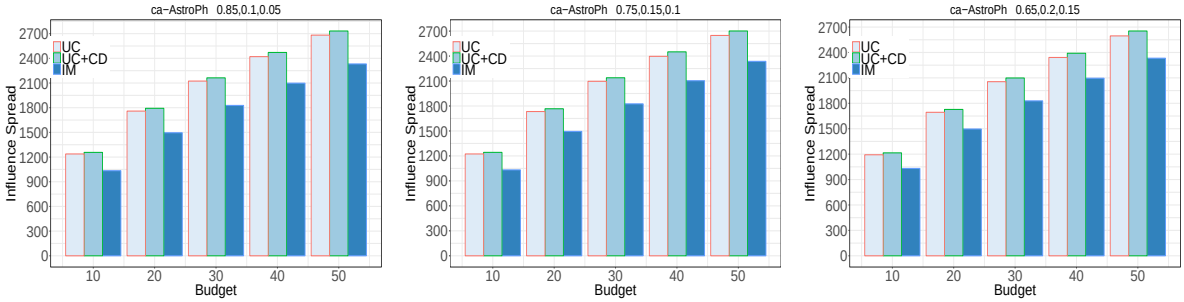
(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$ (k) com-LiveJournal $P = (0.75, 0.15, 0.1)$ (l) com-LiveJournal $P = (0.65, 0.2, 0.15)$

Figure 4.3: Influence spread under curve setting

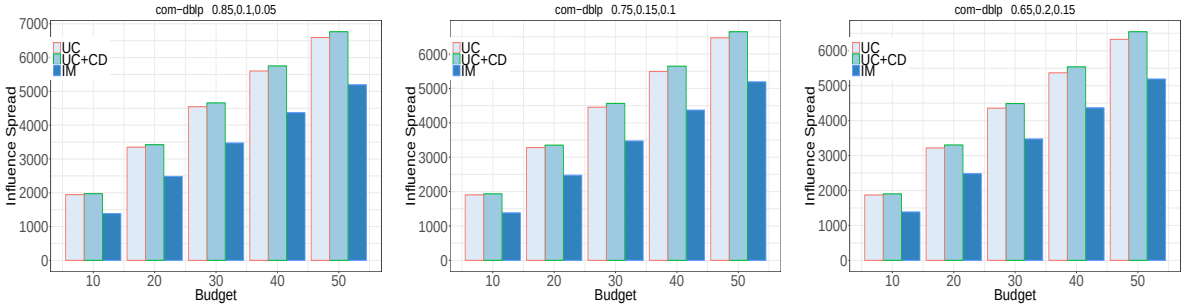
coordinate descent framework is devised. Based on this framework, we prove that under certain conditions the continuous influence maximization problem and the original influence maximization problem share the same optimal solutions. We also demonstrate that there



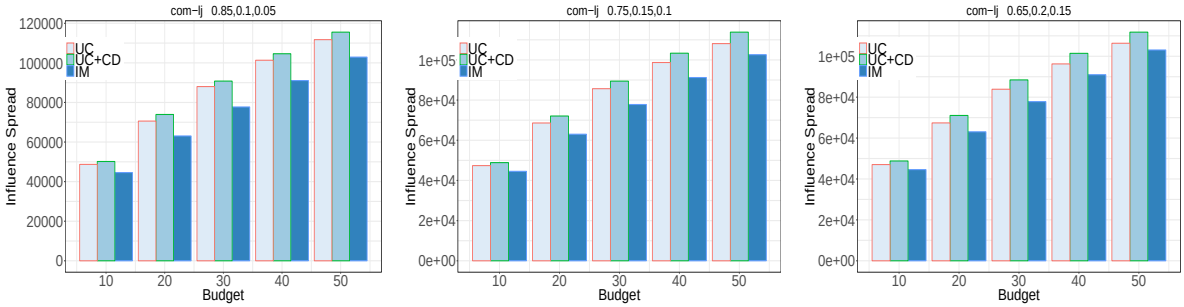
(a) wiki-Vote $P = (0.85, 0.1, 0.05)$ (b) wiki-Vote $P = (0.75, 0.15, 0.1)$ (c) wiki-Vote $P = (0.65, 0.2, 0.15)$



(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$ (e) ca-AstroPh $P = (0.75, 0.15, 0.1)$ (f) ca-AstroPh $P = (0.65, 0.2, 0.15)$



(g) com-dblp $P = (0.85, 0.1, 0.05)$ (h) com-dblp $P = (0.75, 0.15, 0.1)$ (i) com-dblp $P = (0.65, 0.2, 0.15)$



(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$ (k) com-LiveJournal $P = (0.75, 0.15, 0.1)$ (l) com-LiveJournal $P = (0.65, 0.2, 0.15)$

Figure 4.4: Influence spread under pool setting

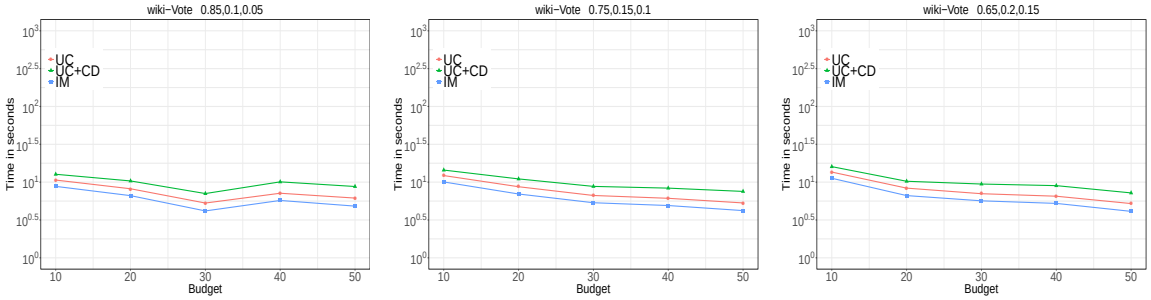
is a data dependent approximation ratio for our solution, where the ratio is decided by the approximation ratio of a traditional influence maximization algorithm (used for finding a good initial configuration) and all seed probability functions. We then develop methods

Dataset	B	$\tau = 100$	$\tau = 20$	Reduction Percentage
Wiki-Vote	10	329	329	0%
	20	475	472	0.6%
	30	579	579	0%
	40	661	653	1.2%
	50	728	718	1.3%
Ca-Astro	10	1241	1241	0%
	20	1760	1760	0%
	30	2120	2120	0%
	40	2422	2422	0%
	50	2680	2680	0%
Com-Dblp	10	1896	1896	0%
	20	3290	3290	0%
	30	4489	4489	0%
	40	5578	5563	0.3%
	50	6557	6530	0.4%
Com-LiveJournal	10	46906	46845	0.1%
	20	69185	69126	0.1%
	30	86469	86469	0%
	40	100027	99948	0.1%
	50	111086	110967	0.1%

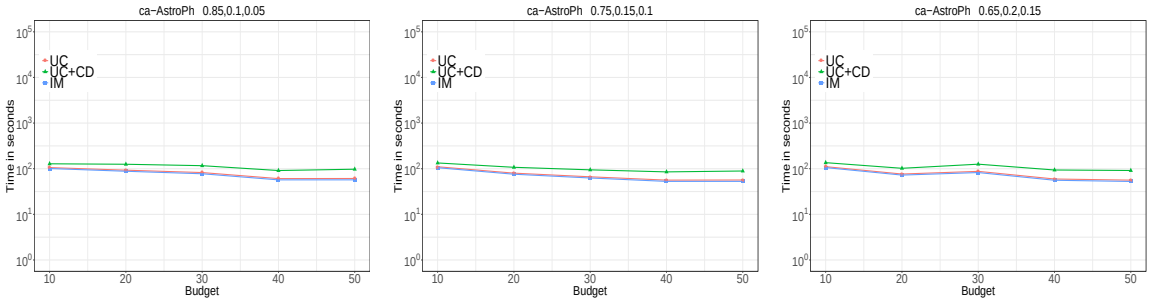
Table 4.4: Effect of the parameter τ in the Unified Discount heuristic (UC).

for implementing the CIM algorithm under triggering models. An analogy between polling-based algorithms and machine learning is discussed. Inspired by this analogy, we point out that there are problems similar to the overfitting issue in machine learning and devise methods to avoid the "overfitting" issue in CIM. The experiment results demonstrate that our methods can improve influence spreads significantly compared to traditional influence maximization, while the extra running time over the baselines is not much.

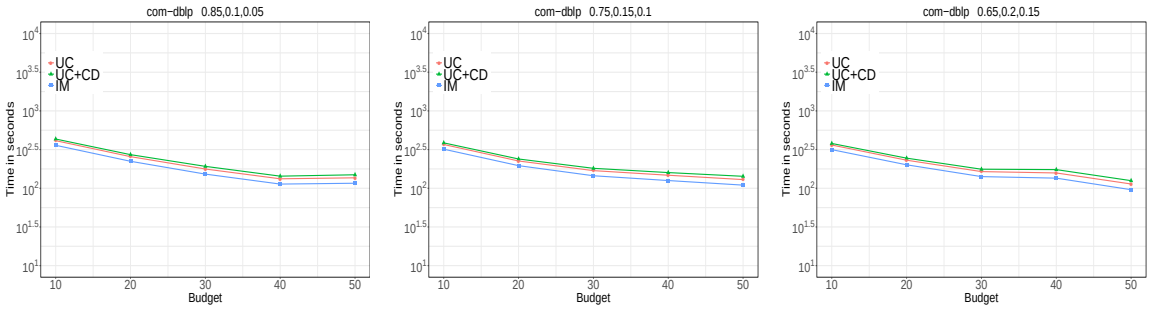
We believe that this work opens a new direction for future work. For example, in the Unified Discount (UC) heuristic for finding a good initial configuration, we conduct a brute force search to find the optimal discount c. Can we devise a better algorithm to search c? Another interesting direction is minimizing the budget of our continuous seeding strategy to cover a given portion of users in a social network. While minimizing budget under integer seeding strategy can be easily obtained by slightly modifying the greedy algorithm for influence maximization, it is far from trivial to design a new algorithm for our continuous seeding strategy.



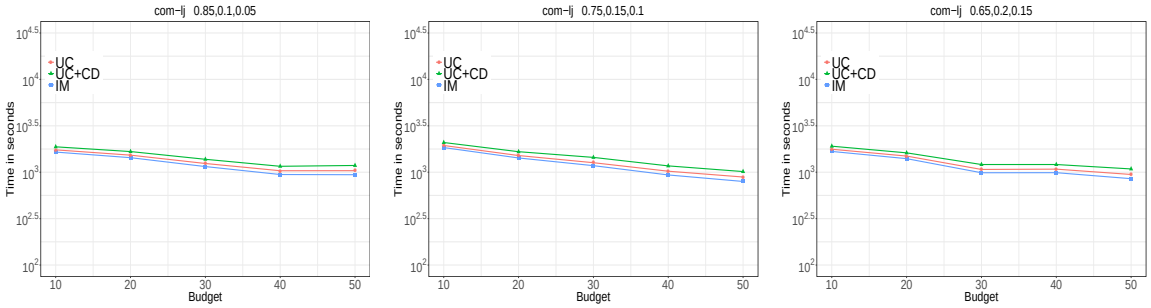
(a) wiki-Vote $P = (0.85, 0.1, 0.05)$ (b) wiki-Vote $P = (0.75, 0.15, 0.1)$ (c) wiki-Vote $P = (0.65, 0.2, 0.15)$



(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$ (e) ca-AstroPh $P = (0.75, 0.15, 0.1)$ (f) ca-AstroPh $P = (0.65, 0.2, 0.15)$

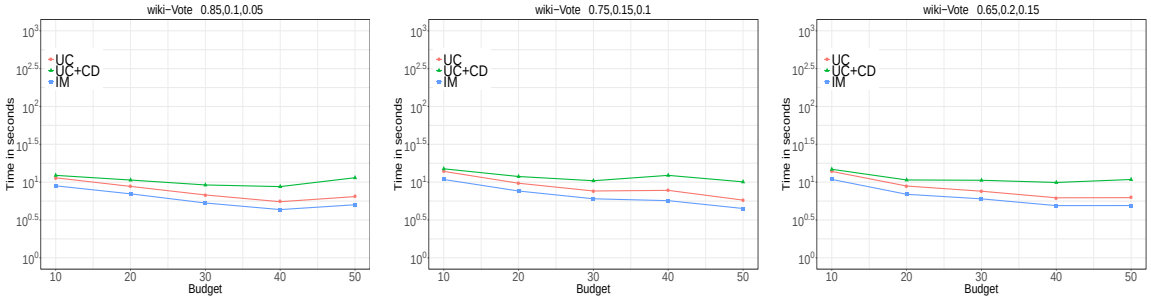


(g) com-dblp $P = (0.85, 0.1, 0.05)$ (h) com-dblp $P = (0.75, 0.15, 0.1)$ (i) com-dblp $P = (0.65, 0.2, 0.15)$

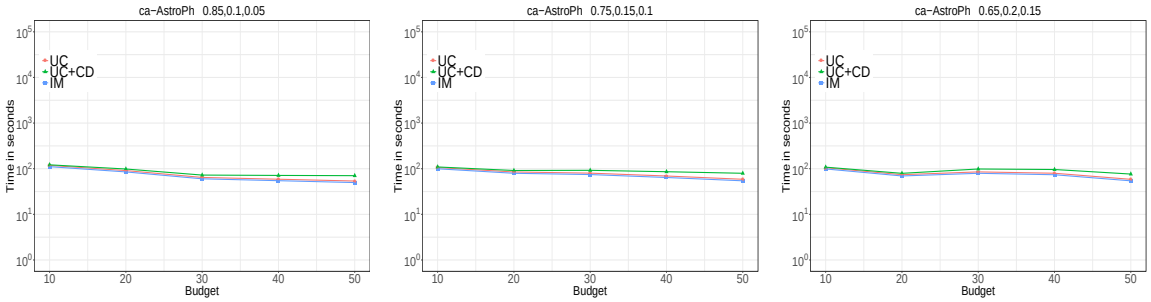


(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$ (k) com-LiveJournal $P = (0.75, 0.15, 0.1)$ (l) com-LiveJournal $P = (0.65, 0.2, 0.15)$

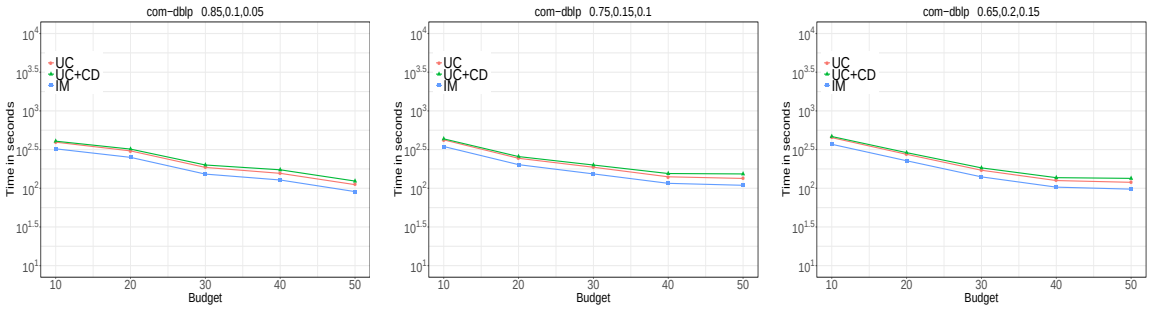
Figure 4.5: Running time under curve setting



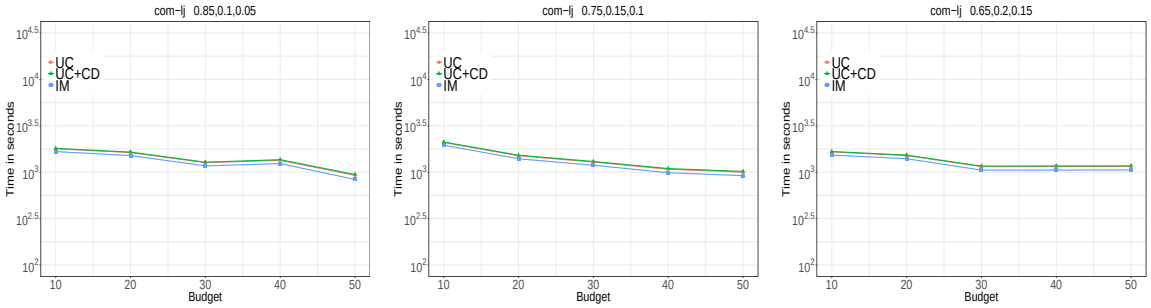
(a) wiki-Vote $P = (0.85, 0.1, 0.05)$ (b) wiki-Vote $P = (0.75, 0.15, 0.1)$ (c) wiki-Vote $P = (0.65, 0.2, 0.15)$



(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$ (e) ca-AstroPh $P = (0.75, 0.15, 0.1)$ (f) ca-AstroPh $P = (0.65, 0.2, 0.15)$



(g) com-dblp $P = (0.85, 0.1, 0.05)$ (h) com-dblp $P = (0.75, 0.15, 0.1)$ (i) com-dblp $P = (0.65, 0.2, 0.15)$



(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$ (k) com-LiveJournal $P = (0.75, 0.15, 0.1)$ (l) com-LiveJournal $P = (0.65, 0.2, 0.15)$

Figure 4.6: Running time under pool setting

Chapter 5

Activity Maximization in Social Networks

In a social network, even about the same information the excitements between different users are different. If we want to spread a piece of new information and maximize the expected total amount of excitements, which seed users should we choose? This problem indeed is substantially different from the renowned influence maximization problem and cannot be tackled using the existing approaches. In this chapter, motivated by the demand in a few interesting applications, we model the novel problem of activity maximization, and tackle the problem systematically. We first analyze the complexity and the approximability of the problem. We develop an upper bound function and a lower bound function that are both submodular so that the Sandwich framework can be applied. We then devise a polling-based randomized algorithm that guarantees a data dependent approximation factor. Our experiments on four real networks clearly verify the effectiveness and scalability of our method, as well as the advantage of our method against the other heuristic methods.

5.1 Introduction

Consider how one can stimulate the discussion about a topic in a social network as much as possible within a budget. Based on messages between users in an instant messaging network, such as Whatsapp and WeChat, one can model topics and strengths/frequencies of interaction activities between users. In some situations, one may want to raise the awareness of a controversial social issue, such as Trump's pulling the US out of Trans-Pacific Partnership (TPP). Within a budget, one wants to spread the information in the network so that people in the network discuss the issue as much as possible. Which users should we choose to start spreading the words?

We model this problem as *activity maximization*. Given a propagation network, which records user interaction activity strength along each edge, we aim to find an optimal set of

seed users under a given budget, such that starting information propagation from the seed users leads to the maximum sum of activity strengths among the influenced users.

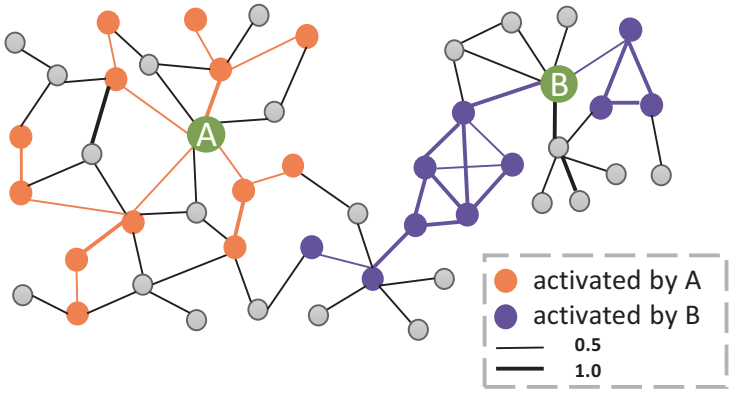


Figure 5.1: A toy example showing the difference between influence maximization and activity maximization.

Isn't this an instance of the well known and well studied influence maximization problem [54]? The answer is "no" indeed. Influence maximization selects a seed set of vertices within a given budget constraint such that the *expected number of vertices* influenced by information diffusion is maximized. However, to satisfy the requirement that "people in the network discuss the issue as much as possible", we not only want to influence many users, but more importantly also want to maximize the expectation of the sum of strengths of the interaction activities between influenced users. Since the activity strength between users differs from user to user, more influenced users do not necessarily lead to more interaction activities. Figure 5.1 shows an example. In the figure, the orange vertices and the blue vertices are activated by seed vertices **A** and **B**, respectively. The thick edges carry an activity strength (i.e., weight) of 1.0 and the thin edges carry a strength of 0.5. Although A can activate more vertices (13) than B (10), the number of edges between the blue vertices as well as the blue vertices and B (i.e., the 15 edges in blue) is more than that between the orange vertices and the orange vertices and B (i.e., the 13 edges in orange). The total activity strength activated by B, 13, is substantially more than the total activity strength activated by A, which is 8.5.

Activity maximization is a novel problem that is substantially different from classic influence maximization. Can we adapt some existing influence maximization methods to solve the activity maximization problem? Unfortunately, the answer is no due to the following two major reasons.

First, the activity maximization problem focuses on the interaction activities between the influenced users. This requires comprehensive consideration of both the information diffusion

dynamics and the diffusion network structure formed by the diffusion process. However, existing influence maximization methods aim to simply maximize the expected number of the active users and seldom take the diffusion network structure into consideration.

Second, at the technical level, the objective functions in the influence maximization problem and the activity maximization problem proposed here have different properties, as to be shown in Section 5.3. Many existing methods for the influence maximization problem rely on some special properties, such as submodurarity and supermodurarity, of the objective function in influence maximization, which unfortunately do not hold for the activity maximization problem.

Motivated by the interesting application scenarios and the technical challenges associated, in this chapter, we propose a novel problem, activity maximization, which aims to maximize the expectation of the total activity among all active users. A unique novel feature of our problem is that the optimization objective captures interactions among active users. We make several contributions.

First, we identify a novel research problem with interesting applications. We propose the novel activity maximization problem that aims to maximize the expectation of the overall activities in a social network. To the best of our knowledge, we are the first to explore the interactions among active vertices in information propagation.

Second, we assess the challenges of the proposed activity maximization problem. We show that the activity maximization problem is NP-hard under the two most popularly used information diffusion models, namely the independent cascade (IC) model and the linear threshold (LT) model. We also prove that computing the activities with respect to a given set of vertices is #P-hard under both the IC model and the LT model. Moreover, we show that the objective function of the problem is neither submodular nor supermodular. The theoretical results clearly show that the proposed activity maximization problem cannot be easily solved using the existing methods for influence maximization. To understand the feasibility of approximate solutions, we appraise the approximability of the problem by constructing a reduction from the densest k-subgraph problem.

Third, to develop practical approximate solutions, we develop a lower bound and an upper bound of activities. We prove that maximizing the lower bound or upper bound is still NP-hard under the IC model and the LT model. Moreover, computing the lower bound or upper bound is still #P-hard under the IC model and the LT model. However, we show the submodularity of the lower bound and the upper bound, which facilitates approximation.

Fourth, we develop a polling based randomized algorithm. We design a sampling method to obtain an unbiased estimation of activities. We also show how to efficiently implement the greedy strategy on the estimate of activities. We extend the sandwich approximation scheme to prove that the proposed algorithm has a data dependent approximation factor.

Last, we verify our algorithm on four real world networks. The experimental results confirm the effectiveness and the efficiency of the proposed algorithm.

Table 5.1: Frequently used notations.

Notation	Description
$G = (V, E, B)$	A social network, where each edge $(u, v) \in E$ is associated with a diffusion model-dependent parameter $B_{u,v}$
$G_S = (V_S, E_S)$	The propagation subgraph induced by seed set S, where V_S is the set of all active vertices and $E_S = \{(u, v) \mid u \in V_S \wedge v \in V_S\}$
$n = \|V\|$	The number of vertices in G
$A_{u,v}$	The interaction strength of edge (u, v)
$\delta_A(S)$	The activity of a given seed set S
$\delta_L(\cdot)$, $\delta_U(\cdot)$	The lower bound and the upper bound respectively
g	A "live-edge" graph instance of G
$g \sim G$	g is sampled from all possible instances of G
$R_g(S)$	The set of vertices reachable from vertex set S in g
g^T	The transpose graph of g: $(u, v) \in g$ iff $(v, u) \in g^T$
$R_{g^T}(v)$	The reverse reachable (RR) set (poll sample) of vertex v
$\mathcal{H}$	The hypergraph consist of hyperedges
$m_{\mathcal{H}}$	The number of the hyperedges in $\mathcal{H}$
$\mathcal{D}(S)$	The degree of the vertex set S in $\mathcal{H}$

The rest of the pager is organized as follows. We formulate the activity maximization problem in Section 5.2. In Section 5.3, we observe several interesting and useful properties of the proposed problem. We develop a lower bound and an upper bound in Section 5.4. In Section 5.5, we devise the polling based algorithm. We report the empirical evaluation results in Section 5.6, and conclude the chapter in Section 5.7. Table 5.1 summarizes the frequently used symbols and their meanings.

5.2 Problem Formulation

In this section, we first review two widely used information diffusion models, and then give the formal statement of the activity maximization problem.

5.2.1 Activity Maximization

The activity maximization problem also considers information diffusion in a social network with an extra parameter A. Each edge $(u, v) \in E$ is associated with an activity strength $A_{u,v}$. Different from diffusion parameter $B_{u,v}$, which indicates how vertex u influences/activates its neighbor v, $A_{u,v}$ captures the interaction strength between u and v when they are both active. The activity strength between a pair of vertices depends on application scenarios, and take any numerical domain. For example, one may learn the activity strength from interaction log data with machine learning methods or simply use some statistical results as activity strength.

Given a social network G, an information diffusion model $\mathcal{M}$, and a seed set S, the diffusion process forms a propagation induced subgraph $G_S = (V_S, E_S)$, where V_S is the set of all active vertices and $E_S = \{(u, v) \in E \mid u \in V_S \wedge v \in V_S\}$ is the set of all edges whose two endpoints are both in V_S. Then, we can define the *activity* of a given seed set S as

$$\delta_A(S) = \mathbb{E}\left[\sum_{(u,v)\in E_S} A_{u,v}\right] \tag{5.1}$$

where $\mathbb{E}[\cdot]$ is the expectation operator. Since information diffusion is a stochastic process, we take the expectation with respect to all possible diffusion instances. The activity measures the overall interaction strength among the active vertices and thus can reflect the overall strength of the activity caused by the information propagated in the social network.

Now, we can formally define the activity maximization problem as follows. Given a social network G, an information diffusion model $\mathcal{M}$, and a budget k, find a seed set S^* such that

$$S^* = \arg \max_{\substack{S \subseteq V \\ |S| = k}} \delta_A(S) \tag{5.2}$$

From the definition, we can see that activity maximization is a discrete optimization problem, just as the traditional influence maximization problem is. Both diffusion parameter B and activity parameter A are inputs to the problem. The activity maximization problem tries to find a set of seed vertices to maximize the activity with given parameter settings. In the next section, we discuss the problem in general. Thus, the solution does no dependent on any specific settings.

5.3 Properties of Activity Maximization

In this section, we first prove the hardness of the activity maximization problem. Then we discuss the properties of the objective function $\delta_A(\cdot)$. Last, we show the approximability of the problem.

5.3.1 Hardness Results

We first assess the hardness of the activity maximization problem.

Theorem 20. *Activity maximization is NP-hard under the IC model and the LT model.*

Proof. We prove by reducing from the set cover problem [53], which is well known in NP-complete. Given a ground set $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ and a collection of sets $\{S_1, S_2, \ldots, S_m\}$ whose union equals the ground set, the set cover problem is to decide if there exist k sets in $\mathcal{S}$ so that the union equals $\mathcal{U}$.

Given an instance of the set cover problem, we construct a corresponding graph with $2n + m$ vertices as follows. We create a vertex x_i for each set S_i, two vertices y_j and y'_j for each element u_j, and two edges (x_i, y_j) and (x_i, y'_j) with propagation probability 1 for the IC model and with influence weight 1 for the LT model and activity 0 if $u_j \in S_i$. We also create an edge between y_j and y'_j with propagation probability 0 and activity 1 for each element u_j. The information diffusion will be a deterministic process, since all propagation probabilities are either 1 or 0. Therefore, the set cover problem is equivalent to deciding if there is a set S of k vertices such that $\delta_A(S) = n$. The theorem follows immediately. □

Activity maximization is NP-hard. Then, what is the hardness of computing the activity with respect to a given seed set S?

Theorem 21. *Given a seed set S, computing $\delta_A(S)$ is #P-hard under the IC model and the LT model.*

Proof. We prove by reducing from the influence spread computation problem, which was proved #P-hard under the IC model and the LT model [19, 21].

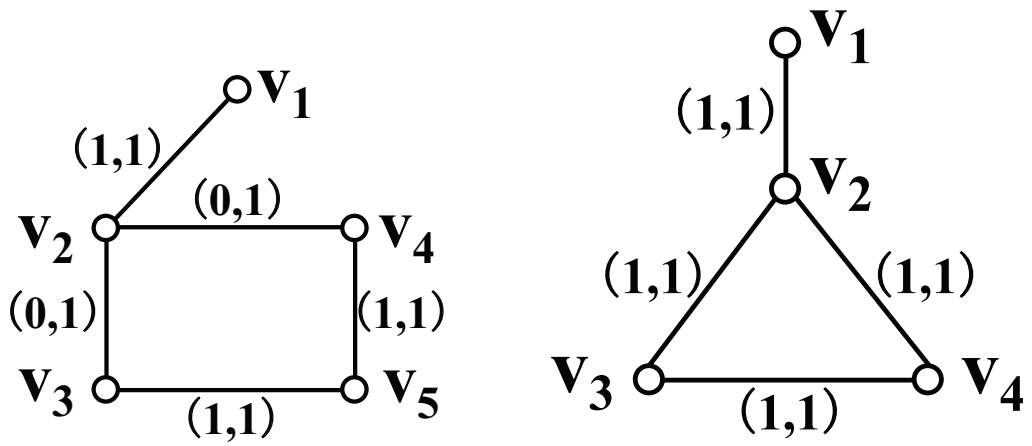
Given an instance of the influence spread computation problem, we keep the same graph G and influence diffusion parameters B. We set $A_{u,v} = 1$ for any $u, v \in V$ and compute $x_1 = \delta_A(S)$ in the graph G. Next, we add a new vertex v' for each vertex v in the graph G and an edge between v and v' with propagation probability 1 for the IC model and with influence weight 1 for the LT model and activity 1. Now, we obtain a new graph G' and can compute $x_2 = \delta_A(S)$ in the graph G'. For any newly added vertex v', the only way to be activated is through its only neighbor v. Moreover, a newly added vertex v' will be activated if its neighbor v is active, since the propagation probability of the newly added edges is 1. Thus, $x_2 - x_1$ is exactly the influence spread in the graph G. The theorem follows immediately. □

In [54], Kempe *et al.* introduced the triggering model that generalizes the IC model and the LT model. In the triggering model, each vertex v independently chooses a subset of its neighbors as its "triggering set" according to some distribution. A vertex will be activated if at least one vertex of its triggering set is active. We can see that the reduction we construct in the proof of Theorem 21 still holds for the triggering model. Thus, we have the following result.

Corollary 21.1. *Given a seed set S, computing $\delta_A(S)$ is #P-hard in any triggering model $\mathcal{M}$ if computing influence spread is #P-hard in $\mathcal{M}$.*

5.3.2 Modularity of Objective Functions

The objective function of influence maximization is submodular under the IC model and the LT model. Unfortunately, the objective function in activity maximization is not submodular. Moreover, we can show that $\delta_A(\cdot)$ is not supermodular as well.



(a) Counter example 1 (b) Counter example 2

Figure 5.2: Counter examples

Fact 2. *$\delta_A(\cdot)$ is not submodular under the IC model and the LT model.*

Proof. We prove by a counter example. Consider Figure 5.2(a). The first number in the tuple on each edge represents the propagation probability for the IC model and the influence weight for the LT model. The second number is the activity of the edge. For example, in the counter example 1, $(1, 1)$ on edge (v_1, v_2) means $B_{v_1,v_2} = 1$ and $A_{v_1,v_2} = 1$. In this example, we have $\delta_A(\{v_1\}) = 1$, $\delta_A(\{v_1, v_5\}) = 5$ and $\delta_A(\{v_5\}) = 2$. That is,

$$\delta_A(\{v_1\}) - \delta_A(\emptyset) < \delta_A(\{v_1, v_5\}) - \delta_A(\{v_5\})$$

Therefore, $\delta_A(\cdot)$ is not submodular. □

From the counter example in the proof of Fact 2 (Figure 5.2(a)), we can see that the reason why $\delta_A(\cdot)$ is not submodular is the "combination effect" between the newly added vertex and the existing seed vertices. For example, If we add v_1 into S when $S = \emptyset$, then there is only one active endpoint for edge (v_2, v_4) and (v_2, v_3), that is v_2. But if we add v_1 to S when $S = \{v_5\}$, then both the two endpoints of edge (v_2, v_4) and (v_2, v_3) are active, since v_3 and v_4 are activated by v_5. The "combination effect" has its roots in the definition of activity. We only count the activity on the edges whose two endpoints are both active. As a result, the newly added vertex and the existing seed vertices may activate the two endpoints of an edge together, which leads to a violation of submodularity.

Fact 3. *$\delta_A(\cdot)$ is not supermodular under the IC model and the LT model.*

Proof. Again, we prove by a counter example. Consider the counter example 2 in Fig 5.2(b), we have $\delta_A(\{v_2\}) = 4$, $\delta_A(\{v_1, v_2\}) = 4$ and $\delta_A(\{v_1\}) = 4$. Thus,

$$\delta_A(\{v_2\}) - \delta_A(\emptyset) > \delta_A(\{v_2, v_1\}) - \delta_A(\{v_1\})$$

That is, $\delta_A(\cdot)$ is not supermodular. □

From the counter example in the proof of Fact 3 (Figure 5.2(b)), we can see that the reason why $\delta_A(\cdot)$ is not supermodular is the "overlap effect" between the newly added vertex and the existing seed vertices. The vertices that the newly added vertex can activate may have already been activated by the existing seed vertices, which means that adding a new vertex does not bring any marginal gain.

5.3.3 Approximability

Since $\delta_A(\cdot)$ is neither submodular nor supermodular, we cannot adopt the standard procedure for optimizing submodular function or supermodular function to get an approximation solution. To explore the approximability of the activity maximization problem, we explore the connection between the activity maximization problem and the densest k-subgraph extraction problem.

Theorem 22. *If there exists a polynomial time algorithm approximating the activity maximization problem within a ratio of α, then there exists a polynomial time algorithm that can approximate the densest k-subgraph problem within a ratio of α.*

Proof. We prove by constructing a reduction from the densest k-subgraph problem to the activity maximization problem. Given a graph and an integer k, the densest k-subgraph problem is to find a subgraph of exactly k vertices that has the maximum density. For a subgraph $G_S = (V_S, E_S)$, the density is defined as $\frac{|E_S|}{|V_S|}$.

Given an instance of the densest k-subgraph problem, we construct a corresponding instance of the activity maximization problem. We keep the same graph and set $B_{u,v} = 0$ and $A_{u,v} = 1$ for $u, v \in V$. Then, the activity maximization problem is to find a set of k vertices and maximize the number of edges whose both endpoints are in this set. It is equivalent to maximizing the density since the number of vertices is constant. □

Khot [56] showed that the densest k-subgraph problem does not admit PTAS[1] (Polynomial Time Approximation Scheme [105]) assuming $NP \nsubseteq \bigcap_{\epsilon>0} BPTIME(2^{n^\epsilon})$, we immediately have the following result.

[1]A PTAS is an algorithm that returns a solution within a factor 1 + ϵ of being optimal (or 1 - ϵ for maximization problems) in polynomial time for any $\epsilon > 0$.

Corollary 22.1. *There is no PTAS for the activity maximization problem assuming* $NP \nsubseteq \bigcap_{\epsilon>0} BPTIME(2^{n^\epsilon})$.

In fact, finding a good approximation to the densest k-subgraph problem is challenging. The current best approximation ratio of $n^{1/4+\epsilon}$ for $\epsilon > 0$ was achieved by Bhaskara *et al.* [8]. It is still unknown if there exists a polynomial time algorithm that can approximate the densest k-subgraph problem with a constant factor.

5.4 Lower Bound and Upper Bound

In this section, we first give a lower bound and an upper bound on activities. Then we discuss the properties of the lower bound and the upper bound.

5.4.1 The Bounds

Since the "combination effect" among seed vertices comprises the submodularity of the objective function $\delta_A(\cdot)$, we try to develop a lower bound of $\delta_A(\cdot)$ that is submodular by ignoring the "combination effect". The major idea is that we only consider the edges whose two endpoints are activated by the same seed vertex. Accordingly, the lower bound can be defined as

$$\delta_L(S) = \mathbb{E}[\sum_{(u,v)\in \bigcup_{x\in S} E_{\{x\}}} A_{u,v}] \tag{5.3}$$

where $E_{\{x\}}$ is the set of edges of the propagation subgraph induced by seed set $\{x\}$. Recall that the propagation subgraph induced by a seed set consists of the vertices that can be activated by the seed set. Here, the seed set consists of only one vertex x. It is easy to see that $\delta_L(S) \leq \delta_A(S)$ for any $S \subseteq V$, since we ignore the edges whose endpoints are activated by different seed vertices.

A straightforward way to get an upper bound is to consider all the edges that have at least one active endpoint. In this way, the upper bound equals to the activity of edges that have one active endpoint plus the activity of edges whose two endpoints are both active. The latter is exactly the activity we want to compute. Here, we present a tighter upper bound from the perspective of active vertices, which can be defined as

$$\delta_U(S) = \mathbb{E}[\sum_{v\in V_S} w(v)] \tag{5.4}$$

where

$$w(v) = \frac{1}{2} \sum_{u\in N(v)} A_{u,v}.$$

Given a seed set S, $\delta_U(S)$ equals to the half of the activity of edges that have one active endpoint plus the activity of edges whose two endpoints are both active. Thus, $\delta_U(S)$ is

better than the straightforward one. Also, we can see that the upper bound is essentially a weighted version of the influence spread, where the weight of vertex v is $\frac{1}{2}\sum_{u\in N(v)} A_{u,v}$. For the influence spread, $w(v) = 1$ for each vertex v.

5.4.2 Properties of the Bounds

Using the lower bound and the upper bound, we can approximate the information activity problem by maximizing the lower bound and the upper bound [69]. However, maximizing the lower bound and the upper bound is still NP-hard.

Theorem 23. *Maximizing the lower bound is NP-hard under the IC model and the LT model.*

Proof. We prove by reducing from the NP-complete set cover problem [53]. We show the reduction constructed in the proof of Theorem 20 still holds for the lower bound. The lower bound only considers the edges whose two endpoints can be activated by the same seed vertex. In the previous reduction, for all the edges whose activity is not equal to 0 (the edges between y_j and y'_j), their two endpoints can be activated by the same vertex. Thus, the set cover problem can be solved by deciding if there is a set S of k vertices such that $\delta_L(S) = n$. □

Theorem 24. *Maximizing the upper bound is NP-hard under the IC model and the LT model.*

Proof. We prove by reducing from the NP-hard influence maximization problem [54].

Given an instance of the influence maximization problem, let d_{max} be the highest degree of the vertices in the graph G. Then, for each vertex v in G, we add $N_d = d_{max} - d_v$ new vertices, $v'_1, v'_2, \ldots, v'_{N_d}$, and N_d new edges, $(v, v'_1), (v, v'_2), \ldots, (v, v'_{N_d})$. Now we obtain a new graph G'. We set the propagation probability of the newly added edges to 0 for the IC model, and set the influence weight of the newly added edges to 0 for the LT model, and set the information activity of all the edges in G' to $\frac{2}{d_{max}}$.

Then, we have $\forall v \in V$, $w(v) = 1$, and $\forall v' \in V' \setminus V$, $w(v') = \frac{2}{d_{max}}$. Since the propagation probability of all newly added edges is 0, the newly added vertices can never be activated. Therefore, we have $I^G(S) = \delta_U^{G'}(S), \forall S \subseteq V$, where $I(S)$ is the influence spread of a give seed set S in G and $\delta_U^{G'}(S)$ is the upper bound in G'.

Next, we prove that $S_U^* = \operatorname{argmax} \delta_U^{G'}(S)$ does not contain any newly added vertices. If there is any newly added vertex in S_U^*, we can always replace it with a vertex in $V \setminus S_U^*$ and increase the value of the objective function. Thus, if S_U^* is the optimal solution of maximizing the upper bound in G', it must be the optimal solution to the influence maximization problem in G. □

Although maximizing the lower bound and the upper bound is NP-hard, the objective functions of the lower bound and the upper bound are submodular.

Theorem 25. *$\delta_L(\cdot)$ is submodular under the IC model and the LT model.*

Proof. Given a graph G and an influence diffusion model, either the IC model or the LT model, we can construct "live-edge" graphs for G using the methods proposed in [54]. Let g be a "live-edge" graph instance. Denote by $Pr(g)$ the probability that g is selected from all possible instances. Let $E_g(S)$ be the set of edges whose two endpoints can be reachable from the same vertex in the seed set S. Then we can rewrite $\delta_L(S)$ to

$$\delta_L(S) = \sum_{g \sim G} Pr(g) \sum_{(u,v) \in E_g(S)} A_{u,v}$$

We only need to prove $Q(S) = \sum_{(u,v) \in E_g(S)} A_{u,v}$ is submodular for any "live-edge" graph instance g, since an non-negative linear combination of submodular functions is also submodular.

To prove, let M and N be two sets such that $M \subseteq N \subseteq V$. For any $v \in V \setminus N$, consider the difference between $Q(M \cup \{v\})$ and $Q(M)$. It must be contributed from the edges whose two endpoints can be reachable from v but cannot be reachable from the vertices in M. These edges must be a super set of the edges whose two endpoints can be reachable from v but cannot be reachable from the vertices in N, since $M \subseteq N$. It follows that $Q(M \cup \{v\}) - Q(M) \geq Q(N \cup \{v\}) - Q(N)$. Therefore, $Q(S)$ is submodular and the theorem follows. □

Theorem 26. *$\delta_U(\cdot)$ is submodular under the IC model and the LT model.*

Proof. We can prove the theorem by the same "live-edge" technique used in the proof of Theorem 25. Let $R_g(S)$ be the set of vertices reachable from S in g. Then, $\delta_U(S)$ can be rewritten to

$$\delta_U(S) = \sum_{g \sim G} Pr(g) \sum_{v \in R_g(S)} w(v)$$

The way to prove that $Q'(S) = \sum_{v \in R_g(S)} w(v)$ is submodular is similar to the proof of $Q(S)$ in Theorem 25. The vertices that can be reachable from v but cannot be reachable from the vertices in M must be a super set of the vertices that can be reachable from v but cannot be reachable from the vertices in N. It follows that $Q'(M \cup \{v\}) - Q'(M) \geq Q'(N \cup \{v\}) - Q'(N)$. Therefore, $Q'(S)$ is submodular and the theorem follows. □

Theorems 25 and 26 are good news. With the submodularity we can adopt the standard procedure for optimizing submodular functions to obtain an approximation solution [78]. One challenge remains. Applying the algorithm proposed in [78] requires evaluating the lower bound and the upper bound. However, computing the lower bound and the upper bound with respect to a given seed set is unfortunately #P-hard.

Theorem 27. *Given a seed set S, computing $\delta_L(S)$ is #P-hard under the IC and the LT model.*

Proof. We prove by reducing from the influence spread computation problem. We show that the reduction we construct in the proof of Theorem 21 still holds for the lower bound case. Let $y_1 = \delta_L(S)$ in the graph G and $y_2 = \delta_L(S)$ in the graph G'. Since the propagation probability of the edge (v, v') is 1 for the IC model and the influence weight of the edge (v, v') is 1 for the LT model, v and v' can be activated by the same seed vertex. It follows that $y_2 - y_1$ is also the influence spread in the graph G. □

Theorem 28. *Given a seed set S, computing $\delta_U(S)$ is #P-hard under the IC and the LT model.*

Proof. We prove by reducing from the influence spread computation problem. The reduction is the same as the one in the proof of Theorem 24. We already showed $I^G(S) = \delta_U^{G'}(S)$ for any seed set $S \subseteq V$. Therefore, the theorem follows immediately. □

Since the activity, the lower bound and the upper bound are all #P-hard to compute, we will discuss how to estimate them in the next section.

5.5 A Polling Based Method

To solve the activity maximization problem, we design a polling based method similar to the one introduced in Chapter 2 for the influence maximization problem. The framework of our method includes two steps. In the first step, it estimates the activity, the lower bound and the upper bound through sampling. In the second step, it finds a solution for maximizing the estimated activity based on estimations of the activity, the lower bound and the upper bound. We prove that we can bound the estimation error and our solution enjoys a data-dependent approximation guarantee for the activity maximization problem.

5.5.1 Estimation

In a social network G, given an information diffusion model, either the IC model or the LT model, and a seed set S, let g be a "live-edge" graph instance of G and $R_g(S)$ be the set of vertices reachable from S in g. Denote by $R_{g^T}(v)$ the reverse reachable (RR) set [104] (poll sample) for vertex v in g, where g^T is the transpose graph [10] of g: $(u, v) \in g$ iff $(v, u) \in g^T$. We write $(u, v) \sim E$ to indicate that we randomly pick (u, v) from E as a sample according to a certain distribution. The meaning of $v \sim V$ is similar.

To estimate the activity, we first have the following result.

Theorem 29. *For any seed set $S \subseteq V$,*

$$\delta_A(S) = T \cdot \Pr_{g \sim G, (u,v) \sim E}\Big[S \cap R_{g^T}(u) \neq \emptyset \wedge S \cap R_{g^T}(v) \neq \emptyset\Big],$$

where $T = \sum_{(u,v) \in E} A_{u,v}$.

Proof.

$$
\begin{aligned}
\delta_A(S) &= \mathbb{E}\left[\sum_{(u,v)\in E_S} A_{u,v}\right] \\
&= \sum_{(u,v)\in E} Pr\left[(u,v)\in E_S\right] A_{u,v} \\
&= \sum_{(u,v)\in E} \Pr_{g\sim G}\left[u\in R_g(S)\wedge v\in R_g(S)\right] A_{u,v} \\
&= \sum_{(u,v)\in E} \Pr_{g\sim G}\left[\exists w_1,w_2\in S, w_1\in R_{g^T}(u)\wedge w_2\in R_{g^T}(v)\right] A_{u,v} \\
&= T\cdot \sum_{(u,v)\in E} \Pr_{g\sim G}\left[\exists w_1,w_2\in S, w_1\in R_{g^T}(u)\wedge w_2\in R_{g^T}(v)\right] \frac{A_{u,v}}{T} \\
&= T\cdot \Pr_{g\sim G,(u,v)\sim E}\left[\exists w_1,w_2\in S, w_1\in R_{g^T}(u)\wedge w_2\in R_{g^T}(v)\right] \\
&= T\cdot \Pr_{g\sim G,(u,v)\sim E}\left[S\cap R_{g^T}(u)\neq\emptyset\wedge S\cap R_{g^T}(v)\neq\emptyset\right]
\end{aligned}
$$

The 5th line is the expected probability with respect to the activity distribution of edges, where the probability for edge (u,v) is $\frac{A_{u,v}}{T}$. □

The intuition of Theorem 29 is that if a seed set S has a high activity value, then the probability that S is simultaneously reachable from both two endpoints of a randomly picked edge in a randomly picked "live-edge" graph instance is high, since we only count the activity on the edges whose two endpoints are both active. Theorem 29 implies that we can estimate $\delta_A(S)$ by estimating the probability of the event $S\cap R_{g^T}(u)\neq\emptyset\wedge S\cap R_{g^T}(v)\neq\emptyset$. To achieve the estimation, we conduct a poll as follows. We select an edge (u,v) with probability $\frac{A_{u,v}}{T}$, and run Monte Carlo simulation of the "live-edge" process. During the process, we record all the vertices that can reach u and v through "live" edges. Algorithm 12 summarizes the process.

One critical observation is that we do not need to conduct the "live-edge" process on the entire graph. Instead, we can simulate the process starting from u and v, respectively. We only need to make sure that each edge is marked consistently as the same status ("live" or "blocked") in these two simulations. We call the pair of two RR sets obtained from a poll a hyperedge. All the generated hyperedges constitute a hypergraph $\mathcal{H}$.

Denote by m_H the number of the hyperedges in $\mathcal{H}$. If a vertex v appears in both RR sets of a hyperedge $\mathcal{E}$, $\mathcal{E}$ is said to be *fully covered* by v. If a vertex v only appears in one of the two RR sets of a hyperedge $\mathcal{E}$, $\mathcal{E}$ is said to be *partially covered* by v. Denote by $\mathcal{D}(S)$ the degree of the set of vertices S, which is the number of hyperedges in $\mathcal{H}$ that can be fully covered by S. According to Theorem 29, $T\cdot\frac{\mathcal{D}(S)}{m_H}$ is an unbiased estimator of $\delta_A(S)$

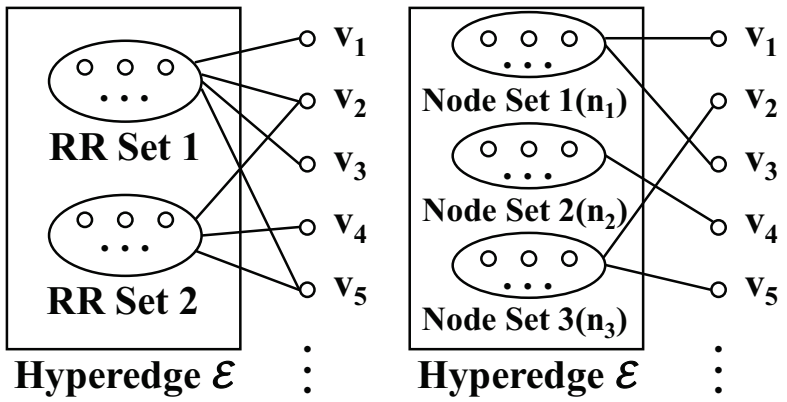


Figure 5.3: Hyperedge for activities

Algorithm 12 Generate Hyperedges

Input: Social network $G = (V, E, B)$, A and diffusion model $\mathcal{M}$
Output: A hyperedge $\mathcal{E}$

1: Initialize $\mathcal{E} = (\emptyset, \emptyset)$
2: Pick an edge (u, v) with probability $\frac{A_{u,v}}{T}$.
3: Generate a "live-edge" graph g according to $\mathcal{M}$
4: Let $N_1 = R_{g^T}(u)$ and $N_2 = R_{g^T}(v)$
5: Let $\mathcal{E} = (N_1, N_2)$
6: **return** $\mathcal{E}$

for any fixed m_H. Please note that there also exists "combination effect" between vertices in this case. For example, in the left part of Figure 5.3, v_1 only appears in the first RR set of hyperedge $\mathcal{E}$ and v_4 only appears in the second RR set. v_1 and v_4, respectively, partially covers $\mathcal{E}$. But $\mathcal{E}$ is fully covered by the combination of v_1 and v_4. Thus, similar to $\delta_A(\cdot)$, $\mathcal{D}(\cdot)$ is not submodular neither.

Similarly, for the lower bound and the upper bound, we have the following two results.

Theorem 30. *For any seed set $S \subseteq V$,*

$$\delta_L(S) = T \cdot \Pr_{g\sim G,(u,v)\sim E}\Big[S \cap (R_{g^T}(u) \cap R_{g^T}(v)) \neq \emptyset\Big],$$

where $T = \sum_{(u,v)\in E} A_{u,v}$.

Proof. The lower bound only considers the edges whose two endpoints can be activated by the same seed vertex. Thus, to prove the theorem, we only need to let $w_1 = w_2$ in the proof of Theorem 29, that is

$$\begin{aligned}\delta_L(S) =& T \cdot \Pr_{g\sim G,(u,v)\sim E}\Big[\exists w \in S, w \in R_{g^T}(u) \wedge w \in R_{g^T}(v)\Big] \\ =& T \cdot \Pr_{g\sim G,(u,v)\sim E}\Big[S \cap (R_{g^T}(u) \cap R_{g^T}(v)) \neq \emptyset\Big]\end{aligned}$$

□

Using Theorem 30, we can estimate the lower bound using essentially the same sampling process as the activity. The only difference is that there is only one vertex set in the hyperedge for the lower bound, that is $N_1 \cap N_2$. In this case, a hyperedge $\mathcal{E}$ is covered by vertex v if and only if $v \in N_1 \cap N_2$.

Theorem 31. *For any seed set $S \subseteq V$,*

$$\delta_U(S) = W \cdot \Pr_{g \sim G, v \sim V}\Big[S \cap R_{g^T}(v) \neq \emptyset\Big],$$

where $W = \sum_{v \in V} w(v)$.

Proof. The upper bound is essentially a weighted variation of the influence spread. Thus, we can apply the proof proposed in [81]. □

There is also only one vertex set in the hyperedge for the upper bound. We can generate the hyperedge using the sampling method proposed in [81].

Since we can estimate the objective function ($\delta_A(\cdot)$, $\delta_L(\cdot)$ or $\delta_U(\cdot)$) by the degrees of the set of vertices, we can regard $\mathcal{H}$ as encoding an approximation to the objective function. With the estimate of the objective function, we go to the second step of the polling based framework, that is, maximizing the estimate. To achieve this goal, we adopt the simple but powerful greedy strategy, which picks the vertex with the largest marginal gain (the increase of degree in $\mathcal{H}$ for our case) iteratively. Next, we show how to efficiently implement a greedy strategy on the hypergraph.

5.5.2 Efficient Implementation of the Greedy Strategy

For the lower bound and the upper bound, there is only one vertex set in each hyperedge. Thus, we can use the standard greedy algorithm for maximum coverage problem to obtain an approximate solution [104]. However, there are two vertex sets in the hyperedge for the activity maximization problem. A hyperedge can be fully or partially covered by a vertex or a vertex set. Thus, we cannot directly apply the greedy strategy. To tackle this issue, here we discuss how to efficiently implement the greedy strategy on the hypergraph.

First, we store the original hyperedges of two RR sets in a more efficient manner. There are three sets, n_1, n_2 and n_3 for each hyperedge $\mathcal{E}$, where n_1 and n_2 are the sets of vertices that can only cover the first and second RR set of $\mathcal{E}$, respectively, and n_3 is the set of vertices that can cover both two RR sets of $\mathcal{E}$. Figure 5.3 illustrates the idea.

Then, we build an inverted index for each vertex. There are three sets, e_1, e_2 and e_3 for each vertex v, where e_1 and e_2 are the sets of hyperedges whose first and second RR set can be covered by v, respectively, and e_3 is the set of hyperedges that can be fully covered by v.

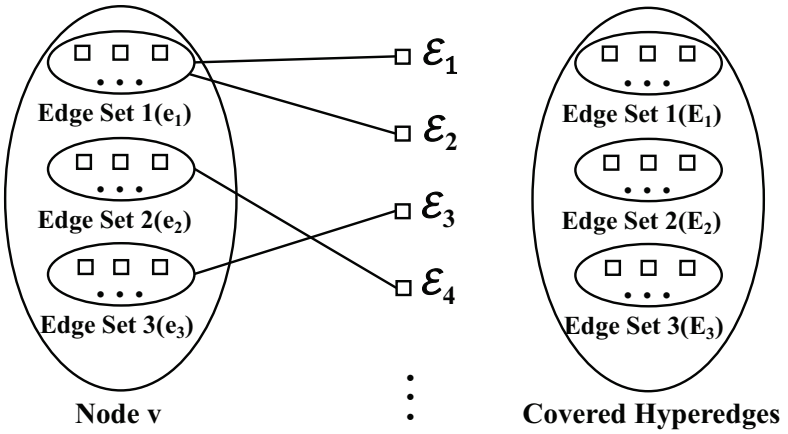


Figure 5.4: Data Structures

Third, we maintain a global data structure to record the current covered hyperedges. There are also three sets, E_1, E_2 and E_3, in this data structure, where E_1 and E_2 are the sets of hyperedges whose first and second RR sets have been covered, respectively, and E_3 is the set of hyperedges that have been fully covered. Figure 5.4 shows these two data structures. With these data structures, we have the following fact.

Fact 4. *Given a seed set S, for each vertex $v \in V \setminus S$, the marginal gain $\mathcal{D}(S \cup \{v\}) - \mathcal{D}(S)$ is*

$$MG(v) = |v.e_3 \setminus E_3| + |v.e_1 \cap E_2| + |v.e_2 \cap E_1| \tag{5.5}$$

RATIONALE. If we add a vertex v to the current seed set S, the newly covered hyperedges can be divided into two groups. The first group is the hyperedges that can be covered by v alone but not covered by S, that is $v.e_3 \setminus E_3$. The second group is the hyperedges that are partially covered by S and are fully covered if v is added to S, that is $v.e_1 \cap E_2$ and $v.e_2 \cap E_1$. ∎

Fact 4 implies that we can pick the vertex with the largest marginal gain in each iteration and then incrementally update the marginal gains of the rest vertices. Algorithm 13 describes the details.

Here, we briefly explain how to incrementally update the marginal gain. Assuming E_1, E_2 and E_3 are updated to E_1', E_2', and E_3', respectively, we update the marginal gains as follows. For each hyperedge $\mathcal{E} \in E_1' \setminus E_1$, we increase the marginal gains of the vertices in $\mathcal{E}.n_2$ by 1. For each hyperedge $\mathcal{E} \in E_2' \setminus E_2$, we increase the marginal gains of the vertices in $\mathcal{E}.n_1$ by 1. For each hyperedge $\mathcal{E} \in E_3' \setminus E_3$, we first decrease the marginal gains of the vertices in $\mathcal{E}.n_3$ by 1. Then, we decrease the marginal gains of the vertices in $\mathcal{E}.n_2$ by 1 if $\mathcal{E} \in E_1$, and decrease the marginal gains of the vertices in $\mathcal{E}.n_1$ by 1 if $\mathcal{E} \in E_2$.

Now, the only remaining question is to decide how many hyperedges we need to sample, which will be addressed next.

Algorithm 13 Maximum Coverage on Hypergraph

Input: Social network G, Hypergraph $\mathcal{H}$ and budget k
Output: Seed set S

```
Initialize S = E_1 = E_2 = E_3 = ∅
for v ∈ V do
    MG(v) = |v.e|
end for
while |S| < k do
    v = argmax_{u∈V\S} MG(u)
    S = S ∪ {v}
    update E_1, E_2, and E_3
    for u ∈ V \ S do
        update MG(u) according to Eq. 5.5
    end for
end while
return S
```

5.5.3 Sample Complexity

In this subsection, we discuss how to use a sample of proper size to restrict the estimate error of the activity, the lower bound and the upper bound. With the technique, we show that the polling algorithm can provide an approximate solution to maximizing the lower bound and the upper bound.

To bound the estimate error of the polling method, we use the Stopping Rule Theorem (Corollary 0.2) in Section 2.1.4. Given a seed set S, we can keep sampling hyperedges until $\mathcal{D}(S) \geq \Upsilon_1$. Then, $T \cdot \frac{\mathcal{D}(S)}{m_H}$ is an (ϵ, δ) estimation [72] of $\delta_A(S)$ according to Corollary 0.2. The analysis is similar in the cases of the lower bound and the upper bound.

Huang *et al.* [51] analyzed the conditions that the polling algorithmic framework must meet to obtain an approximation solution. Let S^* be the optimal seed set and $\hat{S}$ be the seed set returned by the greedy strategy on the estimate of the objective function $f(\cdot)$ ($\delta_L(\cdot)$ or $\delta_U(\cdot)$). Denote by $\hat{f}(\cdot)$ the estimate of the objective function $f(\cdot)$. The conditions are

$$Pr[\hat{f}(\hat{S}) \leq (1+\epsilon_1) f(\hat{S})] \geq 1 - \delta_1 \tag{5.6}$$

$$Pr[\hat{f}(S^*) \geq (1-\epsilon_2) f(S^*)] \geq 1 - \delta_2 \tag{5.7}$$

where $\delta_1 + \delta_2 \leq \delta$ and $\epsilon_1 + (1-1/e)\epsilon_2 \leq \epsilon$. Let N be the number of samples such that both Eq. 5.6 and Eq. 5.7 are guaranteed. Then we have the following lemma from [51].

Lemma 10. *Given a social network G, if the number of hyperedges $m_H \geq N$, then the polling algorithm returns $\hat{S}$ satisfying $Pr[f(\hat{S}) \geq (1-1/e-\epsilon) f(S^*)] \geq 1-\delta$ and $\hat{S}$ is an $(1-1/e-\epsilon)$ approximate solution.*

Applying Corollary 0.2 and Lemma 10, to obtain an approximation solution to maximizing the lower bound or upper bound, we can keep sampling hyperedges and checking if the conditions are met. The SSA-fix algorithm from [51] describes the process.

Using the SSA-fix algorithm, we can provide a $(1 - 1/e - \epsilon)$ approximation solution to maximizing the lower bound and the upper bound with probability of at least $1 - \delta$. But we must point out that the analysis does not hold for the activity maximization problem. This is because a necessary condition of the polling algorithmic framework is that we can approximate the estimate using the greedy strategy. The condition is not met in the case of the activity maximization problem, since the estimate of the activity is not submodular. Thus, the polling algorithm cannot provide an approximation solution to the activity maximization problem. But it is still a good heuristic for the activity maximization problem. Furthermore, by combining the approximation algorithm for the lower bound and the upper bound, we can derive a data dependent approximation scheme for the activity maximization problem.

5.5.4 Data Dependent Approximation

There is no general way to optimize or approximate a non-submodular function. Lu *et al.* [69] proposed a sandwich approximation strategy, which approximates the objective function by approximating its lower bound and upper bound. The sandwich approximation strategy works as follows. First, we find a solution to the original problem with any strategy. Second, we find an approximate solution to the lower bound and the upper bound, respectively. Last, we return the solution that has the best result for the original problem.

Here, we extend the strategy to the case in which the objective function is intractable and have the following result.

Algorithm 14 Sandwich Approximation Framework

1: Let S_U be a α approximation to the upper bound
2: Let S_L be a β approximation to the lower bound
3: Let S_A be a solution to the original problem
4: $\hat{\delta}_A(\cdot)$ is a multiplicative γ-error estimate of $\delta_A(\cdot)$
5: $S = \text{argmax}_{S_0 \in \{S_U, S_L, S_A\}} \hat{\delta}_A(S_0)$
6: **return** S

Theorem 32. *Let S be the seed set returned by Algorithm 14, then we have*

$$\delta_A(S) \geq \max\Big\{\frac{\delta_A(S_U)}{\delta_U(S_U)}\alpha, \frac{\delta_L(S_L^*)}{\delta_A(S_A^*)}\beta\Big\}\frac{1-\gamma}{1+\gamma}\delta_A(S_A^*) \tag{5.8}$$

Proof. Let S_L^*, S_U^* and S_A^* be the optimal solutions to maximizing the lower bound, the upper bound and the activity, respectively. Then, we have

$$\begin{aligned}\delta_A(S_U) &= \frac{\delta_A(S_U)}{\delta_U(S_U)}\delta_U(S_U) \geq \frac{\delta_A(S_U)}{\delta_U(S_U)} \cdot \alpha \cdot \delta_U(S_U^*) \\ &\geq \frac{\delta_A(S_U)}{\delta_U(S_U)} \cdot \alpha \cdot \delta_U(S_A^*) \geq \frac{\delta_A(S_U)}{\delta_U(S_U)} \cdot \alpha \cdot \delta_A(S_A^*) \\ \delta_A(S_L) &\geq \delta_L(S_L) \geq \beta \cdot \delta_L(S_L^*) \geq \frac{\delta_L(S_L^*)}{\delta_A(S_A^*)} \cdot \beta \cdot \delta_A(S_A^*)\end{aligned}$$

Let $S_{max} = \text{argmax}_{S_0 \in \{S_U, S_L, S_A\}} \delta_A(S_0)$, then

$$\delta_A(S_{max}) \geq \max\Big\{\frac{\delta_A(S_U)}{\delta_U(S_U)}\alpha, \frac{\delta_L(S_L^*)}{\delta_A(S_A^*)}\beta\Big\}\delta_A(S_A^*)$$

Since $\forall S_0 \in \{S_U, S_L, S_A\}, |\hat{\delta}_A(S_0) - \delta_A(S_0)| \leq \gamma\delta_A(S_0)$, we have $(1+\gamma)\delta_A(S) \geq (1-\gamma)\delta_A(S_{max})$. It follows that

$$\delta_A(S) \geq \frac{(1-\gamma)}{(1+\gamma)}\delta_A(S_{max})$$

□

Theorem 32 indicates that we can approximate the activity maximization problem within a factor that is dependent on the data. Since it is #P-hard to compute $\delta_A(\cdot)$ and $\delta_U(\cdot)$, and is NP-hard to find S_L^* and S_A^*, we cannot compute the exact approximation factor. But we can estimate $\frac{\delta_A(S_U)}{\delta_U(S_U)}$ by computing its lower bound $\frac{(1-\gamma)\hat{\delta}_A(S_U)}{(1+\gamma)\hat{\delta}_U(S_U)}$. It follows that $\frac{(1-\gamma)^2}{(1+\gamma)^2} \cdot \alpha \cdot \frac{\hat{\delta}_A(S_U)}{\hat{\delta}_U(S_U)}$ is a computable lower bound of the approximation factor.

Now, we put all the pieces of the puzzle together. We first adopt the polling algorithm to maximize the lower bound and the upper bound. As discussed in Section 5.5.3, it provides $(1-1/e-\epsilon)$ approximate solutions to the lower bound and the upper bound, respectively. Consequently, we have $\alpha = \beta = (1 - \frac{1}{e} - \epsilon)$ in Algorithm 14. Then, we also use the polling algorithm to get a heuristic solution (S_A) to the activity maximization problem. Last, we get a (γ, δ) estimation of $\delta_A(\cdot)$ based on Corollary 0.2 to complete Line 5 of Algorithm 14. According to Theorem 32, the sandwich algorithm returns a seed set S such that

$$\delta_A(S) \geq \max\Big\{\frac{\delta_A(S_U)}{\delta_U(S_U)}, \frac{\delta_L(S_L^*)}{\delta_A(S_A^*)}\Big\}\frac{1-\gamma}{1+\gamma}(1 - \frac{1}{e} - \epsilon)\delta_A(S_A^*)$$

5.6 Experiments

In this section, we evaluate our algorithm via a series of experiments on four real-world networks.

Table 5.2: The statistics of the networks.

Network	# Vertices	# Edges	Average degree
Douban	45,559	293,377	6.4
Aminer	1,712,433	4,258,615	2.5
DBLP	317,080	1,049,866	3.3
LiveJournal	3,997,962	34,681,189	8.7

5.6.1 Settings

We ran our experiments on four real-world networks, which include **Douban** [114], **AMiner** [102], **DBLP** [115] and **LiveJournal** [115]. The last two networks are available at the SNAP website (`http://snap.stanford.edu`). Table 5.2 shows the statistics of the networks.

The Douban network is a social network about movie ratings. We use the number of shared movies between a pair of users as their activity strength, which reflects the common interest between users. The AMiner network is an academic social network. We use the number of co-authored papers between a pair of users as their activity strength, which reflects the collaboration strength between users. To explore the possibility of other kind of activity strengths, we also verify our algorithm using two synthetic activity settings on the DBLP network and the Livejournal network. In the first case, we uniformly set $A_{u,v}$ to 1 for each edge (u, v). In the second case, we set $A_{u,v}$ to the value of the diffusion parameter $B_{u,v}$. The intuition is that there may be more interactions between u and v if u is more likely to activate v. The propagation probability $B_{u,v}$ for the IC model and the influence weight for the LT model of an edge (u, v) is set to $\frac{1}{degree(v)}$, as widely used in literature [16]. For the parameters controling approximation quality, we set $\epsilon = 0.1$, $\delta = 0.001$ and $\gamma = 0.05$ for all networks.

We compare the proposed algorithm, referred as Sandwich, with three heuristic algorithms: InfMax, Degree and PageRank. InfMax returns the vertices for influence maximization. We followed the implementation reported in [82]. Degree returns the vertices with high degrees. PageRank returns the vertices with high PageRank [86] scores.

We implemented our algorithm and the baselines in Java. All experiments were conducted on a PC with a 3.4GHZ Intel Core i7-3770 processor and 32 GB memory, running Microsoft Windows 7.

5.6.2 Effectiveness

Figure 5.6 shows the activity computed by each algorithm on the four networks, respectively. For better illustration, we report the *comparative gain ratio* instead of the absolute activity value, since the activity value scales vary greatly with respect to seed set size. It is easier to distinguish the gaps between the baselines and our algorithm when we use comparative gain ratio as the metric, since it is not affected by activity value scales. The comparative

gain ratio of an algorithm $\mathcal{A}$ is defined as $\frac{\delta_A(S_{\mathcal{A}})-\delta_A(S)}{\delta_A(S)}$, where $S_{\mathcal{A}}$ and S are the seed sets returned by algorithm $\mathcal{A}$ and the Sandwich algorithm, respectively.

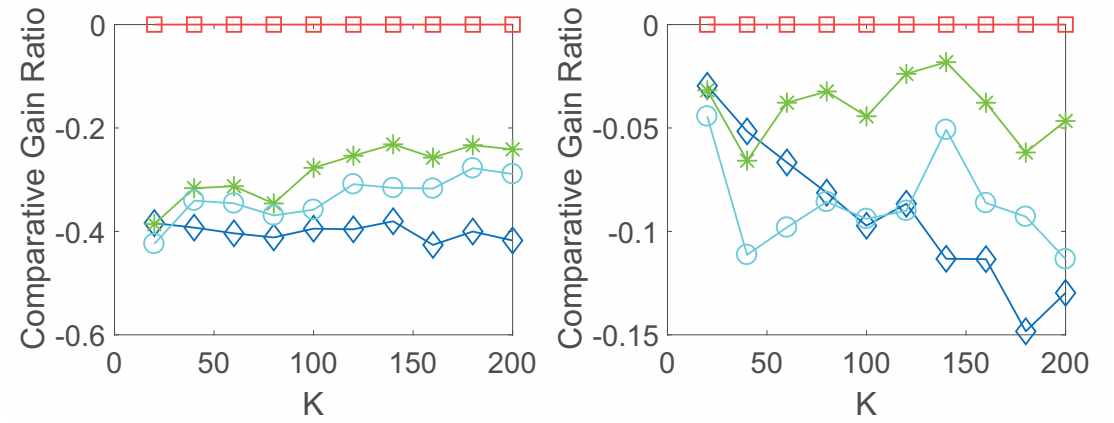
In the cases of real activity settings, our algorithm Sandwich always has the best performance. Only in very few cases of synthetic activity settings, Sandwich is outperformed marginally. In the real activity settings, InfMax has a poor performance and almost in all cases is the poorest one. This is because influence maximization only considers the number of active vertices and ignores the network structure. This phenomenon also demonstrates what we have pointed out in Section 5.1: more active users do not necessarily lead to more interaction activities.

In the uniform settings, algorithm Degree performs well under the IC model but has a relatively bad performance under the LT model. InfMax and PageRank often have a bad performance under both the IC model and the LT model in the uniform settings. In the diffusion settings, InfMax algorithm is a good heuristic under both the IC model and the LT model. Algorithm PageRank performs well on the DBLP network but has a bad performance on the other two networks. Algorithm Degree often has a bad performance under both the IC model and the LT model in the diffusion settings. These results show that these baseline algorithms are not stable in performance in this task and can only work well in some specific network or activity setting. The reason is that these baseline algorithms only use the properties of the social network or the diffusion process but totally ignore the activity strengths on edges. In contrast, our algorithm utilizes the unbiased estimate of the activity and its lower and upper bounds to solve the problem. This is why our algorithm always has a good performance while the baseline algorithms fail in many cases.

5.6.3 Approximation Quality

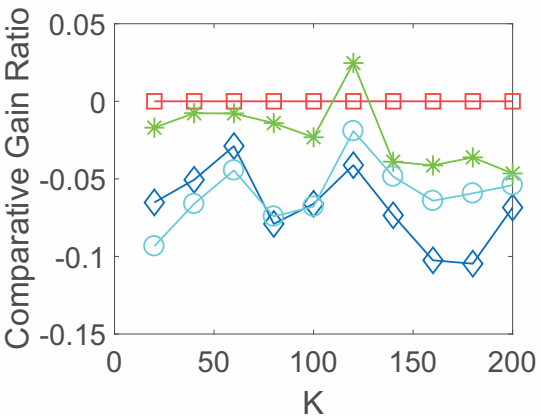
A major advantage of our algorithm is that it carries a data dependent approximation ratio. Since the exact approximation is intractable to compute, we report the computable lower bound of the approximation ratio, that is $\frac{(1-\gamma)^2}{(1+\gamma)^2} \cdot (1 - e - \epsilon) \cdot \frac{\hat{\delta}_A(S_U)}{\hat{\delta}_U(S_U)}$. Figure 5.7 shows the results on the four networks.

The ratio varies in different networks. On the same network, the ratios under the IC model and the LT model also differ. In general, the ratio under the LT model is greater than the one under the IC model in the same activity settings. The ratio does not change much with respect to the size of the seed set k. Roughly the ratio increases when k increases. A possible reason is that the gap between the activity and the upper bound shrinks when k increases, since there are more vertices activated with a larger value of k. Interestingly, we observe that, in terms of approximation ratio, the LT model consistently outperforms the IC model on both the real networks (i.e., Douban and Aminer in Figures 5.7(a)-(b)) and the networks with synthetic activities (i.e., DBLP and LiveJournal in Figures 5.7(c)-(d)). The consistency in the experimental results indicates that the uniform setting and diffusion setting of synthetic activities are two possible ways to simulate real activities.

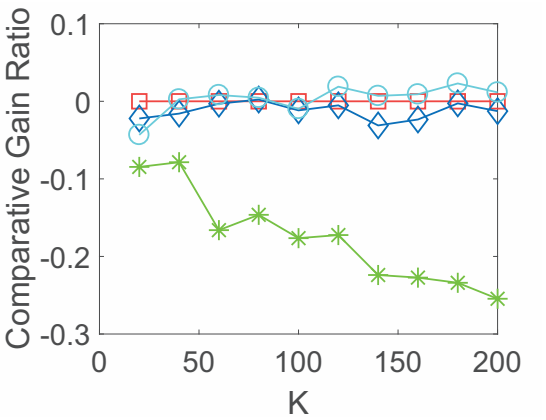


(a) Douban-IC

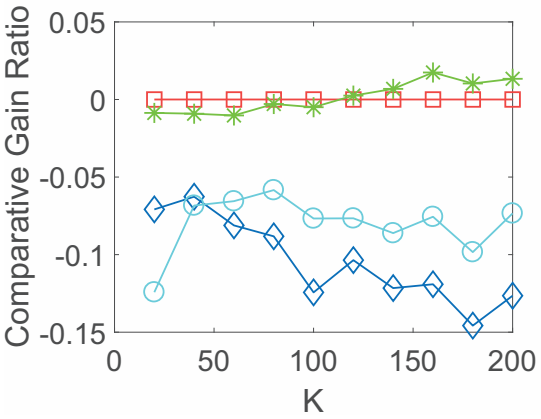
(b) Aminer-IC



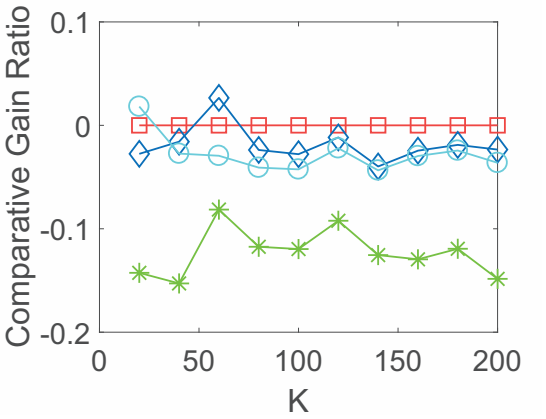
(c) DBLP-uniform-IC



(d) DBLP-diffusion-IC

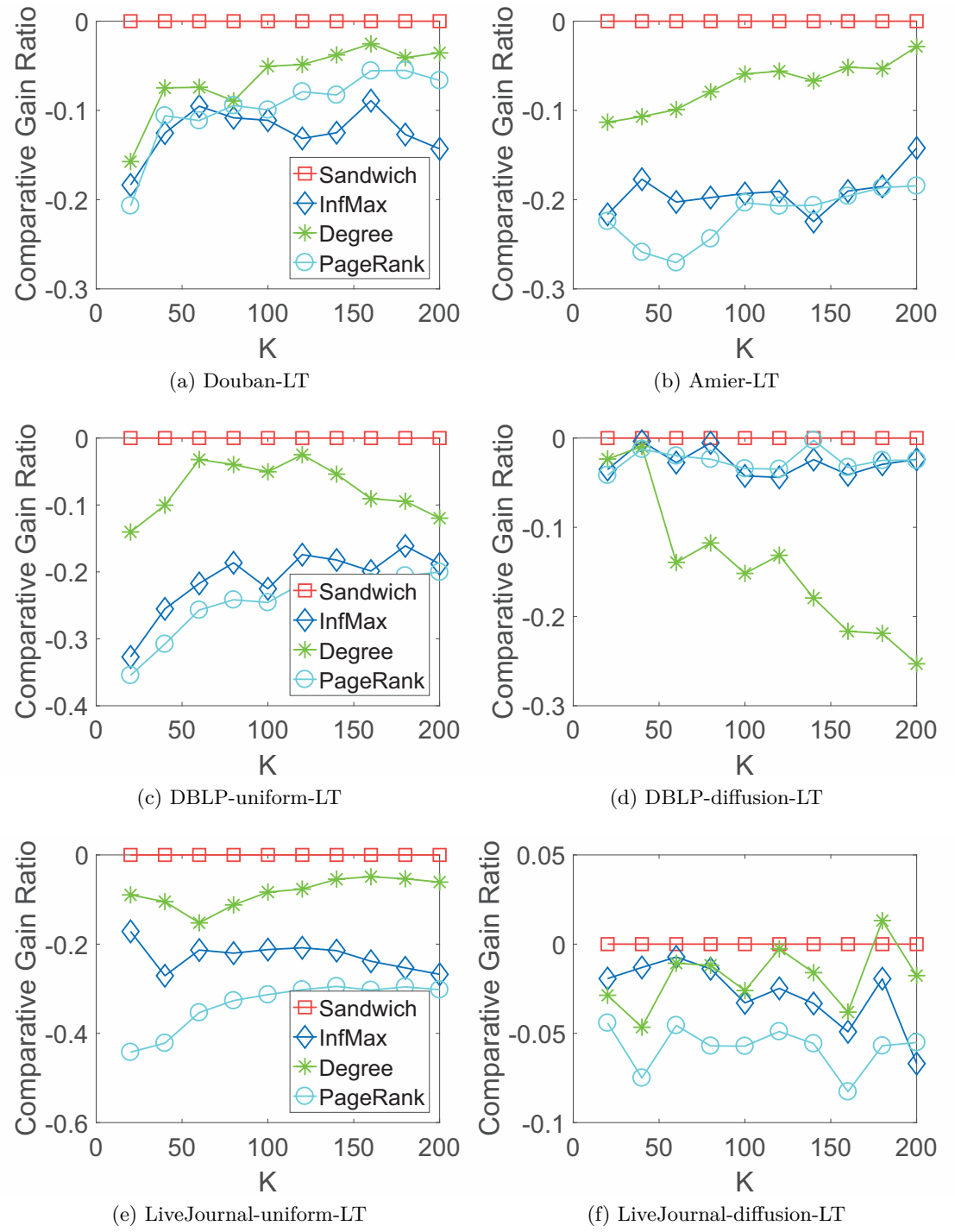


(e) LiveJournal-uniform-IC



(f) LiveJournal-diffusion-IC

Figure 5.5: Information activity on four networks. (a)-(d) show the performances on two real networks (Douban and Aminer) under IC model. (e)-(l) show the performances on real networks (DBLP and LiveJournal) with two types of synthetic activity settings, such as uniform and diffusion.

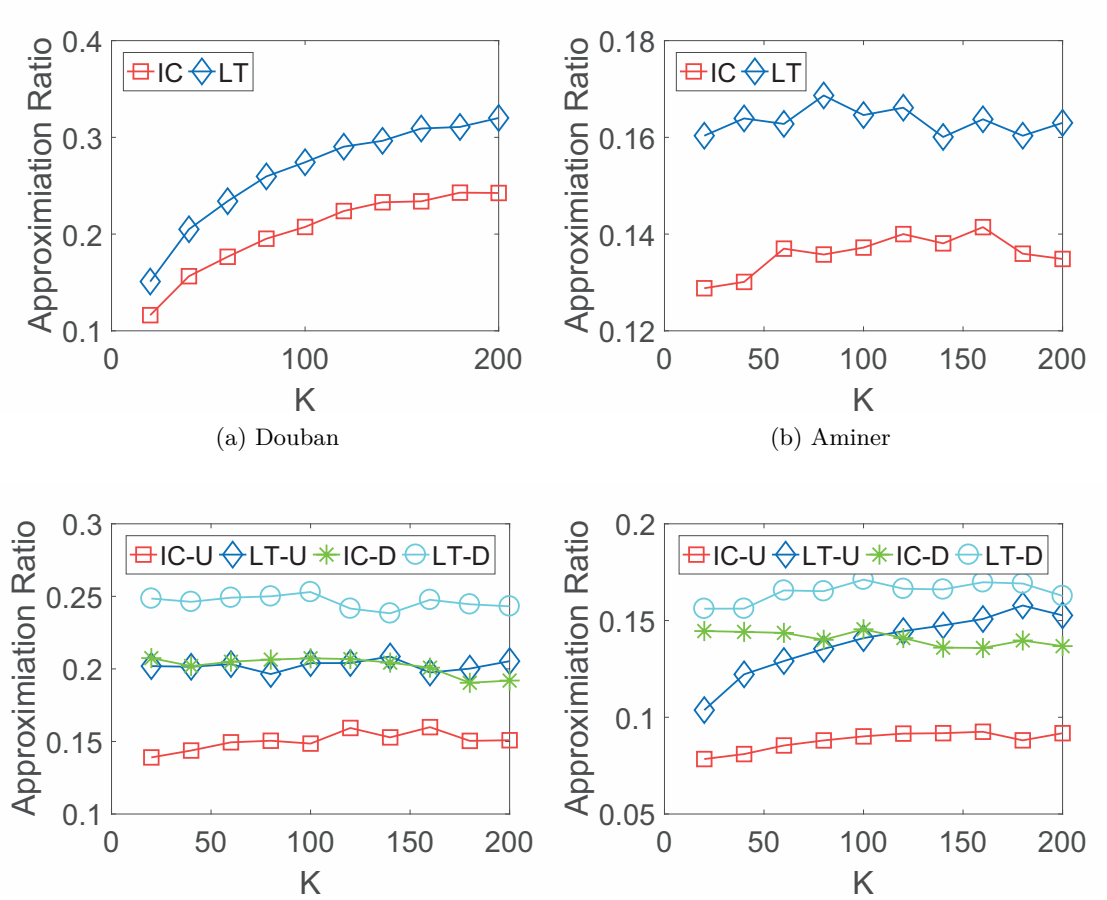


(a) Douban-LT (b) Amier-LT

(c) DBLP-uniform-LT (d) DBLP-diffusion-LT

(e) LiveJournal-uniform-LT (f) LiveJournal-diffusion-LT

Figure 5.6: Information activity on four networks. (a)-(d) show the performances on two real networks (Douban and Aminer) under LT model. (e)-(l) show the performances on real networks (DBLP and LiveJournal) with two types of synthetic activity settings, such as uniform and diffusion.



(a) Douban (b) Aminer

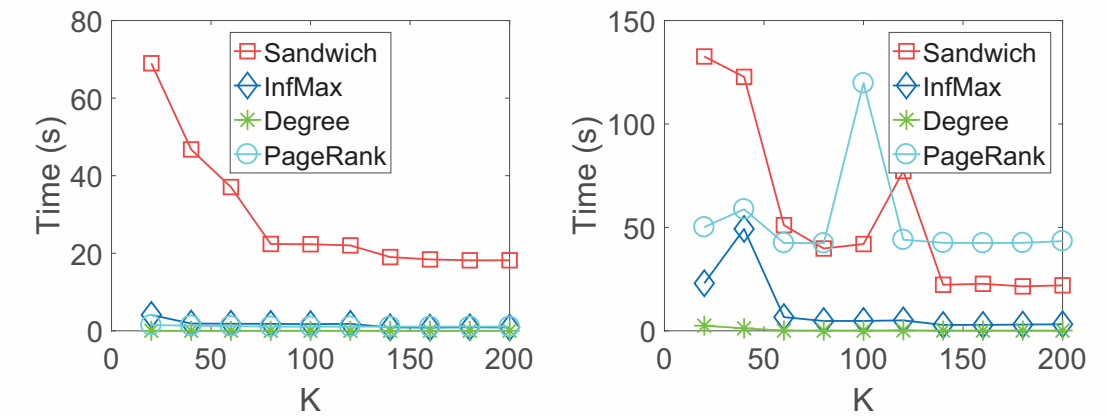
(c) DBLP (d) LiveJournal

Figure 5.7: The performance of approximation ratio on four networks. (a)-(b) show the performances on Douban and Aminer under IC model and LT model. (c)-(d) show the performances on DBLP and LiveJournal with two types of synthetic activity settings, such as uniform (U) and diffusion (D).

5.6.4 Scalability

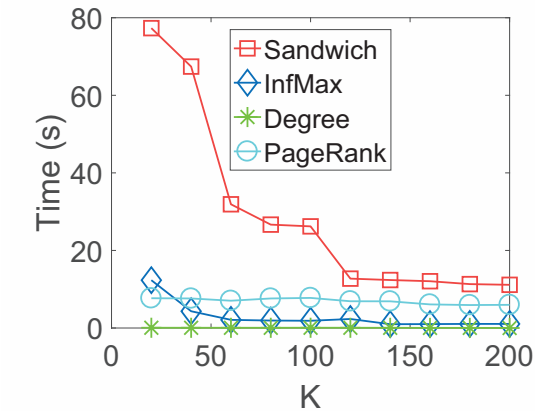
Since the activity settings do not affect the running time, we only report the running time in the uniform case. Fig. 5.8 and Fig. 5.9 shows the running time on the four networks.

In most of the cases, the running time of our algorithm decreases when the size of seed set k increases. This is because the time cost in Sandwich depends on the number of sampled hyperedges. According to Corollary 0.2, the expected number of samples is inversely proportional to μ_Z, which is the probability of the event $S \cap R_{g^T}(u) \neq \emptyset \wedge S \cap R_{g^T}(v) \neq \emptyset$. It increases when k increases. A similar analysis holds for the lower bound and the upper bound. PageRank is faster than our algorithm on the smallest networks but slower on the largest network. Degree and InfMax are more efficient than our algorithm, but they are substantially weaker than ours in effectiveness in many cases. As described in the previous

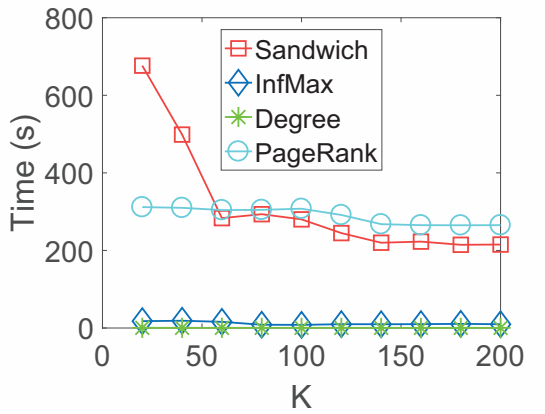


(a) Douban-IC

(b) AMiner-IC



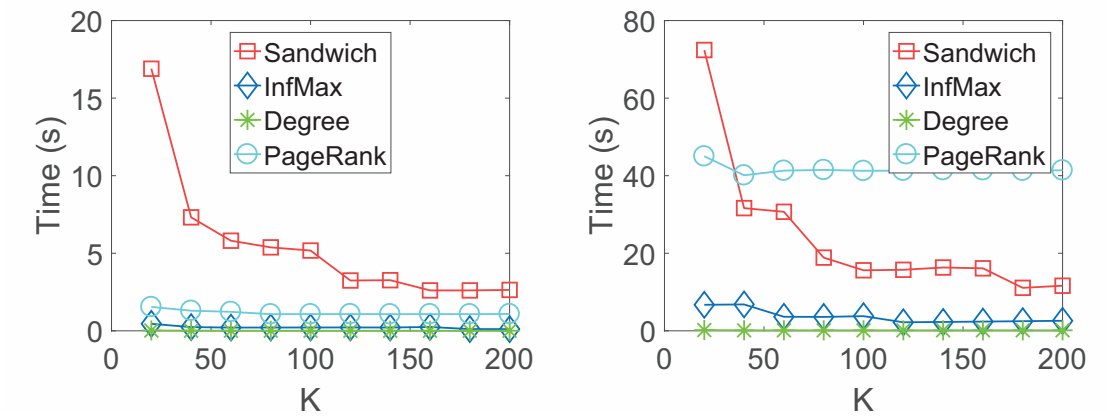
(c) DBLP-IC



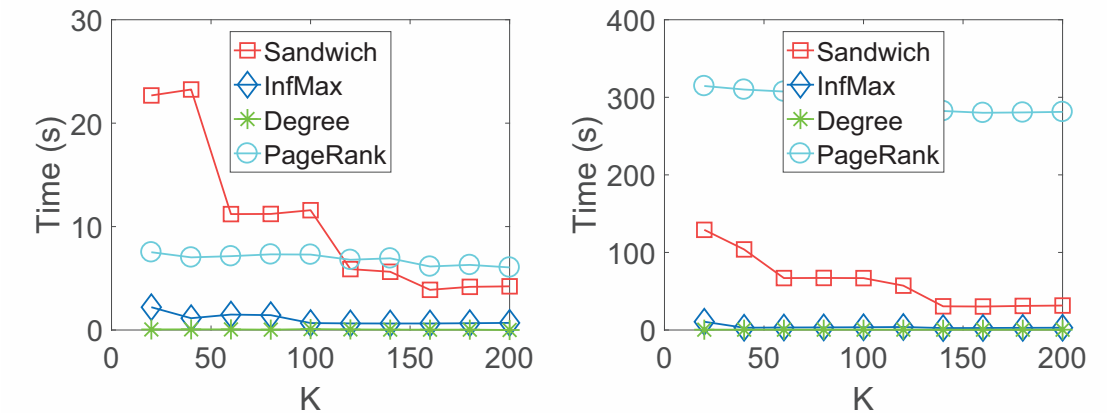
(d) LiveJournal-IC

Figure 5.8: The running time of all methods on four networks under IC model.

sections, there are many differences between our algorithm and InfMax, which lead to different time costs of the two algorithms. First, to obtain a data dependent approximation factor, the Sandwich algorithm actually solves three problems with polling based method. Second, during the sampling process of the original problem and maximizing the lower bound, we need to conduct a poll from both two endpoints of a randomly picked edge. Third, the sampling objects of the two algorithms are different, i.e., vertices versus edges. The sampling complexity of InfMax is mainly dependent on the number of vertices while the sampling complexity of Sandwich is mainly dependent on the number of edges. It is worthy noting that our algorithm is actually very efficient. The largest running time is only about 600 seconds on the largest network, which has millions of vertices and tens of millions of edges.

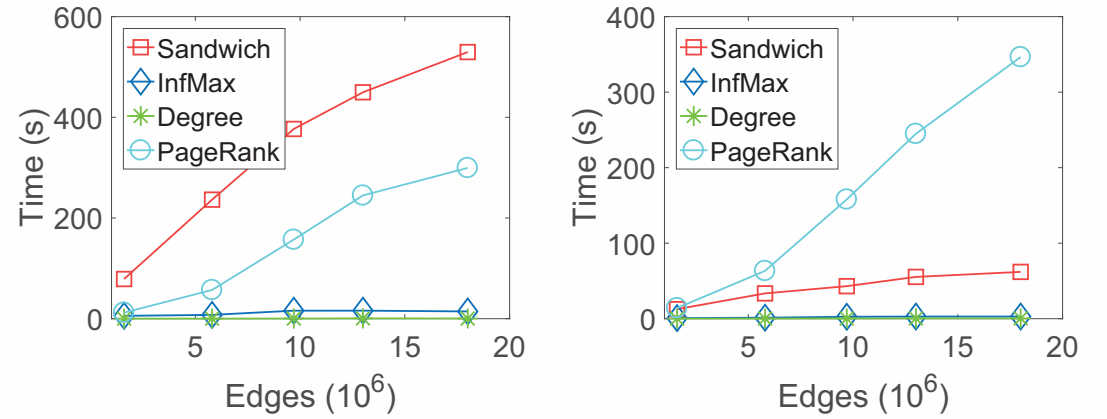


(a) Douban-LT (b) AMiner-LT



(c) DBLP-LT (d) LiveJournal-LT

Figure 5.9: The running time of all methods on four networks under LT model.



(a) IC model (b) LT model

Figure 5.10: The run running time of all methods on five sampled networks

Table 5.3: The sampled LiveJournal networks

Sample ID	1	2	3	4	5
Vertices ($\times 10^5$)	1.0	5.0	9.0	13	20
Edges ($\times 10^6$)	1.6	5.8	9.7	13	18

Table 5.4: Influence spread and information activity on the DBLP and LiveJournal networks (IC Model)

Data	k=20			k=200		
	influence	activity	ratio	influence	activity	ratio
DBLP	2,291	3,509	1.53	13,764	21,165	1.54
LiveJ	66,615	958,95	1.44	186,726	333,598	1.79

5.6.5 Influence Spread versus Activity

To further explore the scalability of the algorithms, we sample five networks from the LiveJournal networks as follows. First, we start a breadth first search (BFS) from a randomly selected vertex on the whole graph G until the desired number of vertices are visited. Denote by N the set of all vertices visited by the BFS. We use N to induce a subgraph G_N as the sampled network. The number of vertices and edges of the five subgraphs are listed in Table 5.3. After we obtain the sample networks, we run the algorithms when the size of seed set is set to 200. The results are shown in Figure 5.10. For both the IC model and the LT model, the Sandwich algorithm scales up roughly linearly with respect to the number of edges. Also, the slope in the IC model is greater than that in the LT model. In other words, the time cost increases more rapidly in the IC model. A possible reason is that the influence in the IC model is more sensitive to the number of edges in the graph, since each edge is activated independently in the IC model.

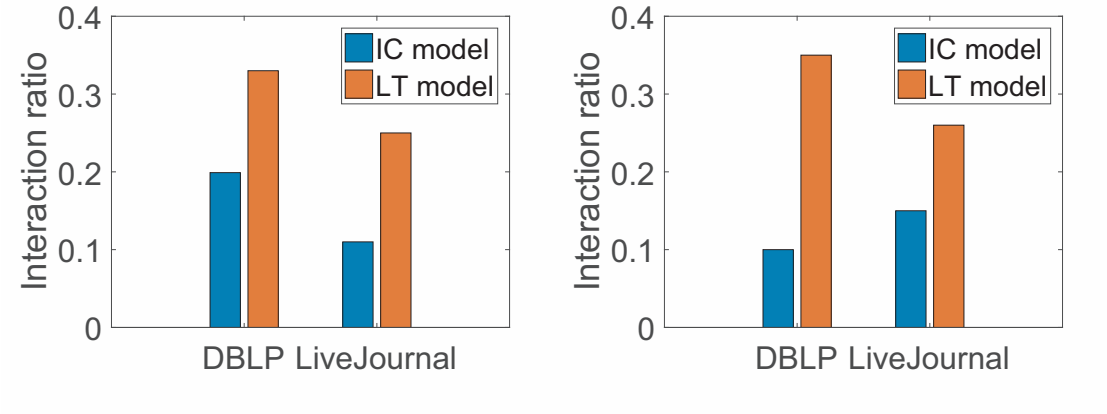
To explore the relation between influence spread and activity strength, we report their values on the DBLP network and the LiveJournal network with the uniform settings. We choose the uniform settings for these experiments here because, in such a situation, the total nactivity strength is exactly the number of edges between the active vertices. In this case, the activities can reflect the effect of the network structure formed by the propagation induced subgraph. In the the other two networks, there is no such correspondence. We also calculate the ratio, which is the influence spread against the information activity. Table 5.4 and Table 5.5 show the results on the two networks.

The ratio differs under different models. In general, the ratio under the LT model is greater than the one under the IC model. Possibly active vertices are more closely connected to each other under the LT model. We also notice that the ratio is similar when $k = 20$ and $k = 200$. This result suggests that the relation between the influence spread and the activity does not vary much with respect to the size of seed set. The reason is that more

Table 5.5: Influence spread and information activity on the DBLP and LiveJournal networks (LT Model)

Data	k=20			k=200		
	influence	activity	ratio	influence	activity	ratio
DBLP	2,834	5,179	1.83	17,445	32,712	1.88
LiveJ	89,559	184,842	2.06	297,014	839,334	2.83

seed vertices lead to more active vertices, but, at the same time, the activity also depends on the network structure among these vertices.



(a) $k = 20$ (b) $k = 200$

Figure 5.11: The interaction ratio performances of IC model and LT model on DBLP and LiveJournal networks under uniform activity setting.

The ratio can be viewed as the average degree of the propagation induced subgraph. The average degree of the propagation induced subgraph is smaller than the average degree of the whole graph. This is because only a small proportion of the vertices can be activated. Thus, there are many edges between active vertices and inactive vertices. The average degree of the propagation induced subgraph only considers the edges between active vertices. Thus, we report the interaction ratio of the active vertices, which is the number of edges whose both endpoints are active against the number of edges that have at least one active endpoint. The results are shown in Figure 5.11. The interaction ratios are not high on the two networks. This indicates that only a small proportion of the neighbors are activated and interact with the active vertices. This result demonstrates an essential difference between activity maximization and influence maximization.

5.7 Conclusions

In this chapter, to address the demand raised in several interesting applications, we proposed and formulated a novel problem, activity maximization. We proved the hardness of the

problem under both the IC model and the LT model. We also developed a lower bound and an upper bound of the objective function, and observed several useful properties of the lower bound and the upper bound. We designed a polling based algorithm to solve the problem that carries a data dependent approximation ratio. Our experimental results on four real networks verified the effectiveness and efficiency of our method. As future work we are interested in learning the activity of user pairs from real-world data. Moreover, considering the budget allocation version of the activity maximization similar to the continuous influence maximization in Chapter 4 is an interesting direction to explore.

Chapter 6

Conclusion and Future Directions

The booming of online social network services provides tremendous new opportunities to many traditional business applications like marketing. Leveraging the social propagation effect lies at the core of mining the marketing values of social networks, and faces many algorithmic challenges. In this thesis, we focus on algorithm designs for social propagation aware marketing, and develop solutions to a number of crucial problems emerging from important marketing applications using social networks.

In this chapter, we summarize our research presented in this thesis and discuss some future directions in social propagation mining.

6.1 Summary of the Thesis

Mining social propagations for marketing applications has been extensively studied. Most previous studies aim at finding a set of influential users to maximize the influence spread in a social network, and often assume that the underlying social network where social propagations take place is static. However, the highly dynamic nature of real social networks and the needs of many deeper marketing applications pose many unsettled and challenging algorithmic problems in leveraging social propagations. In this thesis, we identify three crucial social marketing problems that cannot be solved by previous studies, and make the following contributions.

- One fundamental task in utilizing social propagations in marketing is to extract influential users who can trigger large-scale propagations. There are two big challenges: (1) computing users? influence spread is often #P-hard [22, 20], and (2) real social networks are not static but evolve rapidly. To address these challenges, we devise efficient incremental algorithms that maintain a number of poll samples where we can extract top influential vertices with provable quality guarantees. When the network is only slightly updated, instead of re-generating all poll samples from scratch, our incremental algorithms only retrieve and update a few poll samples. One important problem is how many poll samples we need, as the sample size affects the quality of influential

vertices extracted. We devise efficient methods for deciding small and proper sample sizes for two common kinds of influential vertices queries, namely tracking vertices with influence spread of at least a threshold T , and tracking top-k influential individual vertices. Our methods are theoretically grounded that with high probability, the set of vertices S returned by our algorithms contains all real influential vertices (recall=100%) and influence spread of any false positive vertex in S is only slightly smaller than T in a threshold query, or I^k, the k-th greatest individual influence spread in a top-k query.

- When we want to promote a product through a social network, just identifying influential users is not enough. We need to spend some money on influential users to motivate them to trigger large-scale propagations for our marketing purposes. Suppose we know for each user the function of purchase probability with respect to discount. Then, what discounts should we offer to those social network users so that, under a predefined budget, the adoption of the product is maximized in expectation? To answer this question, we formulate the CIM (Continuous Influence Maximization) problem, which is a generalization of the well-known influence maximization problem. Denote by $C = (c_1, c_2, ...)$ a discount configuration, where $0 \leq c_u \leq 1$ is the discount offered to the user u. To find the best C under a budget B, we devise a 2-coordinate algorithmic framework where any propagation models can be plugged into. The idea is to improve the discount configuration C in an iterative manner. We investigate relations of CIM and traditional influence maximization. We also borrow concepts in statistical learning theory to develop principled implementations of our CIM algorithms under the family of triggering models, where the key point is how to effectively approximate the influence spread function to avoid the "overfitting" issue.
- The number of users involved in a propagation is a proper objective for marketing applications like adoption maximization. However, in some applications, the interaction activities among active users are also valuable to us. Imagine we want to stimulate a propagation from some influential users in a network for spreading a controversial social issue, such as Trump?s pulling the US out of TPP. The goal is to make users in the social network discuss the issue as much as possible. Thus, instead of the number of users retweeting the words, what matters is the post-propagation marketing effect, that is, how intense the discussion between users who retweet will be. We model the strength of the activity/discussion of two users in a network as the weight of the edge connecting them, and propose the Activity Maximization problem. Our goal is to select k seed users to trigger a propagation, such that the expected sum of weights of edges between users involved in the propagation is maximized. We prove the NP-hardness and inapproximability of the activity maximization problem. To solve this problem, we derive an upper bound and a lower bound of the activity function to apply the Sandwich approximation algorithm. We also devise a novel polling based

algorithm to approximate the activity function, the upper bound and the lower bound in order to obtain a data-dependent approximation factor of our solution.

6.2 Future Directions

Besides direct extensions of our current research discussed at the end of each chapter, there are also some important directions in mining social propagations for business applications.

6.2.1 Social Marketing Optimization under Noisy Influence Oracles

So far for social marketing optimization problems such as influence maximization and its variations, budget minimization and the budget allocation problem introduced in Chapter 4, almost all existing studies assume an underlying propagation model in the problem setting. Most studies adopt the Independent Cascade model and the Linear Threshold model, and there are also many studies using the Continuous-Time Diffusion model. Although those popularly used propagation models do capture some intuitions and properties of real social propagations, there always are networks where the mechanism of propagations differs very much from the assumptions of those models. In such cases, the popularly adopted propagation models may lead to large model bias.

In many social marketing optimization problem, the most important information derived from a propagation model is the influence spread function[1]. To settle the potential large model bias issue of propagation models, some studies investigate how to learn the influence spread function directly from users' action data [34, 49]. People even proposed black box learning method like deep neural networks to learn the influence spread function [108, 88].

Compared to the current research in social marketing optimizations, a more realistic problem setting is that we only have a noisy oracle that returns influence spreads of seed sets with a reasonable level of accuracy. Such an oracle can be obtained by applying methods that direct learn the influence spread function [34, 49, 108, 108, 88]. Then one important and challenging problem is, with such a noisy oracle, how do we design robust [48, 17] optimization algorithms for social marketing? Note that since the oracle is often a black box, we cannot apply methods like the polling sketch that utilizes properties of specific propagation models for approximating influence spread.

6.2.2 Ensemble Learning Propagation Graphs

In our research presented in this thesis, we assume that we have the full information of the propagation graph. However, in reality, the propagation graph is usually not known to us in

[1]An influence spread function $I : 2^V \to [0, |V|]$ is just a function returning the influence spread $I(S)$ of a given seed set S.

advance. Since the propagation graph can be regarded as an parameter of a influence propagation model, we can apply machine learning methods like maximum likelihood learning to learn the propagation graph when given some historical propagation records, which can be derived from users' action data [29, 40, 42, 79, 93, 59, 91, 76].

For every stochastic propagation model, we can develop a learning algorithm to learn the propagation graph. However, although popular propagation models like the Continuous-Time Diffusion model and the Linear Threshold model were verified be able to fit real propagations to some degree, they still are different from the real mechanism of propagations. One famous fact in machine learning is that every model has its limitations and ensemble results by multiple models often can get better learning results [118]. Thus, applying ensemble learning to the propagation graph learning problem, how can we combine results (which are graphs) of methods assuming different propagation models? One key primitive is that how to calculate the "average" of a given collection of graphs $\{G_1, G_2, ..., G_k\}$. Unlike numerical data, there is no nature way to define this "average" for graphs.

We give one possible way to take the "average" of multiple graphs. Inspired by the fact from geometry that, the average of $\{\mathbf{x_1}, ..., \mathbf{x_k}\}$ is $\mathbf{y} = \arg\min_{x \in \mathbf{x}} \sum_{i=1}^{k} \frac{1}{k} |\mathbf{x} - \mathbf{x_i}|$. We can define the average of $\{G_1, G_2, ..., G_k\}$ as

$$G = \arg \min_{G' \in \mathbf{G}} \sum_{i=1}^{k} \frac{1}{k} Dis(G', G_i) \tag{6.1}$$

where $Dis(\cdot, \cdot)$ is a graph distance function, for example, the graph edit distance. A big algorithmic challenge is how to efficiently calculate the average G in Eq. 6.1 when we are given a graph distance function.

Taking the average of graphs for ensemble learning can also be regarded as a fundamental tool in graph mining, which can be applied to many other applications like learning scene graphs in robotics vision [113, 80, 117] and learning knowledge graphs [52, 66].

6.2.3 Adversarial Propagation Graph Learning

Learning the propagation network from users' action data is not always a good thing. Sometimes we may want to hide certain parts of the network from a learning algorithm.

The public and private model of graphs [23] draws a lot of attention from both academia and industry recently. Statistics show that 52.6% Facebook users from New York City hide part of their friends list [32]. Thus, the social network owner is responsible to keep these hidden links from being hacked. However, propagation data or timestamped user action data may cause leakage of hidden social connections. For example, if very often a Facebook user A shares the article that user B just shared a few minutes ago, we can infer that probably A follows B or A and B are friends in Facebook, even when A hides her connection to B.

Besides edges that are hidden by social network users, there may be some other edges that the social network owner needs to hide. To make money by providing viral marketing plans to customers [15], the social network owner should make sure that customers cannot figure out good marketing plans themselves. One natural idea is to hide some crucial links for propagations in the network such that a customer who wants to conduct a viral marketing campaign cannot find the real influential users. Since people normally use influence maximization to design viral marketing campaigns, an interesting algorithmic problem is how to select a small given number of edges to hide, such that the solution of influence maximization is impaired the most.

Once the social network owner decides to make what edges invisible to a third party, the task is to massage the propagation data (timestamped user action data) posted online to make a social graph learning algorithm fail to learn the existence of these edges. We can make some user action records not accessible and we can modify timestamps of some records. When one crawls the user action data via APIs for the purpose of data analysis, she will get the massaged data. One key point is that we hope the massaged data not be much different from the original data, such that we can still draw reliable statistics like how many users share an article to do analysis that does not involve the privacy issue.

Bibliography

[1] Nitin Agarwal, Huan Liu, Lei Tang, and Philip S Yu. Identifying the influential bloggers in a community. In *Proceedings of the 2008 international conference on web search and data mining*, pages 207–218. ACM, 2008.

[2] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[3] Charu C Aggarwal, Shuyang Lin, and Philip S Yu. On influential node discovery in dynamic social networks. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 636–647. SIAM, 2012.

[4] Roy M Anderson, Robert M May, and B Anderson. *Infectious diseases of humans: dynamics and control*, volume 28. Wiley Online Library, 1992.

[5] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[6] Joseph Berger, Susan J Rosenholtz, and Morris Zelditch Jr. Status organizing processes. *Annual review of sociology*, 6(1):479–508, 1980.

[7] Smriti Bhagat, Amit Goyal, and Laks VS Lakshmanan. Maximizing product adoption in social networks. In *WSDM*, pages 603–612. ACM, 2012.

[8] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In *STOC*, pages 201–210. ACM, 2010.

[9] Francesco Bonchi. Influence propagation in social networks: A data mining perspective. *IEEE Intelligent Informatics Bulletin*, 12(1):8–16, 2011.

[10] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 946–957. SIAM, 2014.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[12] Mike Brennan. Constructing demand curves from purchase probability data: an application of the juster scale. *Marketing Bulletin*, 6(May):51–58, 1995.

[13] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 182–196. Springer, 2007.

[14] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, P Krishna Gummadi, et al. Measuring user influence in twitter: The million follower fallacy. *Icwsm*, 10(10-17):30, 2010.

[15] Parinya Chalermsook, Atish Das Sarma, Ashwin Lall, and Danupon Nanongkai. Social network monetization via sponsored viral marketing. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 259–270. ACM, 2015.

[16] Wei Chen, Laks VS Lakshmanan, and Carlos Castillo. Information and influence propagation in social networks. *Synthesis Lectures on Data Management*, 5(4):1–177, 2013.

[17] Wei Chen, Tian Lin, Zihan Tan, Mingfei Zhao, and Xuren Zhou. Robust influence maximization. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 795–804. ACM, 2016.

[18] Wei Chen, Wei Lu, and Ning Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. In *AAAI*, pages 592–598. AAAI Press, 2012.

[19] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038. ACM, 2010.

[20] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208. ACM, 2009.

[21] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 88–97. IEEE, 2010.

[22] Xiaodong Chen, Guojie Song, Xinran He, and Kunqing Xie. On influential nodes tracking in dynamic social networks. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 613–621. SIAM, 2015.

[23] Flavio Chierichetti, Alessandro Epasto, Ravi Kumar, Silvio Lattanzi, and Vahab Mirrokni. Efficient algorithms for public-private social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148. ACM, 2015.

[24] Yeong Bin Cho, Yoon Ho Cho, and Soung Hie Kim. Mining changes in customer buying behavior for collaborative recommendations. *Expert Systems with Applications*, 28(2):359–369, 2005.

[25] Fan Chung and Linyuan Lu. Concentration inequalities and martingale inequalities: a survey. *Internet Mathematics*, 3(1):79–127, 2006.

[26] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 629–638. ACM, 2014.

[27] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[28] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

[29] Hadi Daneshmand, Manuel Gomez-Rodriguez, Le Song, and Bernhard Schölkopf. Estimating diffusion network structures: Recovery conditions, sample complexity & soft-thresholding algorithm. In *International Conference on Machine Learning*, pages 793–801, 2014.

[30] Erik D Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, David L Malec, S Raghavan, Anshul Sawant, and Morteza Zadimoghadam. How to influence people with partial incentives. In *Proceedings of the 23rd international conference on World wide web*, pages 937–948. ACM, 2014.

[31] Morton Deutsch and Harold B Gerard. A study of normative and informational social influences upon individual judgment. *The journal of abnormal and social psychology*, 51(3):629, 1955.

[32] Ratan Dey, Zubin Jelveh, and Keith Ross. Facebook users have become much more private: A large-scale study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 346–352. IEEE, 2012.

[33] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001.

[34] Nan Du, Yingyu Liang, Maria Balcan, and Le Song. Influence function learning in information diffusion networks. In *International Conference on Machine Learning*, pages 2016–2024, 2014.

[35] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *Advances in Neural Information Processing Systems*, pages 3147–3155, 2013.

[36] Milad Eftekhar, Yashar Ganjali, and Nick Koudas. Information cascade at group scale. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–409. ACM, 2013.

[37] Mehrdad Farajtabar, Nan Du, Manuel Gomez-Rodriguez, Isabel Valera, Hongyuan Zha, and Le Song. Shaping social activity by incentivizing users. In *NIPS*, 2014.

[38] 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.

[39] Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 2001:1, 2001.

[40] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1019–1028. ACM, 2010.

[41] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Discovering leaders from community actions. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 499–508. ACM, 2008.

[42] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.

[43] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84, 2011.

[44] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 211–220. IEEE, 2011.

[45] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.

[46] Adrien Guille, Hakim Hacid, Cecile Favre, and Djamel A Zighed. Information diffusion in online social networks: A survey. *ACM Sigmod Record*, 42(2):17–28, 2013.

[47] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. Fully dynamic betweenness centrality maintenance on massive networks. *Proceedings of the VLDB Endowment*, 9(2):48–59, 2015.

[48] Xinran He and David Kempe. Stability and robustness in influence maximization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(6):66, 2018.

[49] Xinran He, Ke Xu, David Kempe, and Yan Liu. Learning influence functions from incomplete observations. In *Advances in Neural Information Processing Systems*, pages 2073–2081, 2016.

[50] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, and Zhen He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380. ACM, 2011.

[51] Keke Huang, Sibo Wang, Glenn Bevilacqua, Xiaokui Xiao, and Laks VS Lakshmanan. Revisiting the stop-and-stare algorithms for influence maximization. *Proceedings of the VLDB Endowment*, 10(9):913–924, 2017.

[52] Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *AAAI*, pages 985–991, 2016.

[53] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[54] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[55] David Kempe, Jon M Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11(4):105–147, 2015.

[56] Subhash Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006.

[57] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 259–271. Springer, 2006.

[58] Aradhna Krishna. The impact of dealing patterns on purchase behavior. *Marketing Science*, 13(4):351–373, 1994.

[59] Konstantin Kutzkov, Albert Bifet, Francesco Bonchi, and Aristides Gionis. Strip: stream learning of influence probabilities. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 275–283. ACM, 2013.

[60] Bibb Latané. The psychology of social impact. *American psychologist*, 36(4):343, 1981.

[61] Bibb Latané. Dynamic social impact: The creation of culture by communication. *Journal of communication*, 46(4):13–25, 1996.

[62] Siyu Lei, Silviu Maniu, Luyi Mo, Reynold Cheng, and Pierre Senellart. Online influence maximization. In *Proceedings of the 21st ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 645–654. ACM, 2015.

[63] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470. ACM, 2008.

[64] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[65] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[66] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187, 2015.

[67] Hairong Liu, Longin Jan Latecki, and Shuicheng Yan. Fast detection of dense subgraphs with iterative shrinking and expansion. *IEEE transactions on pattern analysis and machine intelligence*, 35(9):2131–2142, 2013.

[68] Cheng Long and RC-W Wong. Minimizing seed set for viral marketing. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 427–436. IEEE, 2011.

[69] Wei Lu, Wei Chen, and Laks VS Lakshmanan. From competition to complementarity: comparative influence diffusion and maximization. *VLDB*, 9(2):60–71, 2015.

[70] Wei Lu and Laks VS Lakshmanan. Profit maximization over social networks. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 479–488. IEEE, 2012.

[71] Brendan Lucier, Joel Oren, and Yaron Singer. Influence at scale: Distributed computation of complex contagion in networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744. ACM, 2015.

[72] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[73] Elchanan Mossel and Sebastien Roch. On the submodularity of influence in social networks. In *Proceedings of the 39th annual ACM symposium on Theory of computing*, pages 128–134. ACM, 2007.

[74] Mehdi Moussaïd, Juliane E Kämmer, Pantelis P Analytis, and Hansjörg Neth. Social influence and the collective dynamics of opinion formation. *PloS one*, 8(11):e78433, 2013.

[75] Katta G Murty and Santosh N Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical programming*, 39(2):117–129, 1987.

[76] Harikrishna Narasimhan, David C Parkes, and Yaron Singer. Learnability of influence in networks. In *Advances in Neural Information Processing Systems*, pages 3186–3194, 2015.

[77] Ramasuri Narayanam and Amit A Nanavati. Viral marketing for product cross-sell through social networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 581–596. Springer, 2012.

[78] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsâĂŤi. *Mathematical Programming*, 14(1):265–294, 1978.

[79] Praneeth Netrapalli and Sujay Sanghavi. Learning the graph of epidemic cascades. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 211–222. ACM, 2012.

[80] Alejandro Newell and Jia Deng. Pixels to graphs by associative embedding. In *Advances in neural information processing systems*, pages 2171–2180, 2017.

[81] Hung T Nguyen, Thang N Dinh, and My T Thai. Cost-aware targeted viral marketing in billion-scale networks. In *INFOCOM*. IEEE, 2016.

[82] Hung T Nguyen, My T Thai, and Thang N Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 695–710. ACM, 2016.

[83] Huy Nguyen and Rong Zheng. On budgeted influence maximization in social networks. *Selected Areas in Communications, IEEE Journal on*, 31(6):1084–1094, 2013.

[84] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment*, 9(12):1077–1088, 2016.

[85] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 875–884. ACM, 2015.

[86] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[87] Andrea Pietracaprina, Matteo Riondato, Eli Upfal, and Fabio Vandin. Mining top-k frequent itemsets through progressive sampling. *Data Mining and Knowledge Discovery*, 21(2):310–326, 2010.

[88] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119. ACM, 2018.

[89] Matteo Riondato and Eli Upfal. Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(4):20, 2014.

[90] Matteo Riondato and Eli Upfal. Mining frequent itemsets through progressive sampling with rademacher averages. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1005–1014. ACM, 2015.

[91] Manuel Gomez Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. *arXiv preprint arXiv:1105.0697*, 2011.

[92] Maria-Evgenia G Rossi, Fragkiskos D Malliaros, and Michalis Vazirgiannis. Spread it good, spread it fast: Identification of influential nodes in social networks. In *Proceedings of the 24th International Conference on World Wide Web*, pages 101–102. ACM, 2015.

[93] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 67–75. Springer, 2008.

[94] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[95] Carl Shapiro, Shapiro Carl, Hal R Varian, et al. *Information rules: a strategic guide to the network economy*. Harvard Business Press, 1998.

[96] Tomoji Shogenji. A condition for transitivity in probabilistic support. *The British journal for the philosophy of science*, 54(4):613–616, 2003.

[97] Yaron Singer. How to win friends and influence people, truthfully: influence maximization mechanisms for social networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 733–742. ACM, 2012.

[98] Euiho Suh, Seungjae Lim, Hyunseok Hwang, and Suyeon Kim. A prediction model for the purchase probability of anonymous customers to support real time web marketing: a case study. *Expert Systems with Applications*, 27(2):245–255, 2004.

[99] Jimeng Sun and Jie Tang. A survey of models and algorithms for social influence analysis. In *Social network data analytics*, pages 177–214. Springer, 2011.

[100] Yu Sun, Rui Zhang, Andy Yuan Xue, Jianzhong Qi, and Xiaoyong Du. Reverse nearest neighbor heat maps: A tool for influence exploration. In *ICDE*, pages 966–977. IEEE, 2016.

[101] Fangshuang Tang, Qi Liu, Hengshu Zhu, Enhong Chen, and Feida Zhu. Diversified social influence maximization. In *ASONAM*, pages 455–459. IEEE, 2014.

[102] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, pages 990–998. ACM, 2008.

[103] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*. ACM, 2015.

[104] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86. ACM, 2014.

[105] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[106] Gaebler Ventures. Business advertising. `http://www.gaebler.com/Business-Advertising.htm`, 2013.

[107] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[108] Jia Wang, Vincent W Zheng, Zemin Liu, and Kevin Chen-Chuan Chang. Topological recurrent neural network for diffusion prediction. In *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 475–484. IEEE, 2017.

[109] Yue Wang, WeiJing Huang, Lang Zong, TengJiao Wang, and DongQing Yang. Influence maximization with limit cost in social network. *Science China Information Sciences*, 56(7):1–14, 2013.

[110] Zhefeng Wang, Enhong Chen, Qi Liu, Yu Yang, Yong Ge, and Biao Chang. Maximizing the coverage of information propagation in social networks. In *IJCAI*, pages 2104–2110. AAAI Press, 2015.

[111] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 261–270. ACM, 2010.

[112] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. On computing top-t most influential spatial sites. In *VLDB*, pages 946–957. VLDB Endowment, 2005.

[113] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2017.

[114] Tong Xu, Dong Liu, Enhong Chen, Huanhuan Cao, and Jilei Tian. Towards annotating media contents through social diffusion analysis. In *ICDM*, pages 1158–1163. IEEE, 2012.

[115] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[116] Jing Yuan and Shao-Jie Tang. Adaptive discount allocation in social networks. In *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, page 22. ACM, 2017.

[117] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5831–5840, 2018.

[118] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

Appendix: List of Selected Publications

1. **Yu Yang**, Xiangbo Mao, Jian Pei and Xiaofei He. "Continuous Influence Maximization: Algorithmic Framework and Implementation under Triggering Models". Submitted to ACM Transactions on Knowledge Discovery from Data (TKDD)
2. **Yu Yang** and Jian Pei. "Influence Analysis in Evolving Networks: A Survey". Under 2nd Round Review, submitted to *IEEE Transactions on Knowledge and Data Engineering (TKDE)*
3. **Yu Yang**, Lingyang Chu, Yanyan Zhang, Zhefeng Wang, Jian Pei and Enhong Chen. "Mining Density Contrast Subgraphs". In *Proceedings of the 34th IEEE International Conference on Data Engineering (ICDE'18)*, Paris, France, April 16th-19th, 2018, pp.221-232.
4. **Yu Yang***, Zhefeng Wang*, Jian Pei, Lingyang Chu and Enhong Chen. "Activity Maximization by Effective Information Diffusion in Social Networks". *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 29.11 (2017): 2374-2387. (* Equal Contribution[2])
5. **Yu Yang**, Zhefeng Wang, Jian Pei and Enhong Chen. "Tracking Influential Individuals in Dynamic Networks". *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 29.11 (2017): 2615-2628.
6. **Yu Yang**, Jian Pei, and Abdullah Al-Barakati. "Measuring in-network node similarity based on neighborhoods: a unified parametric approach." *Knowledge and Information Systems (KAIS)* (2017): 1-28.
7. **Yu Yang**, Xiangbo Mao, Jian Pei and Xiaofei He. "Continuous Influence Maximization: What Discounts Should We Offer to Social Network Users?". In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, San Francisco, USA, 2016, pp.727-741.

[2]Yu Yang proposed the Activity Maximization problem and did most of the theoretical analysis. Zhefeng Wang derived the data-dependent approximation ratio, wrote the code, conducted the experiments and wrote a draft of the paper.