Social Influence And Its Applications

An algorithmic and data mining study

by

Amit Goyal

B.Tech., Indian Institute of Technology, Bombay, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

February 2013

Abstract

Social influence occurs when one's actions are affected by others. While the study of social influence has a long history in the fields of sociology and marketing, the first study on it from an algorithmic and data mining perspective was performed in 2001. Since then, it has attracted tremendous attention, both in research and in industry. If leveraged carefully, social influence can be exploited in many applications like viral marketing (or targeted advertising in general), recommender systems, analysis of information diffusion in Social Media like Twitter and Facebook, events detection, experts finding, link prediction, ranking of feeds etc. One of the fundamental problems in this fascinating field is the problem of influence maximization, primarily motivated by the application of viral marketing. The objective is to identify a small set of users (called *seed set*) in a social network, who when convinced to adopt a product will influence others in the network leading to a large number of adoptions in an expected sense.

The vision of our work is to take the algorithmic and data mining aspects of viral marketing out of the lab. We organize ours goals and contributions into four categories: (i) With the ever-increasing scale of online social networks, it is extremely important to develop efficient algorithms for influence maximization. We propose two algorithms – CELF++ and SIMPATH that significantly improve the scalability. In particular, SIMPATH outperforms the previous state of the art on all three fronts – running time, memory consumption and the quality of the seed set chosen. (ii) We remark that previous studies often make unrealistic assumptions and rely on simulations, instead of validating models against real world data. For instance, they assume an arbitrary assignment of influence probabilities in their studies which focused more on algorithms than on quality and validity with respect to real data. We attack the problem of learning influence probabilities. One of our key findings is that the probabilities are not static and change over time. In view of this, in another work, we propose a novel data driven approach to influence models. We compare our influence model with previous ones and show that it predicts influence diffusion with much better accuracy. (iii) Next, we propose alternative problem formulations – MINTSS and MINTIME and show interesting theoretical results. These problem formulations capture the problem of deploying viral campaigns on budget and time constraints. In an additional work, we take a fresh perspective on identifying community leaders who may have high influence within their community, using a pattern mining approach. (iv) Finally, we examine applications of social influence. First, we look at the application of viral marketing. Using real world datasets, we show that product adoption is not exactly influence. Given this, we develop a product adoption model and study the problem of maximizing product adoption. Furthermore, we propose and investigate a novel problem in recommender systems, for targeted advertising – RECMAX. In recommender systems, the flow of influence (or information) happens not via explicit friendship (social) links, but based on recommendation system models. Hence, targeted advertising in recommender systems requires these information channels are being taken in account.

Preface

This dissertation is the result of collaboration with several researchers. The common and most influential among them is my supervisor Laks V. S. Lakshmanan from the University of British Columbia, Canada. All the studies included in this dissertation are published in peer-reviewed venues and roughly, each publication constitutes a main technical chapter. Algorithms CELF++ and SIMPATH to improve the efficiency of the influence maximization solutions are included in Chapter 3 and are based on publications in WWW 2011 (companion volume) [60] and ICDM 2011 [61]. The work on learning influence probabilities from past action log of users is presented in Chapter 4 and is based on our paper in WSDM 2010 [56]. Next, a data-driven approach to social influence maximization in Chapter 5 is described in a VLDB 2012 paper [57]. Theoretical analysis on minimizing budget and propagation time in influence propagation is presented in Chapter 6 and is based on our journal in SNAM 2012 [58]. A pattern mining framework to discover community leaders in Chapter 7 is described in a CIKM 2008 paper [55]. Distinguishing product adoption from influence spread, we present techniques to maximize product adoption in Chapter 8. These results are described in a WSDM 2012 paper [17]. Finally, we formulate and study of a novel problem – RECMAX which captures the problem of targeted advertising in recommender systems, in Chapter 9. It is based on our KDD 2012 paper [59].

While studies [60] and [61] were performed in collaboration with Wei Lu from University of British Colmubia, papers [56], [57] and [55] were published in collaboration with Francesco Bonchi from Yahoo! Research, Barcelona, Spain. The journal paper [58] was published in collaboration with Suresh Venkatasubramanian from University of Utah, along with Francesco Bonchi. The study [17] was performed in collaboration with Smriti Bhagat from University of British Columbia[1]. In [59], we did not collaborate with a third researcher. All these studies were performed under the supervision of Laks V. S. Lakshmanan.

[1] At the time of writing this dissertation, Smriti Bhagat was employed in Technicolor.

Table of Contents

List of Tables

List of Figures

Acknowledgments

One validation of the ideas about social influence described in this dissertation is my life at UBC. My life, thoughts and ideas have been deeply influenced by people around me. This dissertation would not have been in this shape without them. It is a great pleasure to express my sincere thanks here.

It is not easy to express my gratitude to my Ph.D. supervisor Dr. Laks V. S. Lakshmanan. He is an outstanding human being. He has remarkable enthusiasm for problem solving and research. He takes great effort in explaining things clearly and intuitively. He showed deep insights whenever I was at a loss and encouraged me whenever I felt down. He gave me complete freedom to explore the problems and research directions I wanted to explore. He never turned down my unsolicited visits and requests. It has been a lot of fun and honor to work with him.

Besides my advisor, I would like to thank Dr. Francesco Bonchi from Yahoo! Research, Barcelona. I was lucky to interact with him when he came to UBC during my early days in Ph.D. All my research works in initial years in Ph.D. was in collaboration with him. Even though he lives in a different continent in a different time zone, he is always available for any help or discussion. He keeps things simple and intuitive. He stays focused on the overall objective of the problem, while exploring various directions. My thoughts and writing style are deeply inspired by him. In a way, he has been my unofficial co-supervisor.

I have been lucky to collaborate with many smart and hard working scholars. In addition to Laks and Francesco, I got opportunities to collaborate with Wei Lu and Dr. Smriti Bhagat from UBC, and Prof. Suresh Venkatasubramanian from University of Utah. Wei is talented and very hard working chap. Weekdays or weekends, days or nights, he is always available, both for work and drinks. We spent many night outs in the lab and had a lot of fun working together. I am sure that he will do excellent job in his Ph.D. and have a successful career. Smriti is a fantastic collaborator. Brain-storming with her is so much fun. She is always cheerful. When it comes to work, she has many novel ideas. She has this rare ability to work for days continuously without needing rest or sleep. She expresses her thoughts clearly and is an extraordinary writer. I learnt a lot from her. Working with Suresh was another fruitful experience. He is a theoretician and brings fresh perspectives to the problems. He is very funny and explains complicated theoretical stuff intuitively.

I would like to thank my Ph.D. supervisory committee members – Raymond Ng and Arvind Gupta. They challenged me with insightful questions and gave me immensely helpful comments over the research direction. I also thank faculty members at the DMM lab at UBC: Raymond Ng and Rachel Pottinger. My research skills have been greatly influenced through close interaction with them by personal meetings, classes, and group seminars.

Three other people who have had deep influence on me are Naresh Kumar, Harman Bajwa,

and Pooja Mehra. We worked in a startup (called Namkis) together. We had several insightful discussions and I learnt a new perspective to look at things from them.

I thank other graduate students in the DMM lab at UBC for their friendship, feedback on my work and advises on various matters: Naresh Kumar, Wei Lu, Shailendra Agarwal, Min Xie, Pei Lee, Vignesh, Glenn Bevilacqua, Shealen Clare, Hongrae Lee, Solmaz Kolahi, Jian Xu, Wendy Hui Wang, Shaofeng Bu, Mohammad Tajer, Byung-Won On, Zhaohong Chen (Charles), Xueyao Liang (Sophia), Tianyu Li, Simona Radu, Yun Lou, Pooya Esfandiar, Ali Moosavi, Dibesh Shakya, April Webster.

Last, but most importantly, I am greatly indebted to my family. My Ph.D. has been a rather long journey since I decided to pursue an academic career. My parents, sisters and brother have been very understanding and supportive all the years. I also would like to thank my brother in law Puneet Gupta for his support and tips about life. My wife, Dolly, has always been very supportive. I was able to follow my passion since I knew I have home and loving family. Nothing would have been possible without their love, support and sacrifice. They are behind all I do.

Let noble thoughts come to us from every side.
– Rig Veda

Chapter 1

Introduction

Through the ages, humans' desire to communicate with each other has led to several extraordinary inventions. From cave drawings and developing alphabets to telephones, radios, printing press, televisions and computers, the technologies with which we communicate have evolved by leaps and bounds. Today, computers powered by internet have revolutionized the way we communicate with each other. In the process of communicating and sharing knowledge and ideas, we influence each other's conventions, beliefs, ideas, and ultimately behavior. The study of this *social influence* has long attracted the attention of psychologists, marketing scientists and even politicians. The last decade has seen scholars from computer science getting engaged with this fascinating field. Meanwhile, the emergence of Web 2.0 and social media platforms like Facebook[2] and Twitter[3] has greatly facilitated a large scale algorithmic and data mining study on social influence by providing massive amounts of data. The goal of this dissertation is to study social influence and its applications from an algorithmic and data mining perspective.

Social influence occurs when one's actions are affected by others. Rashotte defined social influence as change in an individual's thoughts, feelings, attitudes, and behaviors that results from interaction with another individual or a group [107]. Social influence takes many forms and can be seen in conformity, socialization, peer pressure, obedience, leadership, persuasion, sales, and marketing. Morton Deutsch and Harold Gerard described two psychological needs that lead humans to conform to the expectations of others. These include our need to be right, and our need to be liked [41]. If leveraged carefully, social influence can be exploited in many applications. In the field of data mining, some recognized applications include viral marketing (or targeted advertising in general), recommender systems, analysis of information diffusion in Social Media like Twitter and Facebook, events detection, experts finding, link prediction and ranking of feeds[4]. Other interesting problems like outbreak detection, human and animal epidemics, immunization are also related to the study of social influence.

Consider viral marketing for instance. Consider a social network where users perform various actions. As an example, a user in Twitter may decide to buy a new camera and decide to post a tweet about it. Here, posting a tweet is an action. As another example, a user in Yahoo! Movies[5] may decide to rate the movie *There will be blood*, which is also an action. As a third example, a user in Twitter may decide to ask her followers to support a campaign and spread the word. In each of these examples, users may choose to let their network friends see their actions. Seeing actions performed by their friends may make users curious and may sometimes tempt some fraction of the users to perform those actions themselves, some of the

[2] `http://www.facebook.com/`
[3] `http://www.twitter.com/`
[4] E.g. Facebook wall feeds
[5] `http://movies.yahoo.com/`

time. Informally, we can think of this as influence (for performing certain actions) propagating from users to their network friends, potentially recursively.

If such influence patterns repeat with some statistical significance, that can be of interest to companies, say for targeted advertising. For example, if we know that there are a small number of "leaders" who set the trend for various actions, then targeting them for adoption of new products or technology could be profitable to the companies. This kind of targeted advertising is called "Viral Marketing" [42, 78, 109].

> The idea behind viral marketing is to identify a small set of key influential "individuals" in a social network, who when convinced to adopt a product shall influence others in the network leading to a large number of adoptions.

In addition to viral marketing, social influence has been leveraged in other applications like recommender systems [7, 106, 120, 124, 125] analysis of information diffusion in Social Media like Twitter [13, 30, 98, 110, 131, 139, 142], events detection [29, 116], experts finding [43, 129], link prediction [11, 39, 54], and ranking of feeds [117]. Furthermore, patterns of influence can be taken as a sign of user trust and exploited for computing trust propagation [49, 64, 127, 143] in large networks and in P2P systems. In recent times, the notion of influence has been at the center-stage of many startups and companies. For example, Klout [6] and similarly, Peerindex [7] claim to measure social influence scores of users who integrate their Facebook and Twitter profiles with Klout (or Peerindex).

One of the fundamental problems in the study of social influence is the problem of **Influence Maximization**, motivated by the application in viral marketing: The problem, as originally defined by Kempe et al. [78], is as follows: given a social network represented by a directed graph with users as nodes, edges corresponding to social ties, edge weights capturing influence probabilities, and a budget k, the goal is to find a seed set of k users such that by targeting these, the expected influence spread (defined as the expected number of influenced users) is maximized. Here, the expected influence spread of a seed set depends on the influence diffusion process which is captured by a model for influence propagation, or ***propagation model***. In other words, the propagation model governs how influence diffuses or propagates through the network.

While several propagation models exist, two classical models, namely Linear Threshold (LT) and Independent Cascade (IC), have been widely studied in the literature. In both, at a given time, each node can be either active or inactive. Each node's tendency to become active increases monotonically as more of its neighbors become active, and an active node never becomes inactive again. Time unfolds in discrete steps. In the IC model, each active neighbor u of a node v has one shot at influencing v and succeeds with probability $p_{u,v}$, the probability with which u influences v. In the LT model, each node v is influenced by each neighbor u according to a weight $p_{u,v}$, such that the sum of incoming weights to v is no more than 1. Each node v chooses a threshold θ_v uniformly at random from $[0, 1]$. At any timestamp t, if the total weight from the active neighbors of an inactive node v is at least θ_v, then v becomes active at timestamp $t+1$. In both the models, the process repeats until no new node becomes

[6] `http://www.klout.com/`

[7] `http://www.peerindex.com/`

active. Given a propagation model m (e.g., IC or LT) and an initial seed set $S \subseteq V$, the expected number of active nodes at the end of the process is the *expected (influence) spread*, denoted by $\sigma_m(S)$.

1.1 Limitations and Open Problems

1.1.1 Inefficiency

One limitation with the framework proposed by Kempe et al. [78] is the inefficiency of the solution algorithms. They showed that while the Influence Maximization is NP-hard, the objective function (expected influence spread) under both independent cascade and linear threshold models is monotone and submodular. A set function $f : 2^U \rightarrow \mathbb{R}^+$ is monotone if $f(S) \leq f(T)$ whenever $S \subseteq T \subseteq U$. It is submodular if $f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$ for all $S \subseteq T$ and $w \in U \setminus T$. Intuitively, monotonicity indicates that as the size of seed set increases, the expected spread cannot decrease. Similarly, submodularity says the marginal gain $f(S \cup \{w\}) - f(S)$ from adding a new node w shrinks as the seed set grows. This property is also known as the *law of diminishing returns*.

Exploiting these properties, Kempe et al. [78] presented a simple greedy algorithm which repeatedly picks the node with the maximum marginal gain and adds it to the seed set, until the budget k is reached. However, computing exact marginal gain (or exact expected spread) under both the independent cascade and linear threshold models is #P-hard [35, 37]. Hence, it is estimated by running Monte Carlo simulations. This greedy algorithm, referred to as *simple greedy* henceforth, approximates the problem within a factor of $(1 - 1/e - \epsilon)$ for any $\epsilon > 0$. It is worth mentioning that in each run of Monte Carlo simulations, a possible world, representing a deterministic graph, is realized from the probabilistic graph. The spread of the seed set in a possible world is then the number of nodes reachable from the seed set (note that the spread includes the nodes in the seed set as well). Several such possible worlds are formed (typically 10,000) and the average of spread is taken to estimate the expected influence spread.

The simple greedy algorithm suffers from the following severe performance drawbacks: (i) The Monte Carlo simulations that are run sufficiently many times (typically 10,000) to obtain an accurate estimate of (expected) spread prove to be very expensive. (ii) The simple greedy algorithm makes $O(nk)$ calls to the spread estimation procedure (Monte Carlo in this case) where n is the number of users in the network and k is the size of the seed set to be picked (also known as budget). Unsurprisingly, the algorithm doesn't scale up to large real world social networks. As we show in Chapter 5, it may take even more than a month to select a seed set on a network of just 10,000 users.

1.1.2 Non-availability of influence probabilities

The framework employed in previous works, including by Kempe et al. [78] requires two kinds of data – a directed graph and an assignment of probabilities (or weights) to the edges of the graph, capturing degrees of influence. In real life, while the graph representing social network is often explicitly available, edge probabilities are not. To mitigate this, previous

works have resorted to simply making assumptions about these probabilities. The methods adopted for assignment of probabilities to edges include the following: (i) treating them as constant (e.g., 0.01), (ii) drawing values uniformly at random from a small set of constants, e.g., $\{0.1, 0.01, 0.001\}$ in the so-called trivalency "model", or (iii) defining them to be the reciprocal of a node's in-degree, in the so-called weighted cascade "model" (see e.g., [35, 36, 78]). The question of learning influence probabilities from real world data has largely been ignored.

1.1.3 Validation of propagation models against real world datasets

Most of the previous works revolve around two classical propagation models – independent cascade and linear threshold. Both the models, although very popular, have never been tested against real world datasets. It should be noted that Kempe et al. [78], who first introduced these models to the data mining community in their seminal paper, took these models from mathematical sociology. It is not known which propagation model is more appropriate under what conditions.

1.1.4 Alternative optimization criteria

The essence of Influence Maximization is to identify the quality seed users and recommend them to the marketers (decision makers) who would target them for the marketing of the given new product (e.g., by providing free samples or discounts). These marketers may wish to optimize other objectives, instead of maximizing influence. In the context of influence propagation, we can identify three orthogonal dimensions – the number of seed users activated at the beginning (also known as *budget*), the expected number of activated nodes at the end of the propagation (known as expected spread), and the *time* taken for the propagation. We can constrain one or two of these and try to optimize the third. In their seminal paper, Kempe, Kleinberg and Tardos [78] constrained the budget, left time unconstrained, and maximized the expected spread: this problem is known as Influence Maximization. While Influence Maximization has attracted a lot of attention, alternative optimization problems have rarely been studied. For instance, instead of constraining the budget and maximizing spread, we may wish to achieve a certain influence spread with the minimum possible budget. This captures the problem of deploying a viral campaign on a budget. Similarly, we may want to pick a seed set such that by targeting them, influence spreads as quickly as possible. This would address the issue of *timing* when deploying viral campaigns. Therefore, it is interesting to formulate and study alternative optimization problems.

1.1.5 Alternative solution frameworks

Likewise, before spending money, the marketer may wish to examine various solution frameworks instead of just the "optimization" framework. The overall goal of Influence Maximization is to identify small number of key individuals (or leaders) in a social network such that by targeting them, a large number of users get influenced eventually. To achieve this goal, Influence Maximization has been formulated as an optimization problem (described above). While this formulation is elegant and extremely popular, it is interesting to formulate the problem using alternative frameworks, say for example, a pattern mining framework.

1.1.6 Influence vs Product Adoption

Even though Targeted Advertising (or Viral Marketing) has been the primary motivation of the problem of Influence Maximization, it has rarely been studied directly. While the overall strategy of the two problems is same, that is, identify a small set of key individuals for targeting, there are important differences. For instance, previous works on Influence Maximization tacitly assume that a user who is influenced about a product necessarily adopts the product and encourages her friends to adopt it. However, an influenced user may not adopt the product herself, and yet form an opinion based on the experiences of her friends, and share this opinion with others. Furthermore, a user who adopts the product may not like it and hence not encourage her friends to adopt it to the same extent as another user who adopted and liked the product. This is independent of the extent to which those friends are influenced by her. Previous works do not account for these phenomena. Hence, it is important to distinguish adoption from influence.

1.1.7 Targeted Advertising in Recommender Systems

Previous works only account for influence which flows over explicit social links only. This may be suitable for social networks like Facebook or Twitter. For example, on Facebook, the wall feeds are the primary source of flow of information. These wall feeds are composed of actions performed by friends. Similarly, on Twitter, the tweets that are thrown to a user v are usually posted from the users that are followed by user v.

However, recently, a different type of "social networks" has emerged, known as *recommender systems*. In a short span of time, they have become extremely popular and successful. Some examples of recommender systems are Amazon[8] (a product recommendation engine), Youtube[9] (a video recommendation engine) and Netflix[10] (a movie recommendation engine). In recommender systems, items (or information) are presented to users based on the items (or information) that a user has acted upon earlier. The flow of information (or influence) in these social networks is not via explicit friendship links, but based on recommendation system models. Hence, targeted advertising in recommender systems require that these information channels are being taken in account.

1.2 Dissertation Goals

The ultimate goal of this dissertation is to take the algorithmic and data mining aspects of viral marketing out of the lab. While it is too optimistic to hope to solve all the problems inherent in this one dissertation, we aim to focus on some key problems. More precisely, we address the limitations and open problems mentioned above. We divide our goals in four categories as follows.

1. We identify inefficiency of Influence Maximization algorithms as one of key bottlenecks in applying them in real world settings, especially in the era of ever-increasing

[8] `http://www.amazon.com/`
[9] `http://www.youtube.com/`
[10] `http://www.netflix.com/`

scale of online social networks. Therefore, developing highly scalable algorithms for Influence Maximization is one of our goals.

2. Another issue we realized in the previous studies of social influence is the lack of data driven approaches. Even though the motivating scenarios are primarily inspired from real world marketing applications, the framework studied by Kempe et al. [78] and subsequent works [35–37, 79, 82] are theoretical and lack the validation against the real world data. A key goal of this dissertation is to investigate the approaches driven by the real world data.

3. As a third goal, we aim to improve the role that a human (marketer) plays while designing a viral campaign. A marketer, to make informed decisions, needs to play with various parameters and settings. To this end, we propose alternative problem formulations. There are three orthogonal dimensions in the context of influence propagation – budget, spread achieved and time. We can constrain one or two of these and try to optimize the third. Influence Maximization constrains the budget, leaves time unconstrained, and maximizes the expected spread. Other optimization problems are interesting to study. In addition, instead of seeing viral marketing as an optimization problem, a marketer may be interested in looking for "leaders" in general, and their followers (or tribes). Providing these flexible choices by formulating different problem statements and approaches is one of the goals of this dissertation.

4. Finally, we wish to investigate the motivating applications of the study of social influence, like viral marketing, and in general targeted advertising. As mentioned above, influence does not necessarily imply product adoption, and hence, maximizing product adoption instead of influence requires a different strategy. Moreover, with the emergence of different types of social networks, and in particular, recommender systems, the paths of flow of influence (or information) may not correspond to friendship links only. This complex flow of influence calls for different solution strategies. Our final goal in this dissertation is to study the applications of social influence, including in such unconventional social networks.

1.3 Key Contributions

Based on the goals mentioned above, we make the following key contributions in this dissertation (for detailed contributions, see the respective chapters). The contributions are summarized in Figure 1.1.

1.3.1 Improving the efficiency

Considerable work has been done to improve the scalability of influence maximization algorithms [35–37, 82, 91]. Exploiting the property of submodularity of the objective function, Leskovec et al. [91] developed a "lazy-forward" optimization known as Celf which improves the efficiency of the greedy algorithm proposed by Kempe et al. [78] by up to 700 times. Moreover, various efficient heuristics (which do not guarantee any worst case approximation

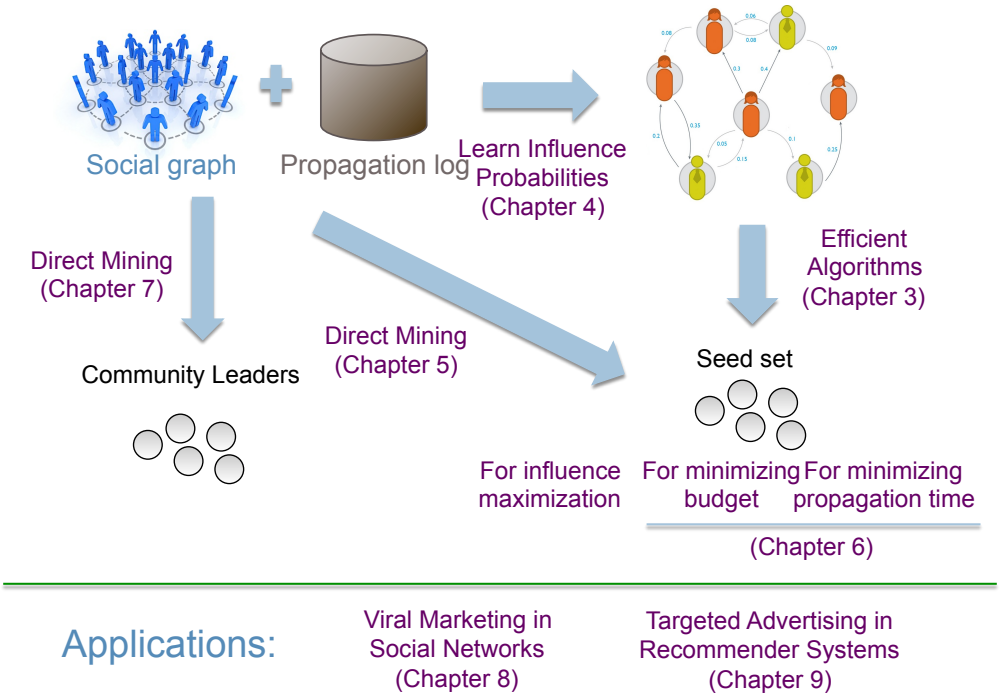


Figure 1.1: Summary of our contributions.

factor, but work well in practice) have been proposed, particularly for the independent cascade model [35, 36, 82]. On the other hand, relatively less work has been done for the linear threshold model [37].

CELF++. In Section 3.1, we introduce CELF++ that further optimizes CELF by exploiting submodularity. Our experiments show that it improves the efficiency of CELF by 17-61%. Since the optimization introduced in CELF++ is orthogonal to the method used for estimating the influence spread, our idea can be combined with the heuristic approaches that are based on the greedy algorithm to obtain highly scalable algorithms for influence maximization.

SIMPATH. We also develop SIMPATH, an efficient and effective heuristic for influence maximization under the linear threshold model. While CELF++ is an optimization that has no trade-offs with the quality of seed sets obtained, SIMPATH is a heuristic. Moreover, it is designed specifically for linear threshold model. We show through extensive experiments on four real datasets that our SIMPATH algorithm is more efficient, consumes less memory and produces seed sets with larger spread of influence than LDAG heuristic [37], the previous state of art algorithm (Chapter 3).

1.3.2 Learning from data

Learning Influence Probabilities in Social Networks. In Chapter 4, we attack the problem of learning influence probabilities from real world data. We propose models and algorithms for learning the model parameters. In addition, we also develop techniques for

predicting the time by which a user may be expected to perform an action. We show that influence probabilities are not static and may change over time. We introduce the metrics of user influenceability and action influence quotient. Users (actions) with high values for this metric do experience genuine influence compared to those with low values. We test our models and algorithms on the Flickr data set where for actions, we take users joining online communities. The data set consists of a graph with 1.3M nodes and more than 40M edges and an action log with 35M tuples referring to 300K different actions.

A Data-Based approach to Social Influence Maximization. Next, in Chapter 5, we perform a large scale experimental study to test various propagation models against real world datasets. We found that classical propagation models like independent cascade (IC) and linear threshold (LT) models grossly overpredict the spread of influence in most cases. On account of this, we develop a data driven framework for the problem of Influence Maximization. In particular, we introduce a new model, which we call credit distribution, that directly leverages available propagation traces to learn how influence flows in the network and uses this to estimate expected influence spread. Our approach also learns the different levels of influenceability of users, and it is time-aware in the sense that it takes the temporal nature of influence into account. We show that credit diffusion model is way more accurate than IC or LT model in modeling influence propagation.

We show that influence maximization under the credit distribution model is NP-hard and that the function that defines expected spread under our model is submodular. Based on these, we develop an approximation algorithm for solving the influence maximization problem. We also build a highly scalable algorithm for Influence Maximization under credit distribution model (Chapter 5).

1.3.3 Alternative problem formulations and frameworks

Alternative optimization problems – MINTSS and MINTIME. In Chapter 6, we study a couple of alternative optimization problems which are naturally motivated by resource and time constraints on viral marketing campaigns. In the first problem, termed *Minimum Target Set Selection* (or Mintss for short), an expected spread threshold η is given and the task is to find the *minimum size seed set* such that by activating it, at least η nodes are eventually activated in the expected sense. This naturally captures the problem of deploying a viral campaign on a budget. In the second problem, termed Mintime, the goal is to minimize the time in which a predefined spread is achieved. More precisely, in Mintime, a spread threshold η and a budget threshold k are given, and the task is to find a seed set of size at most k such that by activating it, at least η nodes are activated in the expected sense, *in the minimum possible time*. This problem addresses the issue of *timing* when deploying viral campaigns. Both these problems are NP-hard, which motivates our interest in their approximation.

For Mintss, we develop a simple greedy algorithm and show that it provides a bicriteria approximation. We also establish a generic hardness result suggesting that improving this bicriteria approximation is likely to be hard. For Mintime, we show that even bicriteria and tricriteria approximations are hard under several conditions. We show, however, that if we allow the budget for number of seeds k to be boosted by a logarithmic factor and allow the spread to fall short, then the problem can be solved *exactly* in PTIME, i.e., we can achieve the

required spread within the time achieved by the optimal solution to MINTIME with budget k and coverage threshold η (Chapter 6).

A Pattern Mining framework. The overall goal of INFLUENCE MAXIMIZATION is to identify small number of key individuals (or leaders) in a social network such that by targeting them, a large number of users get influenced eventually. To achieve this goal, INFLUENCE MAXIMIZATION has been formulated as an optimization problem (described above). While this formulation is elegant and extremely popular, it is interesting to formulate the problem from alternative perspectives.

In Chapter 7, we introduce a novel frequent pattern mining approach to discover leaders and tribes in social networks. We consider alternative definitions of leaders based on frequent patterns and develop algorithms for their efficient discovery. In particular, we consider social networks where users perform actions. Actions may be as simple as tagging resources (urls) as in Delicious[11], rating songs as in Yahoo! Music[12], or movies as in Yahoo! Movies[13], or users buying gadgets such as cameras, handhelds, etc. and blogging a review on the gadgets.

Intuitively, we want to think of user u as a leader w.r.t. an action a provided u performed a and within a chosen time bound after u performed a, a sufficient number of other users performed a. Furthermore, we require that these other users are reachable from u thus capturing the role social ties may have played. Finally, we may want to extract leadership w.r.t. performing actions in general or performing a class of actions. In other words, we say that a user u is a leader if sufficiently many actions performed by her have been followed by sufficiently large number of other users (reachable from u in the propagation trace) in a sufficiently small time window. Similarly, we say that a leader u is a "tribe-leader" if the set of influenced users (users who follow u's actions) remains the same across various actions performed by u. The user u along with the set of these influenced users is termed as a "tribe".

We develop efficient algorithms for extracting various kinds of leaders from an input data consisting of a social graph and a table of user actions. We demonstrate the utility and scalability of our algorithms, via an extensive set of experiments on a real-world dataset obtained by joining data coming from Yahoo! Messenger[14] (the social graph) and Yahoo! Movies (the actions log, where the action corresponds to rating a movie).

1.3.4 Applications of social influence

Applications in Viral Marketing. In Chapter 8, we argue that it is important to distinguish product adoption from influence. We propose a model that factors in a user's experience (or projected experience) with a product. We adapt the classical Linear Threshold (LT) propagation model by defining an objective function that explicitly captures product adoption, as opposed to influence. We show that under our model, adoption maximization is NP-hard and the objective function is monotone and submodular, thus admitting an approximation algorithm. We perform experiments on three real popular social networks – Flixster, Movielens[15]

[11] http://www.delicious.com/
[12] http://www.music.yahoo.com/
[13] http://www.movies.yahoo.com/
[14] http://www.messenger.yahoo.com/
[15] http://movielens.umn.edu/

and Last.fm[16] and show that our model is able to distinguish between influence and adoption, and predict product adoption much more accurately than the classical LT model.

Targeted advertising in Recommender Systems. In Recommender Systems, the flow of information (or influence) happens not via explicit friendship links, but based on recommendation system models. Hence, targeted advertising in recommender systems require these information channels to be taken in account. In Chapter 9, we propose a novel problem RECMAX (short for recommendation maximization): Given a recommender system, a budget of k and a new product ρ, find k users (called seed set) to whom ρ should be marketed such that if they endorse the product by providing relatively high ratings, the number of other users to whom the product is recommended by the underlying recommender system algorithm is maximum.

We motivate RECMAX with real world applications. We show that seeding can make a substantial difference, if done carefully. We prove that RECMAX is not only NP-hard to solve optimally, it is NP-hard to even approximate within any reasonable factor. Given this hardness, we explore several natural heuristics on 3 real world datasets – Movielens, Yahoo! Music [17] and Jester Joke [18] and report our findings. We show that even though RECMAX is hard to approximate, simple natural heuristics may provide impressive gains, for targeted marketing using recommender systems.

1.4 Outline

The rest of this dissertation is organized as follows. In Chapter 2, we provide a brief background and review related work. Chapter 3 corresponds to the first goal – that is, to improve the efficiency and scalability of INFLUENCE MAXIMIZATION algorithms. In Chapter 4, we address the problem of learning influence probabilities, while in Chapter 5, we introduce a data driven approach to INFLUENCE MAXIMIZATION. Both these chapters are related to our second goal which is to validate social influence propagation against real world datasets. The next two chapters – Chapter 6 and Chapter 7 are associated with our third goal which is to formulate different but important problem statements and approaches aimed at providing marketer(s) flexible choices. Chapter 6 proposes two alternative problems – MINTSS and MINTIME and in Chapter 7, we introduce a pattern mining approach to discover community leaders. Our final goal is to study the applications of social influence which we perform in the next two chapters. In Chapter 8, we show that influence is different from product adoption and propose a model for product adoption. In Chapter 9, we apply targeted advertising in recommender systems. Finally, in Chapter 10, we conclude and present future work.

[16] http://www.last.fm/
[17] http://www.music.yahoo.com/
[18] http://eigentaste.berkeley.edu/user/

Chapter 2

Background and Related Work

Work in influence propagation spans the fields of statistics, economics, computer science, sociology, psychology and marketing. It is an extremely difficult task to review the studies from all these fields, and we do not dare to do it in this dissertation. Our work is primarily inspired from the application of Viral Marketing and this related work review mainly focuses on the studies done in computer science field, meanwhile we provide pointers to various surveys performed by other researchers.

2.1 Background

The study of the spread of influence through a social network, i.e., the extent to which people are likely to be influenced by actions of their friends has a long history in the social sciences. The first investigations focused on the adoption of medical [38] and agricultural innovations [132]. Later marketing researchers have investigated the diffusion process of the "word-of-mouth" effect and its application to the so-called viral marketing [15, 51, 95]. We recommend readers to read [90] for a survey.

The first work to consider the propagation of influence and the problem of identification of influential users by a data mining perspective is [42, 109] (see [20] for a survey on social influence from a data mining perspective). Later, Kempe et al. [78] formulated the problem of Influence Maximization as an optimization problem, as described above. Kempe et al. [78] mainly focus on two propagation models – the Independent Cascade (IC) and the Linear Threshold (LT) models. In both, at a given time, each node can be either active or inactive. Each node's tendency to become active increases monotonically as more of its neighbors become active, and an active node never becomes inactive again. Time unfolds in discrete steps.

In the IC model, each active neighbor u of a node v has one shot at influencing v and succeeds with probability $p_{u,v}$, the probability with which u influences v. In the LT model, each node v is influenced by each neighbor u according to a weight $p_{u,v}$, such that the sum of incoming weights to v is no more than 1. Each node v chooses a threshold θ_v uniformly at random from $[0, 1]$. At any timestamp t, if the total weight from the active neighbors of an inactive node v is at least θ_v, then v becomes active at timestamp $t + 1$. In both the models, the process repeats until no new node becomes active. Given a propagation model m (e.g., IC or LT) and an initial seed set $S \subseteq V$, the expected number of active nodes at the end of the process is the *expected (influence) spread*, denoted by $\sigma_m(S)$.

The input to the Influence Maximization problem is a directed and edge-weighted social graph $G = (V, E, p)$, where V represents the set of users/nodes, E represents the set of

edges/links among users and p represents the weights on the edges such that $p_{u,v}$ denote the influence probability on the edge $(u, v) \in E$, Formally, the problem is defined as follows.

Problem 1 (Influence Maximization). *Given a directed and edge-weighted social graph $G = (V, E, p)$, a propagation model m, and a number $k \leq |V|$, find a set $S \subseteq V$, $|S| \leq k$, such that $\sigma_m(S)$ is maximum.*

Under both the IC and LT propagation models, this problem is shown to be NP-hard [78]. Kempe et al., however, showed that the function $\sigma_m(S)$ is monotone and submodular. A function f from sets to reals is monotone if $f(S) \leq f(T)$ whenever $S \subseteq T$. A function f is submodular if $f(S+w)-f(S) \geq f(T+w)-f(T)$ whenever $S \subseteq T$.[19] Intuitively, monotonicity indicates that as the size of seed set increases, the expected influence spread cannot decrease. Submodularity intuitively says an active node's probability of activating some inactive node v does not increase if more nodes have already attempted to activate v and v is hence more "marketing-saturated". It is also called the law of "diminishing returns".

For any monotone submodular function f with $f(\emptyset) = 0$, the problem of finding a set S of size k such that $f(S)$ is maximum, can be approximated to within a factor of $(1 - 1/e)$ by a greedy algorithm [103], a result that directly carries over to the influence maximization problem [78] (see Algorithm 1).

The complex step of the greedy algorithm is in line 3, where we select the node that provides the largest marginal gain $\sigma_m(S+w)-\sigma_m(S)$ with respect to the expected spread of the current seed set S. Computing the expected spread given a seed set is #P-hard under both the IC model [35, 72] and the LT model [37]. In their paper, Kempe et al. run MC simulations of the propagation model for sufficiently many times (the authors report $10,000$ trials) to obtain an accurate estimate of the expected spread, resulting in a very long computation time. Clearly, accuracy improves as more simulations are employed. Kempe et al. claimed that for any fixed $\phi > 0$, there is a $\delta > 0$ such that using $(1 + \delta)$-approximate values of the expected influence spread, we can obtain a $(1 - 1/e - \phi)$-approximation for the Influence Maximization problem. Finally, due to Hoeffding Inequality, it can be calculated how many monte carlo simulations are required in order to achieve $(1 + \delta)$-approximate values of the expected influence spread, for any fixed $\delta > 0$ [72][20]. Long story short, it is possible to find a seed set S of size k such that $\sigma_m(S) \geq (1 - 1/e - \phi) \cdot \sigma_m(S^*)$ where S^* is the optimal seed set of size k.

2.2 Improving the efficiency

We now present related work that has been done from other researchers for improving the efficiency of the greedy algorithm. Note that there are two sources of inefficiency in the greedy algorithm: (i) The monte carlo simulations that are run sufficiently many times to obtain an accurate estimate of influence spread prove to be very expensive. (ii) The simple greedy

[19] In the rest of the thesis, we write $S + w$ in place of $S \cup \{w\}$ and similarly $S - T$ in place of $S \setminus T$.

[20] In practice, it is found that usually 10,000 trials of monte carlo simulations are enough for the accurate estimate.

Algorithm 1 Greedy

Input: G, k, σ_m
Output: seed set S

1: $S \leftarrow \emptyset$
2: **while** $|S| < k$ **do**
3: $u \leftarrow \arg\max_{w \in V-S}(\sigma_m(S+w) - \sigma_m(S))$;
4: $S \leftarrow S + u$
5: **end while**

algorithm makes $O(nk)$ calls to the spread estimation procedure (monte carlo in this case) where n is the number of nodes in the graph and k is the size of the seed set to be picked.

Leskovec et al. [91] exploited submodularity to develop an efficient algorithm based on a "lazy-forward" (known as CELF) optimization in selecting new seeds. The idea is that the marginal gain provided by a node in the current iteration cannot be better than the marginal gain provided by it in the previous iterations. For the sake of completeness, we provide the pseudo-code of the greedy algorithm, when optimized with CELF (Algorithm 2).

Algorithm 2 Greedy optimized with CELF

Input: G, k, σ_m
Output: seed set S

1: $S \leftarrow \emptyset$; $Q \leftarrow \emptyset$.
2: /* First iteration */
3: **for** each $u \in V$ **do**
4: $u.mg = \sigma_m(\{u\})$; $u.flag = 0$.
5: Add u to Q in decreasing order of mg.
6: **end for**
7: /* CELF */
8: **while** $|S| < k$ **do**
9: u = top (root) element in Q.
10: **if** $u.flag == |S|$ **then**
11: $S \leftarrow S \cup \{u\}$.
12: $Q \leftarrow Q - \{u\}$.
13: **else**
14: $u.mg = \sigma_m(S \cup \{u\}) - \sigma_m(S)$.
15: $u.flag = |S|$.
16: Reinsert u into Q and heapify.
17: **end if**
18: **end while**

We use $\sigma_m(S)$ to denote the expected influence spread of seed set S under the propagation model m (like IC or LT). The optimization works as follows. Maintain a heap Q with nodes corresponding to users in the network G. The node of Q corresponding to user u stores a tuple of the form $\langle u.mg, u.flag \rangle$. Here $u.mg = \sigma_m(S \cup \{u\}) - \sigma_m(S)$, the marginal gain of u w.r.t. the current seed set S and $u.flag$ is the iteration number when $u.mg$ was last updated. In the first iteration, marginal gains of each node is computed and added to Q in decreasing order of marginal gains (lines 1-6). Later, in each iteration, look at the top node u in Q and see if

its marginal gain was last computed in the current iteration (using the $flag$ attribute). If yes, then, due to submodularity, u must be the node that provides maximum marginal gain in the current iteration, and hence, it is picked as the next seed (lines 9-12). Otherwise, recompute the marginal gain of u, update its flag and re-insert it in the Q such that the order of marginal gains is maintained (lines 13-16).

It is easy to see that this optimization avoids the recomputation of marginal gains of all the nodes in any iteration, except the first one. The CELF optimization results in 700 times speedup in the greedy algorithm as shown by experiments in [91]. In Section 3.1, we propose CELF++ that further optimizes CELF by exploiting submodularity. Our experiments show that it improves the efficiency of CELF by 17-61%.

In spite of this big improvement over the basic greedy algorithm, greedy with CELF still faces serious scalability problems as shown in [36]. In that paper, Chen et al. propose *degree discount* heuristic that produces influence spread close to that of the greedy algorithm but more efficiently. However, their method is limited to IC (independent cascade) model with uniform probabilities, that is, all edges have the same probability.

In [81, 82], Kimura et al. proposed two shortest path based models – SPM and SP1M, which are special cases of the IC (independent cascade) model. Consider two nodes u and v, then the heuristic SPM only considers the influence that flows via the shortest path from u to v. SP1M, instead considers the top-2 shortest paths from u to v. The intuition is that the majority of influence flows through shortest paths. Not surprisingly, SP1M performs better than SPM. A critical issue with this approach is that it completely ignores the influence probabilities among users.

Chen et al. [35] extended this idea by considering Maximum Influence Paths instead of shortest paths. A maximum influence path between a pair of nodes (u, v) is the path with the maximum propagation probability from u to v. Based on this, the authors propose two models: *maximum influence arborescence* (MIA) model and its extension *prefix excluding* MIA (PMIA) model. We observe in Chapter 6 that these heuristics perform badly on high influence graphs, that is, when the influence probabilities through links are large.

Wang et al. [135] proposed an alternative approach. They argue that most of the diffusion happens only in small communities, even though the overall networks are huge. Taking this as an intuition, they first split the network in communities, and then using a greedy dynamic programming algorithm, they select seed nodes. To compute the marginal gain of a prospective seed node, they restrict the influence spread to the community to which the node belongs. The focus on their study was on IC model. Both papers – by Wang et al. [135] and by Chen et al. [35] appeared in the same conference in the same year (KDD 2010) and hence, it remains unclear which of these two have better performance.

For LT (linear threshold) model, relatively less work has been done. Narayanam et al. [102] proposed an alternative approach for LT model based on Shapely value. Their algorithm SPIN (ShaPely value-based Influential Nodes) is slower than the greedy algorithm and thus not scalable [37]. Chen et al. [37] proposed LDAG heuristic for LT model. They observe that while computing the expected spread is #P-hard in general graphs, it can be computed in linear time in DAGs (directed acyclic graphs). They exploit this property by constructing local DAGs (LDAG) for every node in the graph. A LDAG for user v contains the nodes that have significant influence over v (more than a given threshold θ). Based on this idea,

they propose a heuristic called LDAG which provides close approximation to Greedy algorithm and is highly scalable. In Chapter 3, we propose SIMPATH heuristic for LT model and show through extensive experiments on four real datasets that our SIMPATH algorithm is more efficient, consumes less memory and produces seed sets with larger spread of influence than LDAG.

2.3 Modeling Influence Propagation

2.3.1 Learning Propagation Probabilities

We attack the problem of learning influence probabilities in Chapter 4. Our work was one of the initial works to address this problem. After that, a lot of work has been done to model diffusion probabilities, propagation traces, inferring missing propagation networks and other related modeling issues. In this section, we perform a brief survey of these recent works.

Saito et al. [114] focused on learning propagation probabilities under the IC model. They formally define the likelihood maximization problem and then apply EM (Expectation Maximization) algorithm to solve it. While their formulation is elegant, there are two issues in their approach. First, since EM is an iterative algorithm, it may not be scalable to very large social networks. This is due to the fact that in each iteration, the EM algorithm must update the influence probability associated to each edge. Second, the propagation traces data (log of actions – actions that various users perform in a social network, including timestamps) that is used as input to learn probabilities is very sparse – in particular, it follows a long tail distribution, that is, most of the users perform very few actions. As a result, the EM algorithm is vulnerable to overfitting and may result in bad quality seed sets, as we show in Chapter 5.

Later, Saito et al. [112, 113] extended the IC and LT models to make them time-aware and proposed methods to learn influence propagation probabilities for these extended models. The authors argue for incorporating time delay in action propagations where the time delay is on a continuous time scale, for IC model in [112] and for LT model in [113]. Their argument is in line with our work in Chapter 4 where we show that influence probabilities decrease over time, and hence, a time delay parameter is necessary in modeling the influence propagation. Saito et al. use EM based approaches to learn propagation probabilities, like in their previous work [114], described above. In a recent paper, the same set of authors recognizes the issue of overfitting (as we explain above) that comes with this approach [115]. In [115], the authors propose to consider node attributes as well in learning probabilities.

Cao et at. [27] argue that above mentioned works assume that propagation traces (which user perform what action at what time) which are usually available to social network platforms and not to the outsiders, mainly due to privacy reasons. They proposed "active learning" framework for learning influence probabilities under linear threshold (LT) model. In active learning framework, the information diffusion data is constructed strategically by sending free products to some users and monitor how their social neighbors react. One of the goals in their study is to design techniques such that the influence probabilities are learnt in as few free products as possible, while at the same time, these probabilities are useful for INFLUENCE MAXIMIZATION. They proposed a weighted sampling algorithm and show that it performs better than a baseline random sampling algorithm. In the process, they show an interesting

result that Influence Maximization problem is sensitive to influence probabilities in LT model.

2.3.2 Missing Information and Sparsification

Gomez-Rodriguez et al. [53] highlighted the issue of missing and unobserved information in diffusion data. They argue that even though it is possible to directly detect whether and when a node has become activated (or infected), observing individual transmissions (i.e., who infected whom or who influences whom) is typically very difficult. Moreover, in many applications, the underlying diffusion paths are unobserved. They formalized the problem as an optimization problem of finding k best edges that maximizes the likelihood of the observed data. The problem is NP-hard and the authors proposed approximation algorithms. Their modeling builds on independent cascade model.

Sadikov et al. [111] studied the problem of estimating properties of a target cascade (or propagation tree) $\mathcal{C}$, given only its fraction $\mathcal{C}'$, obtained by uniform sampling of $\mathcal{C}$ nodes. They proposed a k-tree model of cascade to predict the missing data such as the size and depth of the propagation tree.

Gomez-Rodriguez [52] focus on the temporal dynamics of diffusion networks. More precisely, given the cascade data, they estimate the transmission rates of each edge that best explain the observed data.

Yang et al. [142] on the other hand, model the propagation without requiring the information about the social graph. They proposed Linear Influence Model, where the rate of adoption is modeled as depends on which other nodes got infected in the past. Their model is similar of Bass' model [15], described below in Section 2.4, in a way that both ignore the social graphs.

Mathioudakis et al. [97] proposed an alternative problem of finding k most important links in a network that maximizes the likelihood of the observed propagations. They refer to the problem as "sparsification of influence networks". The idea is that by keeping only the most important links in an influence network one can identify the backbone of the social influence propagation. This can be useful in qualitative analysis of influence propagation. Sparsification can also be employed in optimizing the algorithms for analyzing influence propagation.

Although inferring missing data, or most important parts of data in the context of influence propagation is a related and an interesting problem, we do not study it in this dissertation.

2.3.3 Topic-based Influence Propagation

Tang et al. [129] introduce the problem of topic-based social influence analysis. Given a social network and a topic distribution for each user, the problem is to find topic-specific subnetworks, and topic-specific influence weights between members of the subnetworks. They propose a Topical Affinity Propagation (TAP) approach using a graphical probabilistic model. They also deal with the efficiency problem by devising a distributed learning algorithm under the Map-reduce programming model.

Lie et al. [94] propose a generative graphical model which utilizes the content and link information associated with each node (here, a node can be a user, or a document) in the

network to mine topic-level direct influence.

Woo et al. [141] studied the diffusion of violent topics (that deals with terrorism and extremism) in web forums to identify the exposure process of extreme opinions. The adapt the SIR propagation model (described below in Section 2.4) and apply to a major international Jihadi forum that expounds extreme ideology. One of the interesting finding they report is that "anti-America" is the most infectious topic while "wear-hijab" is the least infectious topic.

Our work in Chapter 8 is related to topic-modeling. Although, we do not explicitly model the topics, we use collaborative filtering data to model user opinions in product adoption diffusion. This indirectly accounts for topic diffusion.

2.3.4 Does Influence exist? Influence vs Homophily

In addition to social influence, *homophily* (or *selection*) has been shown to be a possible cause of users' actions. Homophily is defined as the tendency of individuals to associate and bond with similar others. The presence of homophily has been discovered in a vast array of network studies. In 2001, within their extensive review paper, sociologists McPherson, Smith-Lovin and Cook [99] cite over one hundred studies that have observed homophily in some form or another. These include age, gender, class, organizational role, and so forth. In particular, Watts challenges the very notion of influential users that are often assumed in viral marketing papers [13, 136–138].

In data mining community as well, several studies has shown the existence of homophily [6, 8, 39, 88, 122]. While some [6, 8] focus on distinguishing the effects of influence and homophily, others [39, 88] showed the feedback effects that both factors have on each other. Crandall et al. [39], in particular, showed that both influence and homophily can be useful in predicting users' future behavior.

Huang et al. [71] empirically analyzed the effects of word-of-mouth recommendations on real-world online social networks and content sharing communities. They show strong impact that word-of-mouth recommendations can have on users' behavior.

Hill et al. [68] provides an excellent survey on word-of-mouth marketing methods with an emphasis on the statistical methods and the data used to which these methods are being applied. They cite several studies which show the effect of influence in product adoption. In addition, they show, empirically, that users who are connected to active neighbors (neighbors who have already influenced w.r.t. a product), are 3-5 times more likely to adopt the product. One popular case study they talked about is "Hotmail" (`www.hotmail.com`). The hotmail free email service started in July 1996, and by early 1997, it had over 1 million subscribers [101].

Manchanda et al. [96] study the effect of social influence in the context of pharmaceutical industry, with their data containing the launch of a new drug. The two factors their study considered in modeling physician's decision to adopt the new drug were general marketing and the social contagion. They found that (general) marketing plays a large (relative) role in affecting adoption early on. However, the role of contagion dominates from month 4 onwards and, by month 17 (or about half the duration of our data), asymptotes to about 90% of the effect. Another recent and popular study that shows the presence of influence is done by Iyengar et al. [73].

One of the goals in the study of social influence and in particular influence maximization is

to build models that can predict users' actions, without necessarily distinguishing the causes. Therefore, even if homophily and not influence is the primary cause of a user's action, it is often helpful to model it in the propagation/diffusion probabilities in prediction.

2.3.5 Discussion

Above, we briefly described the studies that have been performed to model the influence propagation. In this dissertation, we also spend substantial efforts in modeling influence and product adoption (see Chapters 4, 5 and 8). Even after all these works, there is still a lot of work that needs to be done. For example, we still do not know what are the best methods to learn influence probabilities in case of LT model. Even for IC model, we do not know how to solve over-fitting issues in the methods proposed by Saito et al. [114]. Furthermore, the kind of datasets that are available from different sources are usually different. For example, in some datasets, we are aware of social graph while in other, the social graph is not even present and the information propagates through implicit links. Similarly, in some cases, we are aware of timestamps at which users perform actions, and in other cases, we are not aware of these timestamps. Finally, on some datasets, it may be possible that influence plays a significant role while on other datasets, homophily is a major driver of users' behavior. All of these issues have made the task of modeling influence propagation very difficult.

2.4 Alternative Propagation Models

In addition to IC and LT models, various other propagation models have been proposed to model influence spread. One of the first and popular was proposed by Bass [15]. Bass said that product diffusion follows a S-shaped curved, where adoption is slow at first, takes off exponentially and flattens at the end. He modeled rate of adoption as a function of users who have already adopted the product. It does not explicitly account for the structure of the social network.

Another popular propagation model is SIR – susceptible, infected, recovered. SIR has been extensively studied in the context of disease propagation where a person is first susceptible to the disease, then she can become infected with a probability β if exposed to the infection (that is, to an infected neighbor in the social network). Subsequently, the person recovers at rate r. Once a person recovers, she becomes immune to the disease. In the context of social influence, some studies that focused on SIS models (and its extensions) are [83, 138, 141] among others.

Kempe et al. [78], in addition to LT and IC models, also proposed a broader framework that simultaneously generalizes the LT and IC models, and that has equivalent formulations in terms of thresholds and cascades. In the General Threshold Model, each node v has a monotone activation function $f_v : 2^{N(v)} \rightarrow [0, 1]$, from the set of neighbors N of v, to real numbers in $[0, 1]$, and a threshold θ_v, chosen independently and uniformly at random from the interval [0,1]. A node v becomes active at time $t + 1$ if $f_v(S) \geq \theta_v$, where S is the set of neighbors of u that are active at time t.

Decreasing Cascade Model is similar to the Independent Cascade Model, except that it allows the probability that u succeeds in activating a neighbor v to depend on the set S of v's

neighbors that have already tried. Thus, an incremental function $p_v(u, S)$ is defined where S and $\{u\}$ are disjoint subsets of v's neighbor set.

LT (Linear Threshold) model is a special case of General Threshold model and IC (Independent Cascade) model is a special case of Decreasing Cascade model. Kempe et al. [78] showed that General Threshold and Decreasing Cascade models have the same expressive power. That is, every instance of one model has an equivalent instance of other model. Note that even though these "general" models are equivalent, the special case models LT and IC remains two distinct models.

2.4.1 Discussion: Model-based approach vs Memory-based approach

Most of the approaches to identify influential users in social networks can be classified as model-based approaches. That is, a propagation model is assumed that is supposed to capture the flow of influence. The parameters of the model are then learnt from the observed propagation traces. Finally, to estimate the influence spread of a user, the propagation model is used.

In Chapter 5, we propose an alternative memory-based approach to estimate influence spread of users. We show empirically that our memory based approach is more accurate in estimating influence spread, than popular model based approaches. The discussion of model-based approach vs memory based approach is similar in spirit with that in recommender systems [108].

2.5 Social Influence Analysis in Blogspace and Twitter

Considerable work has been done on analyzing social influence on blogs and micro-blogs, particularly on Twitter (`http://twitter.com`), perhaps due to easiness of collecting data from Twitter.

Agarwal et al. [3] investigated the problem of identifying the influential bloggers in a community. They showed that most influential bloggers are not necessarily most active Gruhl et al. [63] analysed the information diffusion in blogspace by characterizing the individuals and the topics of the blog postings. In [53], the authors look into the problem of inferring networks of diffusion and influence in blogspace.

On Twitter, Cha et al. [30] compared three different measures of influence – indegree (number of followers), retweets and user mentions. They observed that users who have large number of followers are not necessarily influential in terms of spawning retweets or mentions.

Weng et al. [139] discovered that almost 72.4% of the users in Twitter follow more than 80% of their followers. They attributed the cause of this "reciprocity" to homophily. Based on this observation, they proposed a topic-sensitive PageRank measure for influence in Twitter. However, Cha et al. [30] has shown that the reciprocity is low overall in Twitter and contradicted the assumptions of this work.

Romero et al. [110] showed that the majority of users act as passive information consumers and do not forward the content to the network. They proposed techniques to learn users' passivity. They argue that information about passivity of users in a network may be useful from the perspectives of viral marketing.

Bakshy et al. [13] found that the largest cascades tend to be generated by users who have been influential in the past and who have a large number of followers. They also reported that predictions of which particular user or URL will generate large cascades are relatively unreliable. Consequently, they concluded that word-of-mouth diffusion can only be harnessed reliably by targeting large numbers of potential influencers, thereby capturing average effects. The authors made these claims based on the prediction model they employed which is regression trees. It remains unclear whether other modeling approaches would improve the prediction accuracy.

Tsur et al. [131] incorporated the contents of the tweets, which they derive from hashtags, to predict the spread of a tweet. They show that a combination of content features with temporal and topological features reduces the prediction error.

Our dissertation does not specifically analyze influence on Twitter, but the techniques we propose are fairly general in nature and can be applied to Twitter propagation trees as well.

2.6 Revenue Maximization

Instead of optimizing influence, a couple of studies [10, 65] focus on optimizing the revenue by explicitly considers pricing models for products.

Hartline et al. [65] propose a family of strategies called *influence-and-exploit* strategies that work as follows. In the first step (*influence* step) the seller gives the item to a chosen set of customers for free. Then in the *exploit* step, the seller visits the remaining buyers in a random sequence and offers them a price in order to maximize the revenue. The assumption is that the decision of the buyers depends on the customers chosen in the first step and the price offered to them. Furthermore, buyers themselves do not influence each other. They propose various settings and in the general case, the problem is NP-hard and approximation algorithms are developed.

Later, Arthur et al. [10] also work on a similar problem. Their model is inspired by the independent cascade model and work as follows. If a node u buys a product, say at time step t, it recommends the product to all its inactive neighbors. At that time, the seller offers a different price to each of the inactive neighbor. If the inactive neighbor decides to buy the product, a fixed cashback reward is given to the recommender. If there are multiple recommenders, then the buyer must choose only one of them for the cashback reward. The process is repeated until there are no more activations. In such settings, finding a seller strategy that maximizes the revenue is NP-hard. The authors develop approximation algorithms based on influence-and-exploit idea. Their model is different from Hartline's model in two ways: a) The seller is not allowed to pick an initial seed set, instead the seed set is given as input; b) In Hartline's model, the seller is allowed to visit the nodes in an arbitrary manner (in the exploit step), their model make use of the network structure and the timing of an offer made to a customer depends on the cascade of recommendations.

Singer [121] approached this problem as the question: *How do we influence the influencers?*. That is, how to ensure that the seed users that are selected by the influence maximization algorithms are activated? The author argued that there is a cost attached to activate the seed users and focus on identifying it.

In Chapter 8, we study a similar problem of maximizing product adoption, instead of influence or revenue. The overall goal of our problem is same, but our framework is very different. We provide detailed differences in the Chapter.

2.7 Competitive Cascades

Another natural extension of Influence Maximization problem is to introduce a competitive setting. Instead of one product, there can be multiple products in the market all competing with each other.

In [18], Bharathi, Kempe and Salek extended the IC (independent cascade) model to the competitive setting as follows. Each node is either active or inactive. If it is active, then it has adopted exactly one of the multiple products in the market. Once a node adopts a product, it cannot adopt another product. When a node u becomes active at time t w.r.t product r, it makes exactly one attempt to activate each of its inactive neighbor v w.r.t product r. It succeeds with a probability $p_{u,v}$. If it succeeds, the node v activates at time $t + T_{u,v}$, where $T_{u,v}$ is an independent and exponentially distributed continuous random variable. The process continues until there are no more new activations. In the beginning of the game, each of the b players (corresponding to b products) selects a seed set S_i of at most k_i nodes. A node selected by multiple players adopts one of the products randomly.

The authors propose the *best response strategies* for the players and prove that for the last player to commit the strategy, the solution obtained by the simple greedy algorithm is within $(1 - 1/e)$ factor of the optimal solution of the best response. This is due to the monotonicity and submodularity properties of the objective function.

Kostka et al. [85] examined the diffusion of competing rumors in social networks. Their propagation model is again an adaptation of IC (independent cascade) model. Suppose there are b competing rumors. Each node can be either inactive (state 0), or active w.r.t. one of the rumor or in state ∞, where the node rejects all rumors. When a node becomes active w.r.t a rumor r, say at time step t, it attempts to activate each of its inactive neighbors w.r.t the rumor. There are no probabilities on the edges and hence the assumption is that each node trusts the first rumor it encounters and adopts it. If there is more than one rumor, then the node rejects all the rumors and goes to the state ∞. The propagation continues until there are no more activations.

Consider the simplest setting when there are only two competing rumors/players. Let player 1 is the first one to commit the strategy such that he knows the number of seeds picked by player 2 but doesn't know the exact seeds (as player 2 hasn't committed to its strategy yet). The authors prove that not only it is NP-hard to compute the optimal strategy for the player 1, but it is also NP-hard to find an α-approximate solution for any $1 < \alpha \in o(n)$. However, the second player knows the seed set picked by the first player. They prove that although it is NP-hard to find the optimal strategy, the solution provided by the simple greedy algorithm is within $(1 - 1/e)$ factor of the optimal solution.

One might think that the first player is in advantage as it has more freedom to pick the seed set. The authors show through an example that it is not the case.

Carnes et al. [28] consider the competitive setting from the follower's perspective with

only two products. They propose two propagation models, namely *Distance-based model* and *Wave propagation model*. In the *Distance-based model*, the probability of a node v adopting a product is a function of number of nodes in the shortest paths from the two seed sets. In *Wave propagation model*, at any time step, an inactive node randomly picks one of its active neighbors and adopts its product. Both these models reduce to IC (independent cascade) model when there is no competition. They exploit submodularity to prove that the greedy algorithm approximates the second player's optimal strategy within a factor of $(1-1/e)$. Using experiments, they show that the heuristic that chooses high degree nodes is the best strategy for the first player. Moreover, if the first player doesn't pick the initial seed set in this fashion, the second player can in fact outperform it with a relatively low budget.

All the studies mentioned above extend the IC (independent cascade) model. Relatively, less work has been done on using threshold models in competitive settings. Recently, Borodin et al. [22] suggested several extensions to the linear threshold model in the competitive settings. For most of these models (all except one), they show that the greedy approach cannot be used. Furthermore, they show that for a broad family of competitive influence models, it is NP-hard to achieve an approximation that is better than a square root of the optimal solution

2.8 Other Related Work

Leskovec et al. [91] study the propagation problem from a different perspective namely *outbreak detection*: how to select nodes in a network in order to detect the spread of a virus as fast as possible? They present a general methodology for near optimal sensor placement in these and related problems. In the same paper, they also introduced lazy-forward CELF optimization, that we have described above (Algorithm 2).

Cui et al. [40] proposed an interesting problem – given a user, what item she should share with her friends such that her influence among her friends is maximized. They proposed a matrix factorization approach for item-level social influence modeling and devised an efficient projected gradient method to solve it.

Aral et al. [9] investigated how incorporating viral features in the product itself helps in making a product viral. They show that viral product design features can result in identifiable peer influence and social contagion effects. They found that passive-broadcast viral messaging generates a 246% increase in local peer influence and social contagion effects, while adding active-personalized viral messaging only generates an additional 98% increase in contagion.

Budak et al. [26] studied the problem of influence limitation where a bad campaign starts propagating from a certain node in the network and use the notion of limiting campaigns to counteract the effect of misinformation. The goal of their problem statement is to identify a set of individuals that should be convinced to adopt a competing (good) campaign so as to minimize the number of people that adopt the bad campaign.

Chapter 3

Improving the Efficiency

In this chapter, we introduce novel techniques that improve the efficiency of Influence Maximization algorithms, in particular, the greedy algorithm. First, we introduce Celf++, that improves on Celf optimization [91]. We show that Celf++ is 17-61% faster than Celf. Next, we propose Simpath, an efficient and effective algorithm for influence maximization under the linear threshold model. Even though we have provided background and related work in Chapter 2, for the sake of completeness, we provide a brief background and related work here as well. While Celf++ optimization was appeared in WWW 2011 in the companion volume [60], Simpath appeared in ICDM 2011 [61]. Both studies are done in collaboration with Wei Lu and Laks V. S. Lakshmanan. The original paper of Celf++ [60] contained a few bugs in the algorithm. We thank Wei Chen from Microsoft Research Asia for pointing out the bugs to us. Here, we present the fixed version of the paper.

3.1 CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks

Kempe et al. [78] showed the problem of Influence Maximization is NP-hard and a simple greedy algorithm guarantees the best possible approximation factor in PTIME. However, it has two major sources of inefficiency. First, finding the expected spread of a node set is #P-hard. Second, the basic greedy algorithm is quadratic in the number of nodes. The first source is tackled by estimating the spread using Monte Carlo simulation or by using heuristics [35–37, 78, 82, 91]. Leskovec et al. [91] proposed the Celf algorithm for tackling the second. In this work, we propose Celf++ and empirically show that it is 17-61% faster than Celf.

3.1.1 Introduction

In Influence Maximization, we are given a network G with pairwise user influence probabilities (as edge labels) and a number k, and want to find a set S of k users (nodes) such that the expected spread of influence (spread for short) is maximized. In their seminal work [78], Kempe et al. studied this problem, focusing on two fundamental propagation models – *Linear Threshold Model* (LT) and *Independent Cascade Model* (IC). They showed, under both models, the problem is NP-hard and a simple greedy algorithm successively selecting the node with the maximum marginal influence spread approximates the optimum solution within a factor of $(1 - 1/e)$. This is due to the nice properties of *monotonicity* and *submodularity* that the spread function exhibits under these models. In terms of spread, monotonicity says as more neighbors of some arbitrary node u gets active, the probability of u getting active increases.

Submodularity says the marginal gain of a new node shrinks as the set grows. Function f is submodular iff $f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$ whenever $S \subseteq T$.

A major limitation of the simple greedy algorithm is two-fold: (i) The algorithm requires repeated computes of the spread function for various seed sets. The problem of computing the spread under both IC and LT models is #P-hard [35, 37]. As a result, Monte-Carlo simulations are run by Kempe et al. for sufficiently many times to obtain an accurate estimate, resulting in very long computation time. (ii) In each iteration, the simple greedy algorithm searches all the nodes in the graph as a potential candidate for next seed node. As a result, this algorithm entails a quadratic number of steps in terms of the number of nodes.

Considerable work has been done on tackling the first issue, by using efficient heuristics for estimating the spread [35–37, 82] to register huge gains on this front. Relatively little work has been done on improving the quadratic nature of the greedy algorithm. The most notable work is [91], where submodularity is exploited to develop an efficient algorithm called CELF, based on a "lazy-forward" optimization in selecting seeds. The idea is that the marginal gain of a node in the current iteration cannot be better than its marginal gain in the previous iterations. CELF maintains a table $\langle u, \Delta_u(S)\rangle$ sorted on $\Delta_u(S)$ in decreasing order, where S is the current seed set and $\Delta_u(S)$ is the marginal gain of u w.r.t S. $\Delta_u(S)$ is re-evaluated only for the top node at a time and if needed, the table is resorted. If a node remains at the top, it is picked as the next seed. Leskovec et al. [91] empirically shows that CELF dramatically improves the efficiency of the greedy algorithm. See Algorithm 2 for more details.

In this work, we introduce CELF++ that further optimizes CELF by exploiting submodularity. Our experiments show that it improves the efficiency of CELF by 17-61%. Since the optimization introduced in CELF++ is orthogonal to the method used for estimating the spread, our idea can be combined with the heuristic approaches that are based on the greedy algorithm to obtain highly scalable algorithms for INFLUENCE MAXIMIZATION.

3.1.2 CELF++

Algorithm 3 describes the CELF++ algorithm. We use $\sigma(S)$ to denote the spread of seed set S. We maintain a heap Q with nodes corresponding to users in the network G. The node of Q corresponding to user u stores a tuple of the form $\langle u.mg1, u.prev\_best, u.mg2, u.flag\rangle$. Here $u.mg1 = \Delta_u(S)$, the marginal gain of u w.r.t. the current seed set S; $u.prev\_best$ is the node that has the maximum marginal gain among all the users examined in the current iteration, before user u; $u.mg2 = \Delta_u(S \cup \{prev\_best\})$, and $u.flag$ is the iteration number when $u.mg1$ was last updated. The idea is that if the node $u.prev\_best$ is picked as a seed in the current iteration, we don't need to recompute the marginal gain of u w.r.t $(S \cup \{prev\_best\})$ in the *next* iteration.

It is important to note that in addition to computing $\Delta_u(S)$, it is not necessary to compute $\Delta_u(S \cup \{prev\_best\})$ from scratch. More precisely, the algorithm can be implemented in an efficient manner such that both $\Delta_u(S)$ and $\Delta_u(S \cup \{prev\_best\})$ are evaluated simultaneously in a single iteration of Monte Carlo simulation (which typically contains $10,000$ runs). In that sense, the extra overhead is relatively insignificant compared to the huge runtime gains we can achieve, as we will show from our experiments.

In addition to the data structure Q, the algorithm uses the variables S to denote the current

Algorithm 3 Greedy CELF++

Input: G, k
Output: seed set S

1: $S \leftarrow \emptyset$; $Q \leftarrow \emptyset$; $last\_seed \leftarrow$ NULL; $cur\_best \leftarrow$ NULL.
2: **for** each $u \in V$ **do**
3: $u.mg1 \leftarrow \sigma(\{u\})$; $u.prev\_best \leftarrow cur\_best$;
4: $u.mg2 \leftarrow \Delta_u\{cur\_best\}$; $u.flag \leftarrow 0$.
5: $Q \leftarrow Q \cup \{u\}$. Update cur_best based on $u.mg1$.
6: **end for**
7: **while** $|S| < k$ **do**
8: $u \leftarrow$ top (root) element in Q.
9: **if** $u.flag == |S|$ **then**
10: $S \leftarrow S \cup \{u\}$; $Q \leftarrow Q - \{u\}$;
11: $last\_seed \leftarrow u$; $cur\_best \leftarrow$ NULL;
12: **continue**;
13: **else if** $u.prev\_best == last\_seed$ **and** $u.flag == |S| - 1$ **then**
14: $u.mg1 \leftarrow u.mg2$.
15: **else**
16: $u.mg1 \leftarrow \Delta_u(S)$; $u.prev\_best \leftarrow cur\_best$;
17: $u.mg2 \leftarrow \Delta_u(S \cup \{cur\_best\})$.
18: **end if**
19: $u.flag \leftarrow |S|$; Update cur_best; Heapify Q.
20: **end while**

seed set, $last\_seed$ to track the id of last seed user picked by the algorithm, and cur_best to track the user having the maximum marginal gain w.r.t. S over all users examined in the current iteration. The algorithm starts by building the heap Q initially (lines 2-6). Then, it continues to select seeds until the budget k is exhausted. As in CELF, we look at the root element u of Q and if $u.flag$ is equal to the size of the seed set, we pick u as the seed as this indicates that $u.mg1$ is actually $\Delta_u(S)$ (lines 7-12). The optimization of CELF++ comes from lines 13-14 where we update $u.mg1$ without recomputing the marginal gain. Clearly, this can be done since $u.mg2$ has already been computed efficiently w.r.t. the last seed node picked. If none of the above cases applies, we recompute the marginal gain of u (line 15-17).

3.1.3 Experiments

We use two real world data sets consisting of academic collaboration networks: NetHEPT and NetPHY, both extracted from arXiv[21]. NetHEPT is taken from the "High Energy Physics – Theory" section and has 15K nodes and 32K unique edges. NetPHY is taken from the full "Physics" section and has 37K nodes and 174K unique edges. The graphs are undirected, however we make them directed by taking for each edge the arcs in both the directions. We consider the IC model and assign the influence probability to arcs using two different settings, following previous works (e.g., see [35, 36, 78]). In the first setting, for an arc (v, u) we set the influence probability as $p_{v,u} = 1/d_{in}(u)$, where d_{in} is the in-degree of the node u. In the

[21] http://www.arXiv.org

Dataset	Running time (min)			Avg. # node lookups		
	CELF	CELF++	Gain	CELF	CELF++	Gain
Hept WC	245	**178**	27%	18.7	**13.6**	27.2%
Hept IC	5269	**2082**	60.5%	190.5	**113.0**	40.7%
Phy WC	1242	**1028**	17.2%	18.6	**17.9**	3.8%

Table 3.1: Comparison between CELF and CELF++. Number of seeds = 100 in all test cases.

second setting, we assign a uniform probability of 0.1 to all arcs. In all the experiments, we run 10,000 Monte Carlo simulations to estimate the spread.

The results are shown in Table 3.1. We use WC (*weighted cascade*) to refer to the case when the probabilities are non-uniform and IC for the uniform probability 0.1 setting. We only show the results corresponding to NetHEPT WC, NetHEPT IC, and NetPHY WC for brevity. The results for NetPHY IC are similar. In these settings, we found that computing $u.mg2$ for all nodes in the first iteration results in large overhead. So, we apply CELF++ starting from the second iteration. Notice that the optimization behind CELF++ can be applied starting from any iteration. As can be seen, CELF++ is significantly faster than CELF. This is due to the fact that the average number of "spread computations" per iteration is significantly lower. Since we apply the optimization starting from the second iteration, we report the average number of nodes examined starting from the third iteration.

Memory Consumption: Although CELF++ maintains a larger data structure to store the look-ahead marginal gains ($u.mg2$) of each node, the increase of the memory consumption is insignificant. For instance, CELF consumes 21.9 MB for NetHEPT and 39.7 MB for NetPHY, while CELF++ uses 22.4 MB and 41.2 MB respectively.

3.1.4 Conclusions

In this work, we presented CELF++, a highly optimized approach based on the CELF algorithm [91] in order to further improve the naive greedy algorithm for INFLUENCE MAXIMIZATION in social networks [78]. CELF++ exploits the property of submodularity of the spread function for influence propagation models (e.g., Linear Threshold Model and Independent Cascade Model) to avoid unnecessary re-computations of marginal gains incurred by CELF. Our empirical studies on real world social network datasets show that CELF++ works effectively and efficiently, resulting in significant improvements in terms of both running time and the average number of node look-ups.

3.2 Simpath: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model

Kempe et al. [78] showed, among other things, that under the Linear Threshold model, the problem of INFLUENCE MAXIMIZATION is NP-hard, and that a simple greedy algorithm guarantees the best possible approximation factor in PTIME. However, this algorithm suffers from various major performance drawbacks. In this section, we propose SIMPATH, an efficient and

effective algorithm for Influence Maximization under the linear threshold model that addresses these drawbacks by incorporating several clever optimizations. Through a comprehensive performance study on four real data sets, we show that Simpath consistently outperforms the state of the art w.r.t. running time, memory consumption and the quality of the seed set chosen, measured in terms of expected influence spread achieved.

3.2.1 Introduction

In their seminal paper, Kempe et al. [78] formulated Influence Maximization as a discrete optimization problem: Given a directed social graph with users as nodes, edge weights reflecting influence between users, and a number k (called *budget*), find k users, called the *seed set*, such that by activating them, the expected spread of the influence (just *spread* for brevity) according to a propagation model is maximized. The influence diffusion process unfolds in discrete time steps, as captured by the propagation model.

Two classical propagation models discussed in [78] are the *Linear Threshold* (LT) Model and the *Independent Cascade* (IC) Model, both taken from mathematical sociology. In both models, at any time step, a user is either *active* (an adopter of the product) or *inactive*. In the IC model, when an inactive user becomes active at a time step t, it gets exactly one chance to independently activate its currently inactive neighbors at time $t+1$. In the LT model, the sum of incoming edge weights on any node is assumed to be at most 1 and every user chooses an activation threshold uniformly at random from $[0, 1]$. At any time step, if the sum of incoming influence (edge weights) from the active neighbors of an inactive user exceeds its threshold, it becomes active. In both models, influence propagates until no more users can become active.

Influence Maximization under both IC and LT models is NP-hard and the spread function is monotone and submodular [78]. A set function $f : 2^U \rightarrow \mathbb{R}^+$ is monotone if $f(S) \leq f(T)$ whenever $S \subseteq T \subseteq U$. It is submodular if $f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$ for all $S \subseteq T$ and $w \in U \setminus T$. Intuitively, submodularity says the marginal gain $f(S \cup \{w\}) - f(S)$ from adding a new node w shrinks as the seed set grows.

Exploiting these properties, Kempe et al. [78] presented a simple greedy algorithm which repeatedly picks the node with the maximum marginal gain and adds it to the seed set, until the budget k is reached. However, computing exact marginal gain (or exact expected spread) under both the IC and LT models is #P-hard [35, 37]. Hence, it is estimated by running Monte Carlo (MC) simulations. This greedy algorithm, referred to as *simple greedy* henceforth, approximates the problem within a factor of $(1 - 1/e - \epsilon)$ for any $\epsilon > 0$.

Unfortunately, the simple greedy algorithm suffers from the following severe performance drawbacks: (i) The MC simulations that are run sufficiently many times (typically 10,000) to obtain an accurate estimate of spread prove to be very expensive. (ii) The simple greedy algorithm makes $O(nk)$ calls to the spread estimation procedure (MC in this case) where n is the number of nodes in the graph and k is the size of the seed set to be picked.

Considerable work has been done to address the first limitation in the case of the IC model by developing heuristic solutions for seed selection [35, 36, 82]. By contrast, for the LT model, the only similar work to our knowledge is by Chen et al. [37]. They observed that while computing the spread is #P-hard in general graphs, it can be computed in linear time on directed acyclic graphs (DAGs). Moreover, they claim that the majority of the influence flows

only within a small neighborhood and thus they construct one local DAG (LDAG) per node in the graph and assume that the influence flows to the node only through that LDAG. They experimentally show that this heuristic is significantly faster than the greedy algorithm and achieves high quality seed set selection, measured in terms of the spread achieved.

While their approach is interesting, the algorithm has the following limitations: First, it relies heavily on finding a high quality LDAG. However, finding an optimal LDAG is NP-hard [37] and a greedy heuristic is employed to discover a good LDAG. Recall, their algorithm is already a heuristic and using a greedy LDAG in place of the optimal one may introduce an additional level of loss in quality of the seed set chosen in terms of the spread achieved. Second, it assumes that influence flows to a node via paths within only one LDAG and ignores influence flows via other paths. As a result, if other "ignored" LDAGs together have a high influence flow, the quality of the seed set chosen may suffer. Finally, they store all LDAGs in memory, making the approach memory intensive. From here on, we refer to their algorithm as LDAG. In our experiments, we explore each of the limitations of LDAG mentioned above. It is worth noting that LDAG is the state of the art algorithm for INFLUENCE MAXIMIZATION under the LT model.

Toward addressing the second limitation of the simple greedy algorithm, Leskovec et al. [91] proposed the CELF optimization that significantly reduces the number of calls made to the spread estimation procedure (MC simulation). Even though this improves the running time of the simple greedy algorithm by up to 700 times [91], it has still been found to be quite slow and definitely not scalable to very large graphs [37, 57]. In particular, the first iteration is very expensive as it makes n calls to the spread estimation procedure. It should be mentioned that LDAG was shown [37] to be significantly faster than even the simple greedy, optimized using CELF.

In this chapter, we propose the SIMPATH algorithm for INFLUENCE MAXIMIZATION under the LT model. SIMPATH incorporates three key novel ways of optimizing the computation and improving the quality of seed selection, where seed set quality is based on its spread of influence: the larger its spread, the higher its quality.

3.2.2 Contributions and Roadmap

In Section 3.3, we establish a fundamental result on influence propagation in the LT model which says the spread of a set of nodes can be calculated as the sum of spreads of each node in the set on appropriate induced subgraphs. This paves the way for our SIMPATH algorithm. SIMPATH builds on the CELF optimization that iteratively selects seeds in a lazy forward manner. However, instead of using expensive MC simulations to estimate the spread, we show that under the LT model, the spread can be computed by enumerating the simple paths starting from the seed nodes. It is known that the problem of enumerating simple paths is #P-hard [133]. However, the majority of the influence flows within a small neighborhood, since probabilities of paths diminish rapidly as they get longer. Thus, the spread can be computed accurately by enumerating paths within a small neighborhood. We propose a parameter δ to control the size of the neighborhood that represents a direct trade-off between accuracy of spread estimation and running time. Our spread estimation algorithm which we call SIMPATH-SPREAD is given in Section 3.4.

$b_{u,v}$	Influence weight on edge (u, v)
$\Upsilon_{S,v}$	Prob. that v activates if S is the initial seed set
$\sigma(S)$	Expected spread of influence achieved by seed set S
$N^{in}(v)$	Set of in-neighbors of v
$N^{out}(v)$	Set of out-neighbors of v
$\mathsf{P} = \langle v_1, ..., v_m \rangle$	A simple path from v_1 to v_m
$\mathcal{P}(u, v)$	Set of all simple paths from node u to v
δ	Pruning threshold (see Section 3.4)
ℓ	Look-ahead value (see Section 3.5.B)

Figure 3.1: Notations used. Terms Υ, σ and $\mathcal{P}(\cdot)$ can have superscripts implying that these terms are evaluated on the corresponding subgraph. E.g., $\sigma^W(S)$ is the influence spread achieved by the seed set S on the subgraph induced by nodes in W.

We propose two novel optimizations to reduce the number of spread estimation calls made by SIMPATH. The first one, which we call VERTEX COVER OPTIMIZATION, cuts down on the number of calls made in the first iteration (Section 3.5.A), thus addressing a key weakness of the simple greedy algorithm that is *not* addressed by CELF. We show that the spread of a node can be computed directly using the spread of its out-neighbors. Thus, in the first iteration, we first construct a vertex cover of the graph and obtain the spread only for these nodes using the spread estimation procedure. The spread of the rest of the nodes is derived from this. This significantly reduces the running time of the first iteration.

Next, we observe that as the size of the seed set grows in subsequent iterations, the spread estimation process slows down considerably. In Section 3.5.B, we provide another optimization called LOOK AHEAD OPTIMIZATION which addresses this issue and keeps the running time of subsequent iterations small. Specifically, using a parameter ℓ, it picks the top-ℓ most promising seed candidates in the start of an iteration and shares the marginal gain computation of those candidates.

In Section 3.6, we show through extensive experiments on four real datasets that our SIMPATH algorithm is more efficient, consumes less memory and produces seed sets with larger spread of influence than LDAG, the current state of art. Indeed, among all the settings we tried, the seed selection quality of SIMPATH is quite close to the simple greedy algorithm.

Section 3.7 summarizes the section and discusses promising directions for further research.

3.3 Properties of Linear Threshold Model

Consider a digraph $G = (V, E, b)$ with edges $(u, v) \in E$ labeled with influence weights $b_{u,v}$ that represent the influence weight of node u on v. Kempe et al. [78] showed that the LT model is equivalent to the "live-edge" model where a node $v \in V$ picks at most one incoming edge with a probability equal to the edge weight. If an edge is selected, it is considered live. Otherwise, it is considered dead/blocked. This results in a distribution over possible worlds. Then the spread is the expected number of nodes reachable from the initial seed set S over the possible worlds. Let X be any possible world and $\sigma_X(S)$ denote the number of nodes

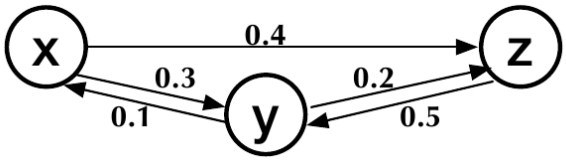


Figure 3.2: Example

reachable from S via live paths in (the deterministic graph) X where a live path is a path made of live edges. Then the spread of S, by definition, is $\sigma(S) = \sum_X Pr[X] \cdot \sigma_X(S)$ where $\sigma_X(S) = \sum_{v \in V} I(S, v, X)$. Here, $I(S, v, X)$ is the indicator function which is 1 if there is a "live" path in X from any node in S to v and 0 otherwise. Hence,

$$\sigma(S) = \sum_{v \in V} \sum_{X} Pr[X] \cdot I(S, v, X) = \sum_{v \in V} \Upsilon_{S,v}, \tag{3.1}$$

where $\Upsilon_{S,v}$ is the probability that v activates if S is the initial seed set. We call it the total influence weight of S on v. We consider only directed paths. Let $\mathsf{P} = \langle v_1, v_2, ...v_m \rangle$ be a path. We write $(v_i, v_j) \in \mathsf{P}$ to indicate that the edge (v_i, v_j) belongs to path P. A simple path is a path in which no nodes are repeated. Henceforth by paths we mean directed simple paths. Clearly, $Pr[\mathsf{P}] = \prod_{(v_i,v_j) \in \mathsf{P}} b_{v_i,v_j}$. By definition of the "live-edge" model, we have[22],

$$\Upsilon_{u,v} = \sum_{\mathsf{P} \in \mathcal{P}(u,v)} Pr[\mathsf{P}] \tag{3.2}$$

where $Pr[\mathsf{P}]$ is the probability of a path P being live and $\mathcal{P}(u, v)$ is the set of all paths from node u to v. As an example, consider the influence graph shown in Fig. 9.2. The influence of node x on z is $\Upsilon_{x,z} = 0.3 \cdot 0.2 + 0.4 = 0.46$.

With superscripts, we denote the corresponding subgraph considered. For example, $\Upsilon^{V-S}_{u,v}$ denotes the total influence weight of u on v in the subgraph induced by $V - S$. For simplicity, we write $V - S$ for $V \setminus S$ and $V - S + u$ to denote $((V \setminus S) \cup \{u\})$. When there is no superscript, the whole graph G is considered. Fig. 3.1 summarizes the notations used in the chapter. SIMPATH builds on the following key result.

Theorem 1. *In the LT model, the spread of a set S is the sum of the spread of each node $u \in S$ on subgraphs induced by $V - S + u$. That is,*

$$\sigma(S) = \sum_{u \in S} \sigma^{V-S+u}(u)$$

We first illustrate the theorem using the example shown in Fig. 9.2. Let $S = \{x, y\}$, then according to the claim, $\sigma(S) = \sigma^{V-y}(x) + \sigma^{V-x}(y) = 1 + 0.4 + 1 + 0.2 = 2.6$ which is the correct spread of the set $\{x, y\}$.

[22]This definition is also used in [37].

Proof of theorem 1: We claim that $\Upsilon_{S,v} = \sum_{u \in S} \Upsilon_{u,v}^{V-S+u}$. The theorem follows from this, upon taking the sum over all nodes $v \in V$ (see Eq. (3.1)). We next prove the claim. Intuitively, it says that the influence weight of a set S on node v is the sum of influence weights of each node $u \in S$ on v on subgraphs induced by $V - S + u$. We prove the claim by induction on path lengths. Specifically, let $\Upsilon_{S,v}(t)$ denote the total influence weight of S on v via all paths of length t. Clearly, $\Upsilon_{S,v} = \sum_t \Upsilon_{S,v}(t)$. To prove the claim above, it suffices to show $\Upsilon_{S,v}(t) = \sum_{u \in S} \Upsilon_{u,v}^{V-S+u}(t), \forall t$.

Base Case: When the path length t is 0, we must have $v \in S$. Then the claim is trivial as $\Upsilon_{S,v}(0) = \Upsilon_{v,v}^{V-S+v}(0) = 1$ and $\Upsilon_{w,v}^{V-S+w}(0) = 0$, $\forall w \neq v$.

Induction Step: Assume the claim for t. Consider paths from S to v of length $t+1$. A path of length $t+1$ from S to v must contain a sub-path of length t from S to some in-neighbor w of v. Clearly, this sub-path cannot pass through v (since the path is simple). Thus, we have:

$$\Upsilon_{S,v}(t+1) = \sum_{w \in N^{in}(v)} \Upsilon_{S,w}^{V-v}(t) \cdot b_{w,v} \tag{3.3}$$

where $N^{in}(v)$ denotes the set of in-neighbors of v. By expanding the term $\Upsilon_{S,w}^{V-v}(t)$ using induction hypothesis in Eq. 3.3, we have:

$$\Upsilon_{S,v}(t+1) = \sum_{w \in N^{in}(v)} \sum_{u \in S} \Upsilon_{u,w}^{V-v-S+u}(t) \cdot b_{w,v}$$

By switching the summations, we have $\Upsilon_{S,v}(t+1) = \sum_{u \in S} \sum_{w \in N^{in}(v)} \Upsilon_{u,w}^{V-v-S+u}(t) \cdot b_{w,v} = \sum_{u \in S} \Upsilon_{u,v}^{V-S+u}(t+1)$. This was to be shown. □

3.4 Spread Estimation using SIMPATH-SPREAD

In this section, we develop SIMPATH-SPREAD, the spread estimation algorithm used by SIMPATH. Eq. 3.1 and 3.2 show that the spread of a node can be computed by summing the weights (i.e., probabilities) of all simple paths originating from it. For instance, consider the example shown in Fig. 9.2. The spread of the node x is $\Upsilon_{x,x} + \Upsilon_{x,y} + \Upsilon_{x,z} = 1 + (0.3 + 0.4 \cdot 0.5) + (0.4 + 0.3 \cdot 0.2) = 1.96$. Moreover, in Theorem 1, we show that the spread of a seed set S can be obtained as the sum of spreads of individual nodes u in S in subgraphs induced by $V - S + u$. Hence, the spread of a seed set can be computed by enumerating paths from nodes in S, although on different subgraphs. Not surprisingly, the problem of enumerating all simple paths is #P-hard [133]. However, we are interested only in path weight (that is, probability of a path being live), which decreases rapidly as the length of the path increases. Thus, the majority of the influence can be captured by exploring the paths within a small neighborhood, where the size of the neighborhood can be controlled by the error we can tolerate. This is the basis of our algorithm. We adapt the classical BACKTRACK algorithm [75, 86] to enumerate all simple paths as follows. We maintain a stack Q containing the nodes in the current path. Initially, Q contains only the current seed node. The algorithm calls the subroutine FORWARD which takes the *last* element in Q and adds the nodes to it in

a depth-first fashion until no more nodes can be added. Whenever a new node is added, the subroutine FORWARD ensures that it does not create a cycle and the path segment has not been explored before. If no more nodes can be added, the algorithm backtracks and the last node is removed from the stack Q. The subroutine FORWARD is called again to get a new path. The process is continued until all the paths are enumerated.

As argued above, the majority of the influence to a node flows in from a small neighborhood and can be captured by enumerating paths within that neighborhood. Thus, we prune a path once its weight reaches below a given *pruning threshold* δ. Intuitively, δ is the error we can tolerate. It represents the trade-off between accuracy of the estimated spread and efficiency of the algorithm (running time). If δ is high, we can tolerate larger error and prune paths early, at the price of accuracy of our spread estimation. On the other hand, if it is low, we will explore more paths, leading to high accuracy but with increased running time. As a special case, if $\delta = 0$, then the spread obtained would be exact. We study the effect of δ on the quality of seed set and running time in Section 3.6.

The spread estimation procedure SIMPATH-SPREAD is described in Algorithms 4, 5, 6. Given a set S and a pruning threshold δ, Algorithm 4 exploits Theorem 1 to compute the spread of S. For each node $u \in S$, it calls BACKTRACK (line 3) which in turn takes a node u, the pruning threshold δ, a given set of nodes $W \subseteq V$ and estimates $\sigma^W(u)$. In addition, the algorithm also takes a set $U \subseteq V$ which is needed to implement optimizations proposed in Section 3.5. To give a preview, in addition to computing $\sigma^W(u)$, the algorithm also computes $\sigma^{W-v}(u)$ for all nodes $v \in U$. We don't need it right now and hence assume U to be an empty set. We will revisit to it in Section 3.5.

Algorithm 4 SIMPATH-SPREAD

Input: S, δ, U
1: $\sigma(S) = 0$.
2: **for** each $u \in S$ **do**
3: $\quad \sigma(S) \leftarrow \sigma(S) +$ BACKTRACK$(u, \delta, V - S + u, U)$.
4: **end for**
5: return $\sigma(S)$.

Algorithm 5 BACKTRACK

Input: u, δ, W, U
1: $Q \leftarrow \{u\}$; $spd \leftarrow 1$; $pp \leftarrow 1$; $D \leftarrow null$.
2: **while** $Q \neq \emptyset$ **do**
3: $\quad [Q, D, spd, pp] \leftarrow$ FORWARD$(Q, D, spd, pp, \delta, W, U)$.
4: $\quad u \leftarrow Q.last()$; $Q \leftarrow Q - u$; *delete* $D[u]$; $v \leftarrow Q.last()$.
5: $\quad pp \leftarrow pp/b_{v,u}$.
6: **end while**
7: return spd.

The subroutine BACKTRACK (Algorithm 5) enumerates all simple paths starting from u. It uses a stack Q to maintain the current nodes on the path, pp to maintain the weight of the current path and spd to track the spread of node u in the subgraph induced by W. $D[x]$

maintains the out-neighbors of x that have been seen so far. Using this, we can keep track of the explored paths from a node efficiently. The variables are initialized in line 1. The subroutine FORWARD is called repeatedly which gives a new pruned path that is undiscovered until now (line 3). Then, the last node is removed from the path and the variable pp is set accordingly. Since u no longer exists in the current path, $D[u]$ is deleted (lines 4-5).

Algorithm 6 FORWARD

Input: $Q, D, spd, pp, \delta, W, U$

```
1: x = Q.last().
2: while ∃y ∈ N^out(x): y ∉ Q, y ∉ D[x], y ∈ W do
3:    if pp · b_{x,y} < δ then
4:       D[x].insert(y).
5:    else
6:       Q.add(y).
7:       pp ← pp · b_{x,y}; spd ← spd + pp.
8:       D[x].insert(y); x ← Q.last().
9:       for each v ∈ U such that v ∉ Q do
10:         spd^{W−v} ← spd^{W−v} + pp.
11:      end for
12:   end if
13: end while
14: return [Q, D, spd, pp].
```

Next, we explain the FORWARD algorithm (Algorithm 6). It takes the *last* element as x on the path (line 1) and extends it as much as possible in a depth-first fashion. A new node y added to the path must be an out-neighbor of x which is not yet explored ($y \notin D[x]$), should not create a cycle ($y \notin Q$) and should be in the graph considered ($y \in W$) (line 2). Lines 3-4 prune the path if its weight becomes less than the pruning threshold δ and the node y is added to the seen neighbors of x. If a node y is added to the path, the weight of the path pp is updated accordingly and the spread is updated (lines 5-7). Finally, the node y is added to the seen neighbors of x (line 8). The process is continued until no new nodes can be added to the path. Lines 9-10 are used only for the optimizations proposed in section 3.5 and can be ignored for now.

3.5 Assembling and Optimizing Simpath

In the previous section, we proposed SIMPATH-SPREAD to compute the spread of a seed set. Now, we can plug it in the greedy algorithm (optimized with CELF) to obtain an algorithm for INFLUENCE MAXIMIZATION. This forms the core of the SIMPATH algorithm. We propose two optimizations to further improve SIMPATH which make it highly efficient.

3.5.1 Vertex Cover Optimization

Even with the CELF optimization, in the first iteration, the spread estimation needs to be done for all nodes in the graph, resulting in $|V|$ calls to the SIMPATH-SPREAD subroutine, where

V is the set of nodes in the input social graph. This makes the selection process of the first seed particularly slow. In this section, we propose the VERTEX COVER OPTIMIZATION that reduces the number of calls to SIMPATH-SPREAD significantly. It leverages the Theorem 2, proved below, which says for any node v, if we have the spread values of all its out-neighbors in the subgraph induced by $V - v$, then we can directly compute the spread of v without making any call to SIMPATH-SPREAD. Using this insight, in the first iteration, we first split the set V of all nodes into two disjoint subsets C and $V - C$ such that C is a vertex cover of the underlying undirected graph G' of the directed social graph G.[23] Then, we *simultaneously* compute for each node $u \in C$, the spread of u in G and in the subgraphs induced by $V - v$, for each in-neighbor v of u that is present in $V - C$. In Algorithm 4, this can be achieved by setting $U = (V - C) \cap N^{in}(u)$ (see Algorithm 7 for details). The BACKTRACK subroutine remains unchanged and in the FORWARD subroutine, in addition to updating *spd* (that is, $\sigma(u)$), lines 9-10 update the variable spd^{V-v} (which represents $\sigma^{V-v}(u)$) for all $v \in (V - C) \cap N^{in}(u)$ properly. Note that in the first iteration, $W = V$. Clearly, v must not be in the current path ($v \notin Q$). Once this is done, we can use Theorem 2 to compute the spread of every node in $V - C$ directly.

Theorem 2. *In the LT model, the spread of a node linearly depends on the spread of its out-neighbors as follows.*

$$\sigma(v) = 1 + \sum_{u \in N^{out}(v)} b_{v,u} \cdot \sigma^{V-v}(u)$$

As an example, consider the graph shown in Fig. 9.2. Spread of node x can be computed as $\sigma(x) = 1 + b_{x,y} \cdot \sigma^{V-x}(y) + b_{x,z} \cdot \sigma^{V-x}(z) = 1 + 0.3 \cdot (1 + 0.2) + 0.4 \cdot (1 + 0.5) = 1.96$.

Proof of theorem 2: Consider $\Upsilon_{v,y}$, the influence of v on an arbitrary node y. Recall that it is the sum of probabilities of simple paths from v to y. If $y \equiv v$, $\Upsilon_{v,y} = 1$. For $y \in V - v$, any path from v to y must pass through some out-neighbor u of v. Let $\mathsf{P} = \langle v, u, ..., y \rangle$ be such a path. Clearly, $Pr[\mathsf{P}] = b_{v,u} \cdot Pr[\mathsf{P}']$, where $\mathsf{P}' = \langle u, ..., y \rangle$. From this, we have

$$\Upsilon_{v,y} = \sum_{u \in N^{out}(v)} \sum_{\mathsf{P}' \in \mathcal{P}(u,y)} b_{v,u} \cdot Pr[\mathsf{P}']$$

where $N^{out}(v)$ is the set of out-neighbors of v. Since path P does not have any cycles, P' must not contain v and thus, we can rewrite the above equation as

$$\begin{aligned} \Upsilon_{v,y} &= \sum_{u \in N^{out}(v)} \sum_{\mathsf{P}' \in \mathcal{P}^{V-v}(u,y)} b_{v,u} \cdot Pr[\mathsf{P}'] \\ &= \sum_{u \in N^{out}(v)} b_{v,u} \cdot \Upsilon^{V-v}_{u,y} \end{aligned}$$

where $\mathcal{P}^{V-v}(u, y)$ is the set of (simple) paths from u to y in the subgraph induced by $V - v$. Taking the sum over all $y \in V$, we get the theorem. □

[23] The underlying undirected graph G' of a digraph G is obtained by ignoring the directions of edges and deleting any duplicate edges. A vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex in the set.

Finding Vertex Cover. Minimizing the number of calls to SIMPATH-SPREAD in the first iteration making use of Theorem 2, requires finding a minimum vertex cover (that is, a vertex cover of minimum size). However, this is an **NP**-complete problem. Fortunately, this problem is approximable and several PTIME approximation algorithms are known (e.g., see [48] and [77]). However, it is important to keep in mind that the overhead in finding an approximate minimum vertex cover should not overshadow the benefits coming from it, by means of saved spread estimation calls.

In our experiments, we observed that when using the approximation algorithms, the amount of savings achieved is dominated by the running time of the algorithms, so we settle for a heuristic solution. We use a maximum degree heuristic that repeatedly picks a node of maximum degree and adds it into the vertex cover C whenever at least one of its incident edges is not yet covered. It does this until all edges in the (underlying undirected) graph G' are covered, i.e., incident to at least one vertex in C. It is worth pointing out that the approximation quality of the vertex cover does not affect the output quality (i.e., seed selection) of our algorithm. We employ it only to improve the efficiency of our algorithm. Thus, we recommend a fast heuristic such as the maximum degree heuristic for VERTEX COVER OPTIMIZATION.

3.5.2 Look Ahead Optimization

As the seed set grows, the time spent on the spread estimation process increases. An important reason is that the number of calls to the BACKTRACK subroutine in an iteration increases hurting the performance of SIMPATH. For instance, if S_i is the seed set after the i-th iteration ($|S_i| = i$), then for each seed candidate x, we need to compute $\sigma(S_i + x)$ to obtain its marginal gain $\sigma(S_i + x) - \sigma(S_i)$, where $\sigma(S_i)$ is known from the previous iteration.

It takes $i + 1$ calls to BACKTRACK to compute $\sigma(S_i + x)$, resulting in $j \cdot (i + 1)$ calls in total, where j is the index of the seed node to be picked in the CELF queue. In other words, j is the number of nodes the algorithm ends up examining before picking the seed in the iteration. In each BACKTRACK call, we compute the spread of an individual node $v \in S_i + x$ on the subgraph induced by $V - S + v$. Except for the node x, the work for all other nodes in S_i is largely repeated, although on slightly different graphs. In this section, we propose LOOK AHEAD OPTIMIZATION, which intelligently avoids this repetition and as a result, significantly reduces the number of BACKTRACK calls. We manipulate Theorem 1 as follows.

$$\sigma(S_i + x) = \sum_{u \in S_i + x} \sigma^{V - S_i - x + u}(u) \tag{3.4}$$

$$= \sigma^{V - S_i}(x) + \sum_{u \in S_i} \sigma^{V - S_i - x + u}(u) \tag{3.5}$$

$$= \sigma^{V - S_i}(x) + \sigma^{V - x}(S_i) \tag{3.6}$$

Thus, if we have a set U of the most promising seed candidates to use from an iteration, say $i + 1$, we can compute $\sigma^{V-x}(S_i)$ for all $x \in U$ in the start of the iteration $i + 1$. Then, for each x, we call BACKTRACK to compute $\sigma^{V-S_i}(x)$.

Let ℓ be the *look-ahead value*, that is, the number of most promising seed candidates in an iteration. At the beginning of the $i+1$-th iteration, we select the top-ℓ nodes as one batch from the CELF queue as U, and estimate $\sigma(S_i + x)$ based on Eq. (7). If the algorithm fails to find a new seed from the current batch, we take the next top-ℓ nodes from the CELF queue as U. The process is repeated until a seed is selected (see Algorithm 7). Consequently, the optimization reduces the number of BACKTRACK calls to $i \cdot \lceil j/\ell \rceil + j$ where $\lceil j/l \rceil$ is the number of batches we end up processing. The look-ahead value ℓ represents the trade-off between the number of batches we process and the overhead of computing $\sigma^{V-x}(S_i)$ for all $x \in U$. A large value of ℓ would ensure that the seed node is picked in the first batch itself ($\lceil j/\ell \rceil = 1$). However, that would be inefficient as the overhead of computing $\sigma^{V-x}(S_i)$ for all $x \in U$ for a large U is high. On the other hand, a small value of ℓ would result in too many batches that we end up processing. As a special case, $\ell = 1$ is equivalent to the optimization not being applied. We study the effect of ℓ on SIMPATH's efficiency in Section 3.6.

3.5.3 SIMPATH: Putting the pieces together

Algorithm 7 shows the complete SIMPATH algorithm. Lines 1-8 implement the VERTEX COVER OPTIMIZATION. First, the algorithm finds a vertex cover C (line 1), then for every node $u \in C$, its spread is computed on required subgraphs needed for the optimization (lines 2-4). This is done in a single call to SIMPATH-SPREAD. Next, for the nodes that are not in the vertex cover, the spread is computed using Theorem 2 (lines 6-7). The CELF queue (sorted in the decreasing order of marginal gains) is built accordingly (lines 5 and 8).

Lines 9-20 select the seed set in a lazy forward (CELF) fashion using LOOK AHEAD OPTIMIZATION. The spread of the seed set S is maintained using the variable *spd*. At a time, we take a batch of top-ℓ nodes, call it U, from the CELF queue (line 11). In a single call to SIMPATH-SPREAD, the spread of S is computed on required subgraphs needed for the optimization (lines 12). For a node $x \in U$, if it is processed before in the same iteration, then it is added in the seed set as it implies that x has the maximum marginal gain w.r.t. S (lines 13-16). Recall that the CELF queue is maintained in decreasing order of the marginal gains and thus, no other node can have a larger marginal gain. If x is not seen before, we need to recompute its marginal gain, which is done in lines 17-19. The subroutine BACKTRACK is called to compute $\sigma^{V-S}(x)$ and Eq. 3.6 is applied to get the spread of the set $S + x$. The CELF queue is updated accordingly (line 20).

3.6 Experiments

We conduct extensive experiments on four real-world datasets ranging from small to large scale, to evaluate the performance of SIMPATH and compare it with well-known INFLUENCE MAXIMIZATION algorithms on various aspects such as efficiency, memory consumption and quality of the seed set.

The code is written in C++ using the Standard Template Library (STL), and all the experiments are run on a Linux (OpenSuSE 11.3) machine with a 2.93GHz Intel Xeon CPU and 64GB memory. The code is available for download at `http://people.cs.ubc.ca/~welu/downloads.html`. It includes the implementation of all the algorithms listed in this section.

Algorithm 7 SIMPATH

Input: $G = (V, E, b), \delta, \ell$

Find the vertex cover of input graph G. Call it C.
for each $u \in C$ **do**
  $U \leftarrow (V - C) \cap N^{in}(u)$.
  Compute $\sigma(u)$ and $\sigma^{V-v}(u)$, $\forall v \in U$ in a single call to Algorithm 4: SIMPATH-SPREAD(u, δ, U).
  Add u to CELF queue.
end for
for each $v \in V - C$ **do**
  Compute $\sigma(v)$ using Theorem 2.
  Add v to CELF queue.
end for
$S \leftarrow \emptyset$. $spd \leftarrow 0$.
while $|S| < k$ **do**
  $U \leftarrow$ top-ℓ nodes in CELF queue.
  Compute $\sigma^{V-x}(S)$, $\forall x \in U$, in a single call to Algorithm 4: SIMPATH-SPREAD(S, δ, U).
  for each $x \in U$ **do**
    if x is previously examined in the current iteration **then**
      $S \leftarrow S + x$; Update spd.
      Remove x from CELF queue. Break out of the loop.
    end if
    Call BACKTRACK$(x, \delta, V - S, \emptyset)$ to compute $\sigma^{V-S}(x)$.
    Compute $\sigma(S + x)$ using Eq. 3.6.
    Compute marginal gain of u as $\sigma(S + x) - spd$.
    Re-insert u in CELF queue such that its order is maintained.
  end for
end while
return S.

3.6.1 Datasets

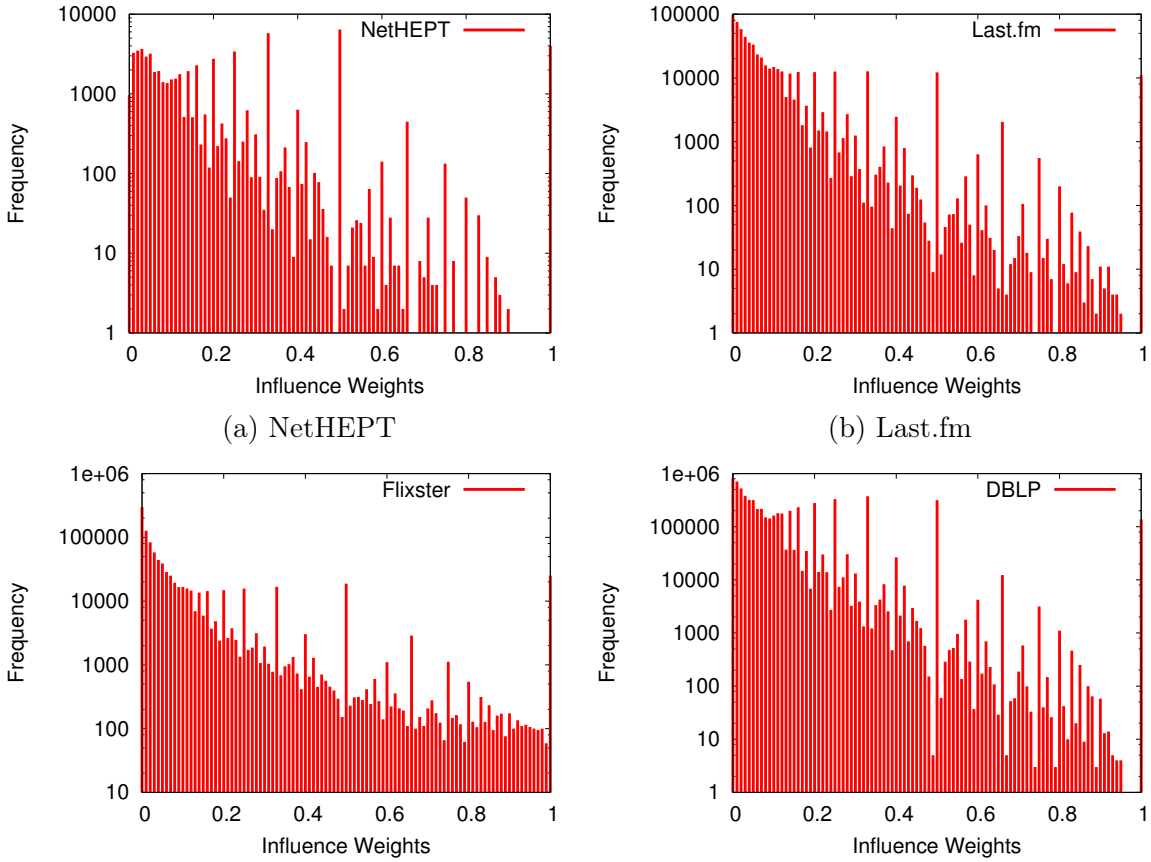
We use four real-world graph data sets whose statistics are summarized in Table 3.2. To obtain influence weights on edges, we adopt the methods in [56] and [78] and learn them. More precisely, we assign the weight on an edge (u, v) as $b_{u,v} = A(u, v)/N(v)$ where $A(u, v)$ is the number of actions both u and v perform, and $N(v)$ is a normalization factor to ensure that the sum of incoming weights to v is 1, i.e., $N(v) = \sum_{u \in N^{in}(v)} A(u, v)$. The distribution of influence weights for all datasets is in Fig. 3.3. The details are as follows:

NetHEPT. Frequently used in previous works [35–37, 78] in this area, NetHEPT is a collaboration network taken from the "High Energy Physics (Theory)" section of arXiv[24] with nodes representing authors and edges representing co-authorship. Here, an action is a user publishing a paper, and $A(u, v)$ is the number of papers co-authored by u and v. The numbers of nodes and directed edges in the graph are 15K and 62K respectively[25].

Last.fm. Taken from a popular music and online radio website with social networking fea-

[24] `http://arxiv.org/`

[25] The dataset is publicly available at `http://research.microsoft.com/en-us/people/weic/graphdata.zip`



(a) NetHEPT (b) Last.fm

(c) Flixster (d) DBLP

Figure 3.3: The distributions of influence weights for each dataset.

tures[26], this network data contains 100K users, 3.15M edges, 66K groups, and 1M subscriptions [119]. We consider "joining a group" as an action, and thus, $A(u, v)$ is the number of groups both u and v have joined. After excluding users who did not join any group, we are left with 60K users with 584K directed edges.

Flixster. Taken from a social movie site[27] allowing users to share movie ratings, the raw data contains 1M users with 28M edges. There are 8.2M ratings distributed among 49K distinct movies [74][28]. Here, an action is a user rating a movie and thus, $A(u, v)$ is the number of movies rated by both u and v. Users who do not rate any movies are excluded. The resulting graph contains 99K users and 978K directed edges.

DBLP. A collaboration network from the DBLP Computer Science Bibliography, much larger compared to NetHEPT. We obtain all the latest bibliographic records (snapshot on June 7, 2011) available from the website. Similar to NetHEPT, $A(u, v)$ is the number of manuscripts

[26] `http://www.last.fm/`

[27] `http://www.flixster.com/`

[28] The dataset is available at http://www.cs.sfu.ca/~sja25/personal/datasets

Table 3.2: Statistics of datasets.

Dataset	NetHEPT	Last.fm	Flixster	DBLP
#Nodes	15K	61K	99K	914K
#Directed Edges	62K	584K	978K	6.6M
Avg. Degree	4.1	9.6	9.9	7.2
Maximum Out-degree	64	1073	428	950
#Connected Components	1781	370	1217	41.5K
Avg. Component Size	8.55	164	81.4	22
Largest Component Size	6794	59.7K	96.5K	789K

both u and v have co-authored. The graph consists of 914K nodes and 6.6M edges.

3.6.2 Algorithms Compared

Besides LDAG, which is the state of art, we include several other algorithms and some generic model independent heuristics (listed below) in our empirical evaluation.

HIGH-DEGREE. A heuristic based on the notion of "degree centrality", considering high-degree nodes as influential [78]. The seeds are the nodes with the k highest out-degrees.

PAGERANK. A link analysis algorithm to rank the importance of pages in a Web graph [23]. We implement the power method with a damping factor of 0.85 and pick the k highest-ranked nodes as seeds. We stop when the score vectors from two consecutive iterations differ by at most 10^{-6} as per L_1-norm.

MC-CELF. The greedy algorithm with CELF optimization [91]. Following the literature, we run $10,000$ Monte Carlo (MC) simulations to estimate the spread of a seed set.

LDAG. The algorithm proposed in [37]. As recommended by the authors, we use the influence parameter $\theta = 1/320$ to control the size of the local DAG constructed for each node.

SIMPATH. Our proposed algorithm described in Sections 3.4 and 3.5 (Algorithm 7). Unless otherwise noted, the pruning threshold δ used in SIMPATH is set to 10^{-3} and the look-ahead value ℓ is set to 4. These values were chosen based on empirically observing the performance.

SPS-CELF++. In Section 3.1, we proposed CELF++, an improvement to CELF. Thus, a natural question is whether we can obtain an algorithm better than SIMPATH by using CELF++ in place of CELF. We investigate this question next. More concretely, define SPS-CELF++ to be the greedy algorithm with CELF++ (instead of CELF) used to select seeds and SIMPATH-SPREAD (instead of MC) used for spread estimation. SPS-CELF++ also leverages VERTEX COVER OPTIMIZATION for the first iteration (Section 3.5.A). As recommended by [60], we apply CELF++ starting from the second iteration. For fairness of comparison, we set δ to the same value 10^{-3} that we used in SIMPATH.

As described in Section 3.1, CELF++ works well when it is possible to compute marginal gain of a node u w.r.t. the current seed set S_i and $S_i + x$ simultaneously where x is the node having the maximum marginal gain among all nodes examined in the current iteration. Next, we show how to do this within SIMPATH architecture. From Theorem 1, using an algebraic

Table 3.3: SIMPATH's improvement over LDAG

Dataset	Improvement in		
	Spread	Running Time	Memory
NetHEPT	8.7%	21.7%	62.9%
Last.fm	1.7%	42.9%	86.5%
Flixster	8.9%	33.6%	87.5%
DBLP	2.3%	67.2%	87.1%

manipulation similar to that in Eq. (3.6), we can obtain

$$\sigma(S_i + x + u) = \sigma^{V-x}(S_i + u) + \sigma^{V-S_i-u}(x)$$

Thus, in the i-th iteration, while computing $\sigma(S_i + u)$, we also compute $\sigma^{V-x}(S_i + u)$ by setting $U = \{x\}$ in Algorithm 4. Finally, if x is selected as seed, we just compute $\sigma^{V-S_i-u}(x)$ to obtain $\sigma(S_i + x + u)$.

However, CELF++ cannot be applied in conjunction with the LOOK AHEAD OPTIMIZATION as CELF++ requires to compute $\sigma^{V-x}(S_i + u)$, where x is the previous best node (having maximum marginal gain), but when we compute the spread of S_i on different subgraphs at the beginning of an iteration, this x is unknown. As a result, in SPS-CELF++, we can incorporate VERTEX COVER OPTIMIZATION but not LOOK AHEAD OPTIMIZATION. We include SPS-CELF++ in our evaluation. Since it is more efficient than simple greedy using CELF++, we do not evaluate the latter separately.

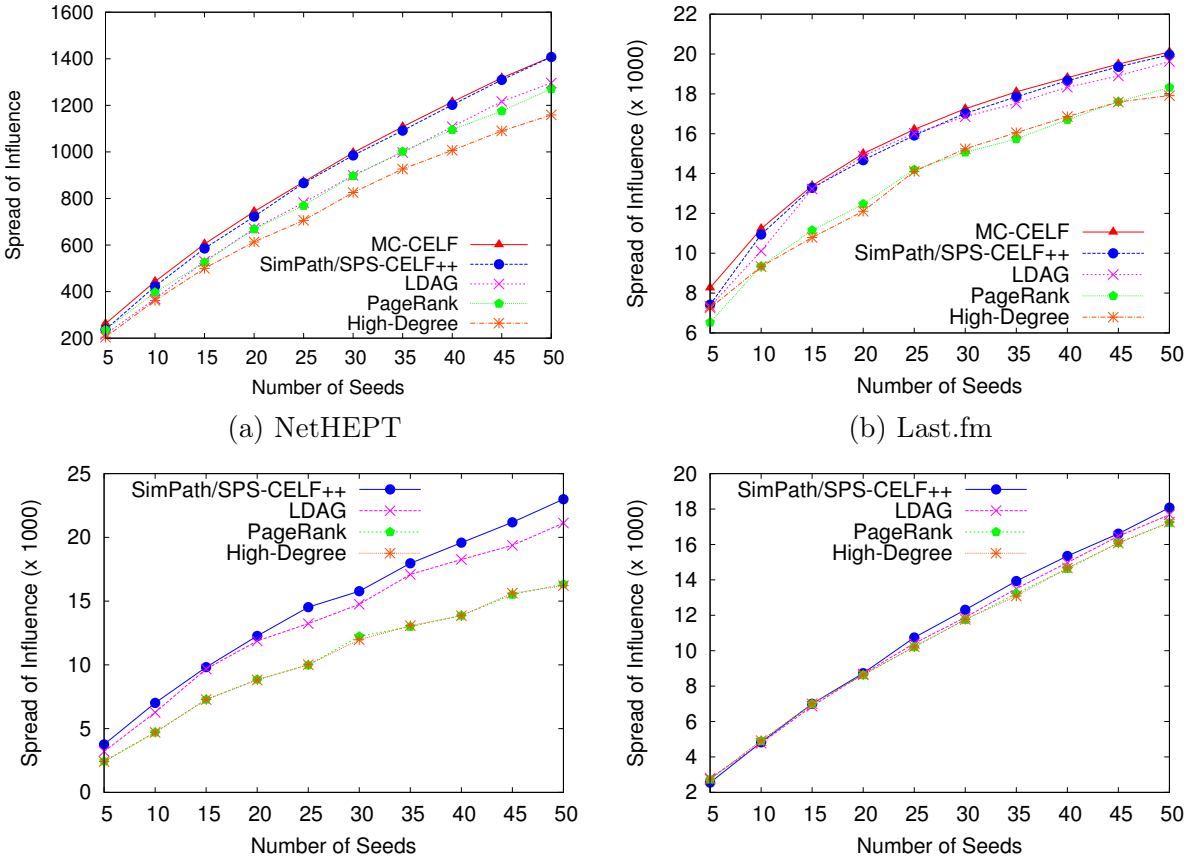
3.6.3 Experimental Results

We compare the performance of the various algorithms on the following metrics: quality of seed sets, running time, memory usage, and scalability. We also study the effectiveness of our two optimizations: the VERTEX COVER OPTIMIZATION and LOOK AHEAD OPTIMIZATION.

Table 3.3 gives an overall summary of the relative performance of LDAG and SIMPATH observed in our experiments. It shows the percentage improvement registered by SIMPATH over LDAG on quality of seed set (measured in spread), running time, and memory consumption. On all counts, it can be seen that SIMPATH outperforms LDAG, the state of the art. We will drill down into the details in the next subsections.

Due to MC-CELF's lack of efficiency and scalability, its results are only reported for NetHEPT and Last.fm, the two datasets on which it can finish in a reasonable amount of time.

On Quality of Seed Sets. The quality of the seed sets obtained from different algorithms is evaluated based on the expected spread of influence. Higher the spread, better the quality. For fair comparisons, we run MC simulations $10,000$ times to obtain the "ground truth" spread of the seeds sets obtained by all algorithms. Fig. 3.4 shows the spread achieved against the size of the seed set. We show the spread of the seed sets chosen by SIMPATH and SPS-CELF++ as one plot because they both use Algorithm 4 (SIMPATH-SPREAD) to estimate the spread and hence, the seed sets produced by both the algorithms are exactly the same.



(a) NetHEPT (b) Last.fm

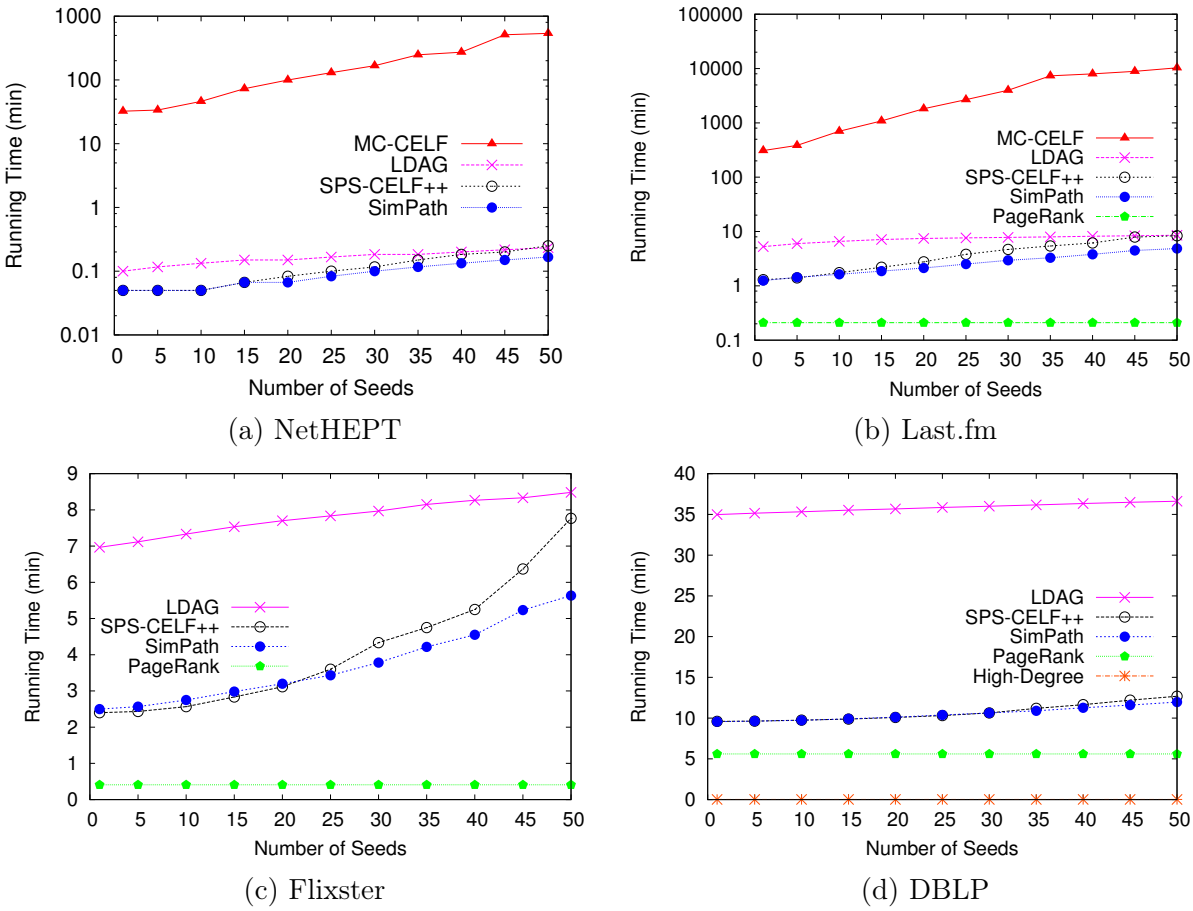
(c) Flixster (d) DBLP

Figure 3.4: Influence spread achieved by various algorithms.

The seed sets output by SIMPATH are quite competitive in quality with those of MC-CELF. For instance, both have the spread 1408 on NetHEPT, while on Last.fm, SIMPATH is only 0.7% lower than MC-CELF in spread achieved. Note that MC-CELF is too slow to complete on Flixster and DBLP.

Except for MC-CELF, on all datasets, SIMPATH is able to produce seed sets of the highest quality. The biggest differences between SIMPATH and LDAG are seen on Flixster and NetHEPT, where the seed set of SIMPATH has 8.9% and 8.7% larger spread than that of LDAG, respectively. PAGERANK and HIGH-DEGREE have similar performances, both being worse than SIMPATH, e.g., PAGERANK is 9.7%, 8.2%, 27.5%, and 4.7% lower than SIMPATH in spread achieved on NetHEPT, Last.fm, Flixster, DBLP, respectively.

On Efficiency and Memory Consumption. We evaluate the efficiency and scalability on two aspects: running time and memory usage. Fig. 3.5 reports the time taken by various algorithms against the size of the seed set. Note that the plots for NetHEPT and Last.fm have a *logarithmic scale* on the y-axis. MC-CELF takes 9 hours to finish on NetHEPT and 7 days on Last.fm while it fails to complete in a reasonable amount of time on Flixster and DBLP.



(a) NetHEPT (b) Last.fm

(c) Flixster (d) DBLP

Figure 3.5: Efficiency: Comparisons of running time. Running times below 0.01 minutes are not shown.

High-Degree and PageRank finish almost instantly in most cases[29]. Except for them, Simpath is the fastest among other algorithms on all datasets. It is not only highly efficient on moderate datasets such as NetHEPT (10 seconds), but also scalable to large data, finishing in 5.6 min. on Flixster and in 12.0 min. on DBLP to select 50 seeds. Ldag is approximately twice and 3 times slower on Flixster and DBLP, respectively.

On the other hand, Sps-Celf++ is less scalable than Simpath on large datasets. This is because that Sps-Celf++ is not compatible with the Look Ahead Optimization, so it makes a much larger number of calls to Backtrack (Algorithm 5) to compute the marginal gains of seed candidates, especially when $|S|$ becomes large. Recall that the Vertex Cover Optimization is indeed applicable to Sps-Celf++, indicating that Look Ahead Optimization makes a substantial difference in running time between Simpath and Sps-Celf++ (more on this later).

Next, we compare the memory consumption of Mc-Celf, Simpath, Sps-Celf++, and

[29]Running time below 0.01 min. are not shown on the plots.

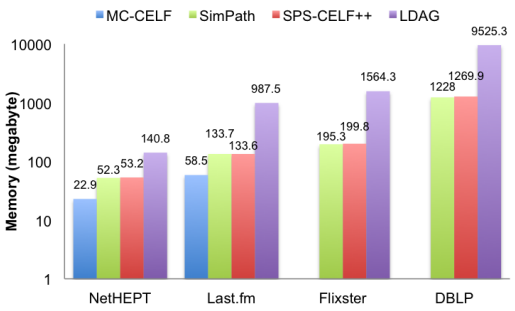


Figure 3.6: Comparison of memory usages by Mc-Celf, Simpath, and Ldag (logarithmic scale).

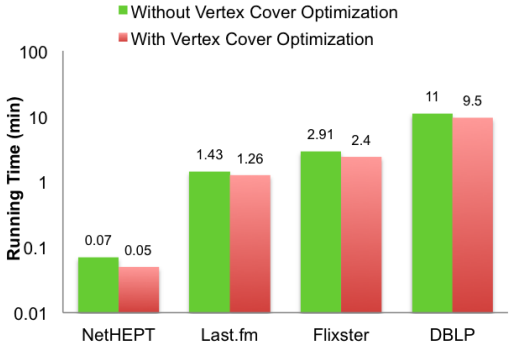


Figure 3.7: Effects of Vertex Cover Optimization on the running time of Simpath's 1st iteration (logarithmic scale).

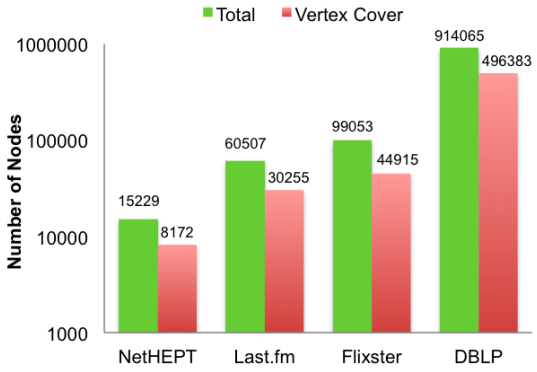
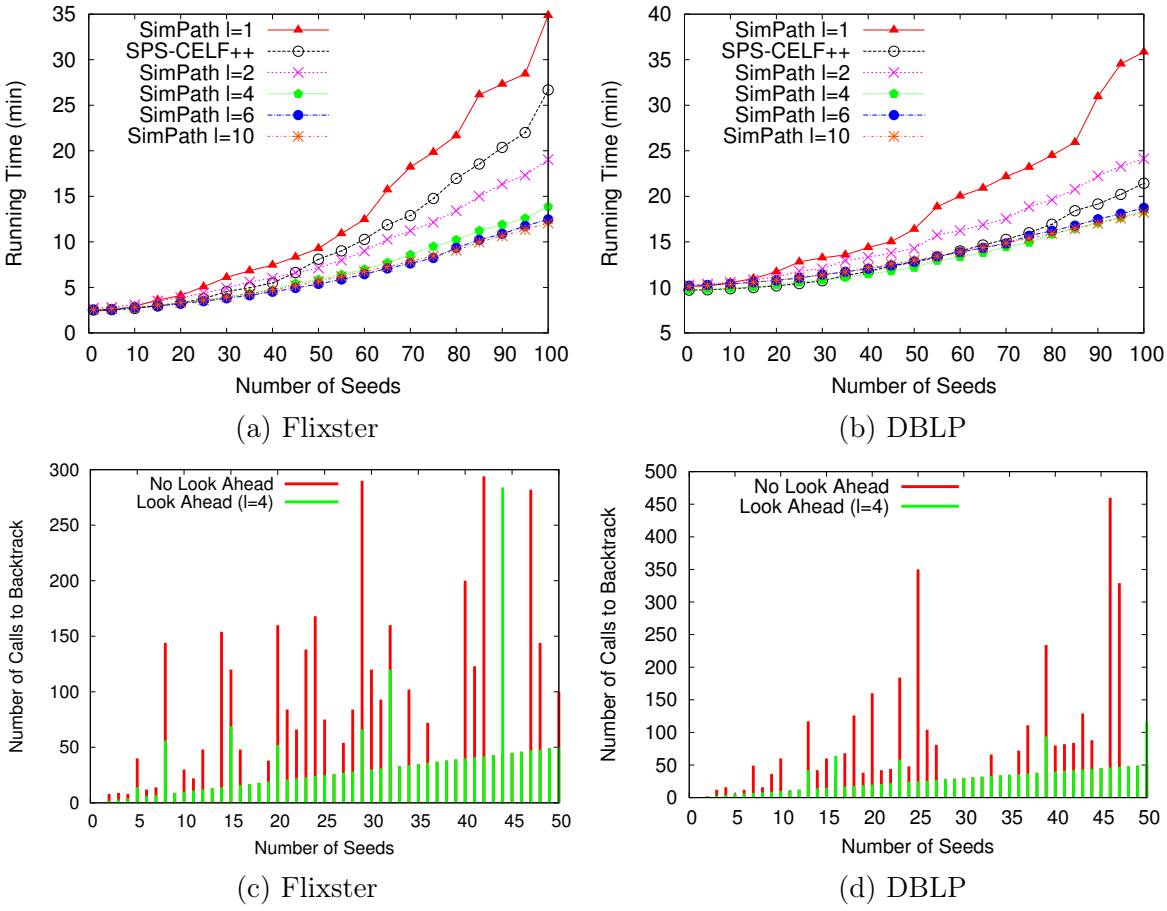


Figure 3.8: Size of Vertex Covers for four datasets (logarithmic scale).

Ldag in Fig. 3.6 (*logarithmic scale*). Ldag consumes the most memory among the three algorithms, as it maintains a local DAG for every single node in the graph. Relatively, the memory usage by Simpath is much less. For instance, on the largest dataset DBLP, Ldag consumes 9.3GB while Simpath uses only 1.2GB. Indeed, Table 3.3 shows that Simpath can save up to 87% of the memory footprint that is used by Ldag.

On Vertex Cover Optimization. Recall that Vertex Cover Optimization aims to reduce the number of spread estimation calls in the first iteration, thus addressing a limitation of Celf. We show its effectiveness in Fig. 3.7 and 3.8 (both have *logarithmic scale* on Y-axis). In Fig. 3.7, we compare the running time of the first iteration of Simpath with and without the Vertex Cover Optimization. With the optimization turned on, the first iteration is 28.6%, 11.9%, 17.5%, and 13.6% faster on NetHEPT, Last.fm, Flixster, and DBLP, respectively, than without the optimization. This is because with this optimization, the number of calls to Simpath-Spread is only for a fraction of nodes in the graph (i.e., the vertex cover).

Next, in Fig. 3.8, we report the size of the vertex cover found by the maximum degree



(a) Flixster (b) DBLP

(c) Flixster (d) DBLP

Figure 3.9: Look Ahead Optimization on Flixster and DBLP: (a) and (b) show the effects of different look-ahead values ℓ on running time; (c) and (d) show the number of Backtrack calls reduced by Look Ahead Optimization.

heuristic for each dataset. On an average, Vertex Cover Optimization reduces the number of calls to Simpath-Spread by approximately 50%.

On Look Ahead Optimization. We show the effectiveness of Look Ahead Optimization on Flixster and DBLP (results are similar on other two datasets). First, we choose five values to compare running time: 1 (equivalent to no look ahead), 2, 4, 6, and 10 (Fig. 3.9 (a)-(b)). We also include Sps-Celf++ (which is not compatible with this optimization) as baseline. In both cases we select 100 seeds. On both Flixster and DBLP, Simpath with small ℓ (1 or 2) performs poorly and is slower than Sps-Celf++. For DBLP, $\ell = 4$ is the best choice, while $\ell = 6$ is the best for Flixster. When ℓ increases to 10, the running time goes up slightly on both datasets, suggesting that to take a batch of 10 candidates introduces more overhead than the benefit brought by the optimization (see Section 3.5.B).

In Fig. 3.9 (c)-(d), we show the number of Backtrack calls reduced by using the Look Ahead Optimization. On both datasets, without look-ahead, the number of

Table 3.4: Pruning threshold η as a trade-off between quality of seed sets and efficiency. $|S| = 50$. Running time in minutes.

δ	NetHEPT		DBLP	
	$\sigma(S)$	Time	$\sigma(S)$	Time
10^{-1}	1160	0.02	16592	1.02
10^{-2}	1362	0.03	17868	2.02
10^{-3}	1408	0.18	18076	12.0
10^{-4}	1414	1.3	18151	104.3
10^{-5}	1416	9.9	18350	927.2

Backtrack calls grows drastically and fluctuates when $|S|$ increases, while with look-ahead, it grows gently and steadily, being mostly around the value of $|S|$.

On Pruning Threshold of Simpath. To study how effectively the pruning threshold δ represents a trade-off between efficiency and quality of seed set, we run Simpath with different values of δ on one moderate dataset (NetHEPT) and one large (DBLP). The look-ahead value ℓ is set to 4.

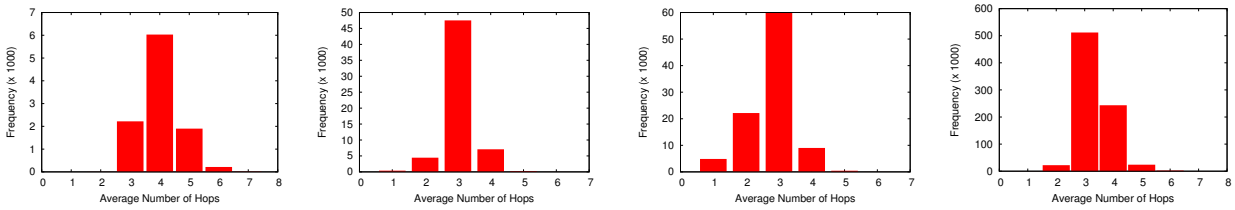
The results in Table 3.4 clearly show that on both datasets, as δ decreases, the running time of Simpath rises, while the estimated spread of influence $\sigma(S)$ is improving. For instance, when we decrease δ from 10^{-3} to 10^{-4}, the running time increases 7.7 folds while the spread is increased only by 0.41%. Other datasets follow similar behavior. It suggests that 0.001 is indeed a good choice for δ. It is worth mentioning that when δ is relatively small (e.g., 10^{-4} and 10^{-5}), Simpath can even produce seed sets with larger spread than those of Mc-Celf with $10,000$ simulations (which give a spread of 1408 on NetHEPT).

On Number of Hops. The next analysis we perform is how the choice of the pruning threshold is related to the average number of hops explored. Fig. 3.10 shows the distribution. On all the four datasets, the distribution follows a bell curve with the mean of either 3 or 4. For instance, on Flixster, for 60K users, the influence decays below 10^{-3} in 3 hops on an average. We also found that for any user, on all four datasets, the influence decays below 10^{-3} in a maximum of 8 hops. These statistics support our conjecture that the majority of the influence flows in a small neighborhood, and thus, even though enumerating simple paths in #P-hard in general, computation of influence spread by enumerating simple paths can be done efficiently and accurately by focusing on a small neighborhood.

To summarize, our experiments demonstrate that Simpath consistently outperforms other Influence Maximization algorithms and heuristics. It is able to produce seed sets with quality comparable to those produced by Mc-Celf, but is far more efficient and scalable. Also, Simpath is shown to have higher seed set quality, lower memory footprint, and better scalability than other well-established heuristics and the state of the art Ldag.

3.7 Summary and Future Work

Designing a scalable algorithm delivering high quality seeds for Influence Maximization problem under the LT model is the main goal of this chapter/section. The simple greedy



(a) NetHEPT (b) Last.fm (c) Flixster (d) DBLP

Figure 3.10: Frequency Distribution of Average Number of Hops.

algorithm is known to produce the best possible seed sets (in terms of influence spread) in PTIME but suffers from severe performance issues. The CELF optimization [91] helps reduce the number of spread estimation calls significantly, except in the first iteration. On the other hand, the LDAG heuristic proposed by Chen et al. [37], the current state of art, is shown to be significantly faster than the greedy algorithm and is often found to generate a seed set of high quality. We propose an alternative algorithm SIMPATH that computes the spread by exploring simple paths in the neighborhood. Using a parameter δ, we can strike a balance between running time and desired quality (of the seed set). SIMPATH leverages two optimizations. The VERTEX COVER OPTIMIZATION cuts down the spread estimation calls in the first iteration, thus addressing a key limitation of CELF, while the LOOK AHEAD OPTIMIZATION improves the efficiency in subsequent iterations. Through extensive experimentation on four real data sets, we show that SIMPATH outperforms LDAG, in terms of running time, memory consumption and the quality of the seed sets. Currently, we are investigating extension of our techniques for other propagation models.

Chapter 4

Learning Influence Probabilities in Social Networks

Recently, there has been tremendous interest in the phenomenon of influence propagation in social networks. The studies in this area assume they have as input to their problems a social graph with edges labeled with probabilities of influence between users. However, the question of where these probabilities come from or how they can be computed from real social network data has been largely ignored until now. Thus it is interesting to ask whether from a social graph and a log of actions by its users, one can build models of influence. This is the main problem attacked in this chapter. In addition to proposing models and algorithms for learning the model parameters and for testing the learned models to make predictions, we also develop techniques for predicting the time by which a user may be expected to perform an action. We validate our ideas and techniques using the Flickr data set consisting of a social graph with 1.3M nodes, 40M edges, and an action log consisting of 35M tuples referring to 300K distinct actions. Beyond showing that there is genuine influence happening in a real social network, we show that our techniques have excellent prediction performance.

This chapter is based on our WSDM 2010 paper [56]. This study has been done in collaboration with Francesco Bonchi and Laks V. S. Lakshmanan.

4.1 Introduction

In recent years, there has been tremendous interest in the phenomenon of influence exerted by users of an online social network on other users and in how it propagates in the network. The idea is that when a user sees their social contacts performing an action such as joining an online community (say TapIt[30]), that user may decide to perform the action themselves. In truth, when a user performs an action, she may have any one of a number of reasons for doing so: she may have heard of it outside of the online social network and may have decided it is worthwhile; the action is very popular (e.g., buying an iPhone 4G may be such an action); or she may be genuinely influenced by seeing her social contacts perform that action. If there is genuine influence, it can be leveraged for a number of applications, arguably the most famous among which is viral marketing [42, 78, 109]. Other applications include personalized recommendations [124, 125] and feed ranking in social networks [117]. Besides, patterns of influence can be taken as a sign of user trust and exploited for computing trust propagation [49, 64, 127, 143] in large networks and in P2P systems.

[30]TapIt is a water bottle refilling network founded in 2008 to give people free access to clean sustainable water on the go: `tapitwater.com`.

While many of the applications mentioned above essentially assume that influence exists as a real phenomenon, two key pieces are missing from the picture. First, while some empirical studies have reported evidences of influence propagating on the social linkage [31, 68], others authors have challenged the fact that influence really exists and that it propagates between users. Indeed, Watts [136–138] challenges the very notion of influential users that are often assumed in viral marketing papers. As well, in a recent paper, Anagnostopoulos et al. [6] have developed techniques for showing that influence is *not* genuine and using them, showed that in the Flickr tagging data, while there is substantial social correlation in tagging behavior, it cannot be attributed to influence. This raises the question, is there evidence of genuine influence in any real social network data? The second missing piece is a systematic study of models of influence. In particular, all viral marketing papers assume that they are given as input a social graph with edges labeled by the probability with which a user's action will be influenced by her neighbor's actions. *To our knowledge, the question how or from where one can compute these probabilities of influence has been largely left open.*

In this chapter, our goal is to address both the issues above: we devise various probabilistic models of influence, and w.r.t. them we show that influence is genuinely happening in a real-world social network.

The starting observation is that while real social networks don't come with edges labeled with influence probabilities, they do come with an *action log*. Informally, an action log is a table that chronicles any actions performed by every user. It is thus interesting to ask whether by analyzing the action log together with the network, we can study the two questions above. In undertaking such a study, two things should be kept in mind. First, any models proposed for influence should be compatible with the assumptions made in applications such a viral marketing. Viral marketing papers typically assume a diffusion model of influence which satisfies a property known as submodularity. While we defer a formal definition to Section 4.3, intuitively it can be thought of as a law of diminishing returns. Second, the action log is huge in size. Thus, any algorithms developed for learning and testing the influence models should make minimal number of scans over this data.

In this chapter, we make the following contributions:

- We propose a solution framework. By starting with the diffusion models assumed in viral marketing, we elicit desiderata for models of probabilistic influence. These include a mandatory *submodularity* property and a desirable, but not mandatory *incrementality* property.

- We propose a variety of probabilistic models of influence between users in a social network. We show that all of them satisfy submodularity while all with the exception of one satisfies incrementality. Intuitively, an incremental model allows efficient testing.

- We develop algorithms for learning (the parameters of) all the proposed models, taking as input a social graph and an action log. We optimize the scans and show that our algorithms can learn all the models in no more than two scans. A highlight is that some of our models let us predict not just whether a user will perform an action but the time by which she will perform it.

- One of the most accurate models is what we call *continuous time* model. Unfortunately, it is the most expensive to test (it's not incremental). To mitigate this, we develop an approximate model called *discrete time* model, which is incremental and is much more efficient to test.

- It turns out evaluating (testing) the learned models is far from trivial. Thereto, we develop algorithms for testing all the models learned. The algorithms require just one scan over the action log.

- Last but not the least, we put the models and algorithms to test on the Flickr data set where for actions, we take users joining online communities. The data set consists of a graph with 1.3M nodes and more than 40M edges and an action log with 35M tuples referring to 300K different actions. Our results show there is genuine influence between users. In particular, we introduce the metrics of *user influenceability* and *action influence quotient*. Users (actions) with high values for this metric do experience genuine influence compared to those with low values. Our results also show that all proposed models have a reasonable performance, and continuous time model has the best performance. Discrete time model has an almost identical performance while it is much more efficient to test. We also show that our models can predict the time at which a user will perform an action with an impressive error margin.

In Section 4.2, we give a formal statement of the problem studied while in Section 4.3, we develop a solution framework and in Section 4.4 we present the models for probabilistic influence. Section 4.5 presents the algorithms for learning the models and for evaluating them, while Section 4.6 presents results from an extensive set of experiments.

4.2 Problem Definition

We are given a *social graph* in the form of an undirected graph $G = (V, E, \mathcal{T})$ where the nodes V are users. An undirected edge $(u, v) \in E$ between users u and v represents a social tie between the users. $\mathcal{T} : E \to \mathbb{N}$ is a function labeling each edge with the timestamp at which the social tie was created.[31] We're also given an *action log*, a relation $Actions(User, Action, Time)$, which contains a tuple (u, a, t_u) indicating that user u performed action a at time t_u. It contains such a tuple for every action performed by every user of the system. We will assume that the projection of *Actions* on the first column is contained in the set of nodes V of the social graph G. In other words, users in the *Actions* table correspond to nodes of the graph. We let $\mathcal{A}$ denote the universe of actions. In the following, we assume for ease of exposition that a user performs an action at most once. We denote with A_u the number of actions performed by user u in the training set, with $A_{u\&v}$ the number of actions performed by both u and v in the training set, with $A_{u|v}$ the number of actions either u or v performs in the training set. Clearly, $A_{u|v} = A_u + A_v - A_{u\&v}$. We also use A_{v2u} to denote the number of actions propagated from v to u in the training set. We next define propagation of actions.

[31] For convenience, we assume social ties are never broken. This assumption is inessential.

Definition 1 (Action propagation). *We say that an action $a \in \mathcal{A}$ propagates from user v_i to v_j iff: (i) $(v_i, v_j) \in E$; (ii) $\exists (v_i, a, t_i), (v_j, a, t_j) \in Actions$ with $t_i < t_j$; and (iii) $\mathcal{T}(v_i, v_j) \leq t_i$. When this happens we write $prop(a, v_i, v_j, \Delta t)$ where $\Delta t = t_j - t_i$.*

Notice that there must be a social tie between v_i and v_j, both must have performed the action after the moment in which their social tie was created. This leads to a natural notion of a propagation graph, defined next.

Definition 2 (Propagation graph). *For each action a, we define a propagation graph $PG(a) = (V(a), E(a))$, as follows. $V(a) = \{v \mid \exists t : (v, a, t) \in Actions\}$; there is a directed edge $v_i \xrightarrow{\Delta t} v_j$ in $E(a)$ whenever $prop(a, v_i, v_j, \Delta t)$.*

The propagation graph consists of users who performed the action, with edges connecting them in the direction of propagation. Observe that the propagation graph is a DAG. Each node can have more than one parent; it is directed, and cycles are impossible due to the time constraint which is the basis for the definition of propagation. Note that the propagation graph can possibly have disconnected components. In other words, the propagation of an action is just a directed instance (a flow) of the undirected graph G, and the log of actions $Actions(User, Action, Time)$ can be seen as a collection of propagations. When a user performs an action, we say that it is activated w.r.t. that action. Once a user activates, it becomes contagious and cannot de-activate. It may now influence all its inactive friends. The power to influence the neighbors is what we model as influence probability. The problem we tackle in this chapter is how to learn influence probabilities among the users, by mining the available set of past propagations. Formally, we want to learn a function $p : E \to [0, 1] \times [0, 1]$ assigning to both directions of each edge $(v, u) \in E$ the probabilities: $p_{v,u}$ and $p_{u,v}$.

4.3 Solution Framework

In the following, we introduce the framework we adopt which is an instance of the General Threshold Model. Consider an inactive user u and the set of its activated neighbors S, and suppose that each neighbor $v \in S$ activates after v and u became neighbors. To predict whether u will activate, we need to determine $p_u(S)$, the joint influence probability of S on u. If $p_u(S) \geq \theta_u$, where θ_u is the activation threshold of user u, we can conclude that u activates. For ease of exposition, assume individual probabilities of influence between users are static, i.e., are independent of time. We do not need this assumption for our results. Since influence probabilities are meant for use in viral marketing [78] [79], our definitions must be consistent with the diffusion models used in these papers. These papers typically assume the diffusion models are *monotone*, which says the function $p_u(S)$ should satisfy: $p_u(S) \leq p_u(T)$ whenever $S \subseteq T$. Moreover, it should be *submodular*, i.e., $p_u(S \cup \{w\}) - p_u(S) \geq p_u(T \cup \{w\}) - p_u(T)$ whenever $S \subseteq T$. There can be various ways to define $p_u(S)$. In this chapter, for computational ease, we assume that the probability of various friends influencing u are independent of each other. Hence, the joint probability $p_u(S)$ can be defined as follows:

$$p_u(S) = 1 - \prod_{v \in S} (1 - p_{v,u}) \tag{4.1}$$

In the context of testing a learned model, we need to be able to compute and update the influence probabilities on the fly. That is, as new neighbors get activated, the joint influence probability needs to be updated. We should be able to compute $p_u(S \cup \{w\})$ incrementally without revisiting the neighbor set influence probabilities, i.e., solely in terms of $p_u(S)$ and $p_{w,u}$. This is not part of the requirement imposed by the diffusion models assumed for viral marketing. Thus, this is a desirable, but not mandatory property.

Theorem 3. *The joint influence probability as defined in Eq. 4.1 is monotone and submodular. Besides, it can be updated incrementally if the individual influence probabilities $p_{v,u}$ are static.*

Proof. Let S be the set of neighbors of u that are active and suppose a new neighbor w of u gets activated. The new joint influence probability $p_u(S \cup \{w\})$ can be computed incrementally from $p_u(S)$ as follows.

$$\begin{aligned} p_u(S \cup \{w\}) &= 1 - (1 - p_{w,u}) * \prod_{v \in S} (1 - p_{v,u}) \\ &= 1 - (1 - p_{w,u}) * (1 - p_u(S)) \\ &= p_u(S) + (1 - p_u(S)) * p_{w,u} \end{aligned} \tag{4.2}$$

The monotonicity can be seen from Eq. 4.2. The difference $p_u(S \cup \{w\}) - p_u(S)$ is clearly non-negative as the domain of individual probabilities is $[0, 1]$. Similarly, submodularity can be shown as follows.

$$\begin{aligned} p_u(S \cup \{w\}) - p_u(S) - p_u(T \cup \{w\}) + p_u(T) \\ = (1 - p_u(S)) * p_{w,u} - (1 - p_u(T)) * p_{w,u} \\ = (p_u(T) - p_u(S)) * p_{w,u} \geq 0 \end{aligned}$$

since, by monotonicity, $p_u(T) \geq p_u(S)$. □

User Influenceability. As mentioned in the introduction, there can be three reasons which prompt any user to perform an action. First, influence from friends and family members. Second, she is affected by some external event(s). And the last is that she is a very active user and is doing things without getting influenced by anyone.

In this work, we mainly focus on modeling and learning the influence propagation from neighbors. For some users, external influence plays a significant role and for others that is not the case. Users who are initiators of actions and who are more influenced by external factors are unpredictable or less influenceable. So, we define an *influenceability score* representing how influenceable a user is, as the ratio between the number of actions for which we have evidence that the user was influenced, over the total number of actions performed by the user. More precisely we define:

$$infl(u) = \frac{|\{a \mid \exists v, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}\}|}{A_u} \tag{4.3}$$

In the equation above, we can use any appropriate value for the parameter $\tau_{v,u}$. We propose to use the average time delay, defined as follows:

$$\tau_{v,u} = \frac{\sum_{a \in \mathbf{A}} (t_u(a) - t_v(a))}{A_{v2u}} \tag{4.4}$$

where $t_u(a)$ is the time when u performs a and $\mathbf{A}$ is the set of actions in the training data. We conjecture that users with a high value for $infl(u)$ may exhibit a high degree of being influenced by their neighbors compared to those with a low value for this metric.

Action Influenceability. We define the *influence quotient* for an action to distinguish between actions for which there is more evidence of influence propagation from the rest of the actions. More precisely, we define:

$$infl(a) = \frac{|\{u \mid \exists v, \Delta t : prop(a, v, u, \Delta t) \wedge 0 \leq \Delta t \leq \tau_{v,u}\}\}|}{\text{number of users performing } a} \tag{4.5}$$

We expect that for actions with high $infl(a)$, predictions (based on influence models) of user performing those actions will yield a relatively higher precision and recall values compared to other actions. We will revisit this in the experimental section.

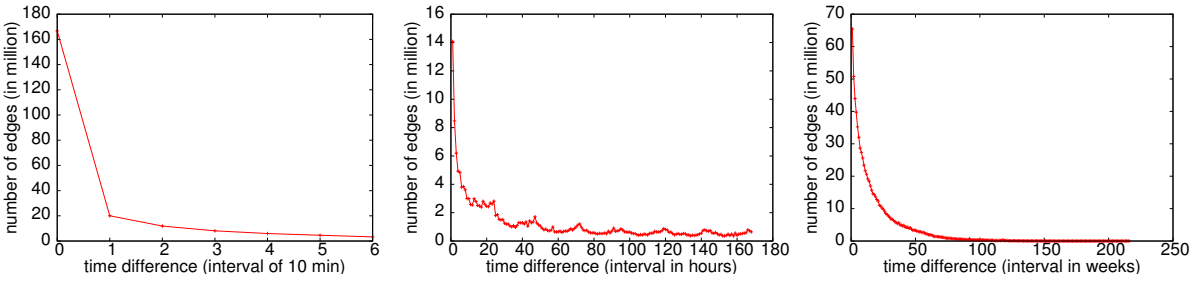
4.4 Models

Recall from the previous section that we assume the probabilities of influence by individual neighbors of a user are independent. Thus, if we have a model for capturing individual influences, we can compute the joint influence using Eq. 4.1. Next, we propose 3 types of models to capture $p_{v,u}$, the probability with which u is influenced by its neighbor v. The first class of models assumes the influence probabilities are static and do not change with time. The second class of models assumes they are continuous functions of time. As a preview, it will turn out continuous time models are by far the most accurate, but they are very expensive to test on large data sets. Thus, we propose an approximation known as Discrete Time Models where the joint influence probabilities can be computed incrementally and thus efficiently. In all the models we propose, we also discuss how to learn estimates of various parameters from the training data set.

4.4.1 Static Models

These models are independent of time and are the simplest to learn and test. We present three instances of static models.

Bernoulli distribution. Under this model, any time a contagious user v tries to influence its inactive neighbor u, it has a fixed probability of making u activate. If u activates, it is a successful attempt. Each attempt, which is associated with some action, can be viewed as a Bernoulli trial. The Maximum Likehood Estimator (MLE) of success probability is the ratio



(a) (b) (c)

Figure 4.1: Frequency of common actions vs the time difference between two users performing actions. (a) during the first hour at a granularity of 10 minutes; (b) during the first week at hourly granularity (without considering the cases in which the time difference is less than one hour, i.e., the cases in (a)); (c) the rest of the dataset with weekly granularity

of number of successful attempts over the total number of trials. Hence, influence probability of v on u using MLE is estimated as:

$$p_{v,u} = \frac{A_{v2u}}{A_v} \tag{4.6}$$

Jaccard Index. The Jaccard index is often used to measure similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. We adapt the Jaccard index to estimate $p_{v,u}$ as follows:

$$p_{v,u} = \frac{A_{v2u}}{A_{u|v}} \tag{4.7}$$

Partial Credits (PC). When a user u in a network is influenced to perform an action, it may be influenced by the combination of its neighbors who have performed the action before. Thus, it is reasonable to infer that each of these predecessors shares the "credit" for influencing u to perform that action. Suppose user u performs an action a at time $t_u(a)$ and S its set of activated neighbors such that $\forall v \in S, \mathcal{T}(v,u) \leq t_v(a) < t_u(a)$. Let $|S| = d$. Then, in the partial credits model, we give each of u's neighbors an equal credit $1/d$ for making u perform the action. In general, the credit given to user $v \in S$ who performed an action a before u can be defined as:

$$credit_{v,u}(a) = \frac{1}{\sum_{w \in S} I(t_w(a) < t_u(a))} \tag{4.8}$$

where I is an indicator function.

The notion of partial credit is orthogonal to whether we use Bernoulli or Jaccard as the base model. Thus, we have two additional models. In the *Bernoulli model with partial credit*,

the influence probability is estimated as follows:

$$p_{v,u} = \frac{\sum_{a \in \mathbf{A}} credit_{v,u}(a)}{A_v} \tag{4.9}$$

where $\mathbf{A}$ is the set of actions in the training data. In the *Jaccard model with partial credit*, the influence probability is estimated as follows:

$$p_{v,u} = \frac{\sum_{a} credit_{v,u}(a)}{A_{u|v}} \tag{4.10}$$

In case of static models, the joint influence probability $p_u(S)$ as defined in equation 4.1 can be computed incrementally as stated in Theorem 3.

4.4.2 Continuous Time (CT) Models

In reality, influence probability may not remain constant independently of time. It is natural to expect that when a user first comes to see/hear of its neighbor(s) performing an action, it may feel the urge to explore it and that, with time, this urge (i.e., influence) may decay. To understand whether this is true in practice, we computed the number of actions that propagated between pairs of neighbors in Flickr and plotted it against the time that elapsed between them: see Figure 9.3. The figure shows the behavior at three levels of granularity – weeks, hours, and intervals of 10 minutes. In all cases, the data consistently shows an exponential decay behavior, confirming our intuition above.

Accordingly, we define $p^t_{v,u}$, the probability of v influencing its neighbor u at time t, as follows.

$$p^t_{v,u} = p^0_{v,u} e^{-(t-t_v)/\tau_{v,u}} \tag{4.11}$$

where $p^0_{v,u}$ is the maximum strength of v influencing u. In the exponential decay model, the maximum strength is realized right after v performing the action, i.e., when $t = t_v$. This maximum strength, $p^0_{v,u}$, can be estimated in exactly the same was as for the static models. Thus, we can have four variations of the continuous time model corresponding to the four static models discussed earlier. We omit the obvious detail. The parameter $\tau_{v,u}$ is called the *mean life time*. It corresponds to the expected time delay between v performing an action and u performing the same action. Once $p^t_{v,u}$ is defined, we can derive the joint probability of influence, $p^t_u(S)$, of u being influenced at time t by the combination of its active neighbors, can be derived exactly as for static models. More precisely, we have:

$$p^t_u(S) = 1 - \prod_{v \in S} (1 - p^t_{v,u}) \tag{4.12}$$

The parameter $\tau_{v,u}$ can be estimated as the average time delay in propagating an action from v to its neighbor u in the training set. Formally, it is defined as in equation 4.4.

In this class of continuous time models, the joint influence probability $p^t_u(S)$ changes as each time step as it is a continuous function of time. As a new neighbor activates and becomes

contagious, there may be a sharp increase in $p_u^t(.)$ and then it starts decreasing again with time. Hence, the function $p_u^t(.)$ is a piecewise continuous function. Since the probability $p_{v,u}^t$ changes at each time step, the joint influence probability cannot be computed incrementally. Every time a new neighbor activates, we have to compute it from scratch. If the size of set S is d, then there would be at most d local maxima for the function. If $\max_t \{p_u^t(.)\} \geq \theta_u$, the activation threshold of u, we conclude that u activates.

In addition to predicting the activation state of an user, this class of models enables us to predict time at which the user is most likely to perform the action. The details appear in Section 4.5.3.

4.4.3 Discrete Time (DT) Models

As noted above, Continuous Time Models are not incremental in nature, hence they are very expensive in terms of run time required for testing. Therefore, in this section, we propose an approximation to Continuous Time Models, called *Discrete Time Models*. Here, we say that the influence of an active user v on its neighbor u remains constant at $p_{v,u}$ for a time window of $\tau_{v,u}$ after v performs the action. After that it drops to 0, i.e. a user v is contagious for u in the time interval $[t_v, t_v + \tau_{v,u}]$. It allows us to use the incrementality property established in Theorem 3. Here, the definition of S needs to be modified such that it contains only contagious neighbors of u. Hence, when a contagious neighbor w becomes non-contagious, we need to update $p_u(S)$ and it can be incrementally updated as follows.

$$p_u(S \setminus w) = \frac{p_u(S) - p_{w,u}}{1 - p_{w,u}} \tag{4.13}$$

Analogous to Static Models, there can be 4 variations of Discrete Time models depending on how the constant influence probability $p_{v,u}$ is estimated, i.e., using Bernoulli, Jaccard, or their particl credit variants. For brevity, we only give the equation for partial credits here as other cases are easier.

The Partial Credit definition (4.8) for discrete time should be modified as

$$credit_{v,u}^{\tau_{v,u}}(a) = \frac{1}{\sum_{w \in S} I(0 < t_u(a) - t_w(a) \leq \tau_{v,u})} \tag{4.14}$$

where I is the indicator function. The new influence probability $p_{v,u}$ can be computed as stated in equations 4.9 and 4.10 with the new definition of *credit*. As we show in the experiments section, these models provide an efficient yet effective approximation for Continuous Time Models.

4.5 Algorithms

In this section, we present algorithms for learning the parameters of the various models proposed in the previous section, as well as algorithms for testing the learned models. One of the key aspects we pay attention to is efficiency of not just the training algorithms (which

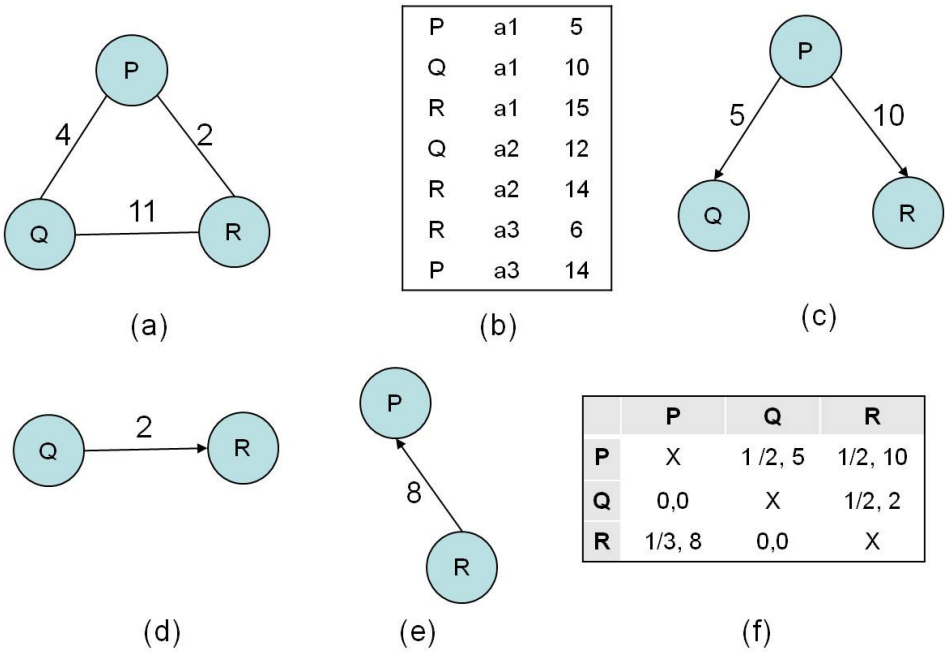


Figure 4.2: (a) Undirected social graph containing 3 nodes and 3 edges with timestamps when the social tie was created; (b) Action Log; (c) Propagation Graph for action a1 (d) Propagation Graph for action a2 (e) Propagation Graph for action a3 (f) Influence Matrix.

learn the parameters) but also that of the testing algorithms which apply the models on the test data and make predictions. As mentioned before, any model that enjoys the incremental property affords an efficient testing algorithm.

The input to these algorithms consists of a social graph together with an action log. We assume the action log is sorted on action-ids and tuples on an action are chronologically ordered. This allows the algorithms to process the data one action at a time. In practice, action log tends to be huge (from tens to hundreds of millions of tuples) so we optimize our algorithms to minimize the number of scans over the action log.

4.5.1 Learning the Models

As in any machine learning approach, the first step is to learn the parameters of a model. We note that *our algorithms are able to learn all the models simultaneously in no more than two scans of the (training sub-set of the) action log table.* Furthermore, to learn parameters for static and continuous time Models, the algorithm needs only one scan of the actions log. The overview is presented in Algorithm 8. To illustrate our algorithms, we will use a running example shown in Figure 4.2. Recall, action log is sorted on action-ids and then by time. The algorithm maintains the tuples for the current action a in a tree-based data structure *current_ table* indexed on user-ids. As it reads a new tuple of the form (u, a, t_u) saying user u performs the action a at time t_u, we look into the *current_ table* for those neighbors v of

u, such that the link between u and v has been established before either of them performs a. E^{t_v} represents the set of edges in the social graph at time t_v. It is worth noting that we don't need to assume social ties are never broken, as long as we can efficiently find E^{t_v}. Since the data is sorted in chronological order, $t_v \leq t_u$.

Algorithm 8 Learning - Phase1

```
1: for each action a in training set do
2:    current_table = φ
3:    for each user tuple < u, a, t_u > in chronological order do
4:       increment A_u
5:       parents = φ
6:       for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈ E^{t_v} do
7:          if t_u > t_v then
8:             increment A_{v2u}
9:             update τ_{v,u}
10:            insert v in parents
11:         end if
12:         increment A_{v&u}
13:      end for
14:      for each parent v ∈ parents do
15:         update credit_{v,u}
16:      end for
17:      add (u, a, t_u) to current_table
18:   end for
19: end for
```

Lines 1-6 of the algorithm are self-explanatory. The condition in line 7 ensures that we avoid the cases when the two users perform a at the same time as then it is questionable whether propagation actually happened. Next, for all the interesting neighbors from which the action propagated, we update the required counts/parameters (lines 8-11).

As an example, Figure 4.2(a) shows a social graph containing three users P, Q and R with three edges among them. The edges are labeled with timestamps at which the two users became friends. The action log containing 3 actions $a1$, $a2$ and $a3$ is presented in Figure 4.2(b). Using the social graph and action log, the propagation graphs for actions $a1$, $a2$ and $a3$ are shown in Figure 4.2(c), (d) and (e). Edges are directed in Propagation Graphs and labeled with time taken to propagate the action. Note that even though both Q and R perform $a1$, $PG(a1)$ doesn't contain the edge from Q to R because when Q performed $a1$ at time 10, R was not in its neighborhood. They became friends at time 11. Define the influence matrix of a model to be an $(n \times n)$ matrix IM, with $IM[i,j] = (p_{i,j}, \tau_{i,j})$ or $IM[i,j] = (p^0_{i,j}, \tau_{i,j})$, depending on whether it's a discrete or continuous time model. Figure 4.2(f) shows the influence matrix containing parameters learnt for Continuous Time Model with Bernoulli as the base model for computing $p^0_{i,j}$. For instance, the entry $IM[P,Q]$ shows that $p^0_{P,Q}$ is 1/2 and $\tau_{P,Q}$ is 5.

For learning partial credits, we maintain the tree-based data structure *parents* containing the contagious neighbors from which the action has been propagated. Next, the algorithm goes through the *parents* list again and updates the partial credits as defined in Eq. 4.8 (lines 12-13). Finally, the tuple (u, a, t_u) from the action log is added to *current_table* (line 14).

Notice that static models and continuous time models are learned in one scan.

To learn $infl(u)$, τ (learned from first scan) is needed and this requires a second scan of the action log. Similarly, learning parameters of discrete time models also requires τ beforehand. The second phase of the learning algorithm is described in Algorithm 9. The algorithm is very similar to Algorithm 8 except in Step 6 we require that $t_u - t_v \leq \tau_{v,u}$. Notice that $infl(u)$ is updated whenever we find at least one neighbor from which u is influenced.

Algorithm 9 Learning - Phase2

```
1: for each action a in training set do
2:    current_table = φ
3:    for each user tuple < u, a, t_u > in chronological order do
4:       parents = φ
5:       for each user v : (v, a, t_v) ∈ current_table && (v, u) ∈ E^{t_v} do
6:          if 0 < t_u − t_v < τ_{v,u} then
7:             increment A_{v2u}
8:             insert v in parents
9:          end if
10:      end for
11:      for each parent v ∈ parents do
12:         update credit_{v,u}^{τ_{v,u}}
13:      end for
14:      if parents != φ then
15:         update infl(u)
16:      end if
17:      add (u, a, t_u) in current_table
18:   end for
19: end for
```

4.5.2 Evaluating the Models

An overview of the evaluation algorithm for Static Models is given in Algorithm 10. We assume the same sort order for the action log as for the training algorithms. We maintain a *results_table* with entries of the form $< u, p_u, perform_u >$ as we scan the action log, where the flag $perform_u$ represents whether the user u has actually performed the action in question or not: its value is 0 if u never performs the action but at least one of its neighbors does, is 1 if u performs it and at least one of its neighbors performs it before it, and is 2 if u is the initiator of the action in its neighborhood. p_u represents the probability of the user u performing the action given its neighbors who have already performed the action. At any instant, the *results_table* contains all the activated users for the current action and their neighbors.

As we scan the actions log and read a new tuple of the form $< v, a, t_v >$, we add v and all its neighbors to the *results_table* with the appropriate influence probability and $perform_v$ flag. It may be possible that v is already present in the table because one or more of its neighbors are already active. In that case, we just update the $perform_v$ flag to 1 (lines 4-5). If it is not present in the table, then that means it is the initiator of the current action in its neighborhood, hence the $perform_v$ flag should be set to 2 (lines 6-7). Similarly, some of

its neighbors may already be present in the table and in that case, we update the influence probability of v over them incrementally (lines 8-10). At the end of reading all tuples for an action, the *results_ table* contains all the users who are active or are neighbors of one or more active users.

We depict the performance of our models using ROC curves, which plot the true positive rate (TPR = TP/(TP+FN)) against the false positive rate (FPR = FP/(FP+TN)), where TP represents true positives, FN represents false negatives etc. The appropriateness of ROC curves over precision recall curves for binary classification has been recognized [105]. The closer the hump of the curve to the point $(0, 1)$ the better the performance. In our problem setting, we ignore all the cases when none of the user's friends is active, as then the model is inapplicable. Hence, we define TP as cases when user performs the action, at least one of its neighbors performs the action before it and model estimates it performs the action. FP is the number of cases when user doesn't perform the action, at least one of its neighbors performs action and model estimates it performs the action. Similarly, TN is the number of cases when user doesn't perform the action, and at least one of its neighbors performs the action and the model estimates it doesn't perform the action. Finally, FN is the number of cases when the user performs the action, at least one of its neighbors performs the action before it and the model estimates it doesn't perform the action.

After processing all the tuples for an action, the algorithm scan the *results_ table* to update TP, FN, FP and TN (lines 13-17). It is straightforward to adapt this algorithm to evaluate Discrete Time Models. We omit the details here due to the lack of space.

Algorithm 10 Evaluate-Basic

```
 1: for each action a in test set do
 2:    results_ table = φ
 3:    for each user tuple < v, a, t_v > in chronological order do
 4:       if v ∈ results_ table then
 5:          set perform_v flag to 1
 6:       else
 7:          add v to results_ table with p_v=0 and perform_v=2
 8:       end if
 9:       for each user u : (v, u) ∈ E^{t_v} do
10:          if u ∈ results_ table then
11:             update p_u incrementally as in Theorem 1
12:          else
13:             add u to results_ table with appropriate p_u and perform_u=0
14:          end if
15:       end for
16:    end for
17:    for each entry < u, p_u, perform_u > in results_ table do
18:       if (perform_u == 1 && p_u ≥ θ_u) it is TP
19:       if (perform_u == 1 && p_u < θ_u) it is FN
20:       if (perform_u == 0 && p_u ≥ θ_u) it is FP
21:       if (perform_u == 0 && p_u < θ_u) it is TN
22:    end for
23: end for
```

In case of continuous time models, testing becomes complex. Algorithm 11 provides an overview. Here, we store in the *results_ table* the time t_u at which user u performs the action a. Once the *results_ table* is formed for an action (in lines 2-10), the algorithm iterates over all the entries in it. For each entry $\langle u, p_u, perform_u, t_u \rangle$, it collects all the relevant neighbors and keeps them in chronological order in *sorted_ parents* table (lines 11-14). Next, the algorithm tries to find the global maximum of joint influence probability of u performing the action w.r.t time. Whenever a new neighbor performs the action, the joint influence probability increases sharply and then starts decreasing again until another neighbor gets activated. So, if there are d neighbors who perform a, then the joint probability distribution would have (up to) d local maxima. To find the global maximum, we have to analyze all the local maxima (lines 15-18). If it is more than the threshold θ_u, then we conclude that u activates. The required metrics: TP, FN, FP and TN are updated accordingly (lines 19-23).

Algorithm 11 Evaluate-Complex

1: **for** each action a in test set **do**
2: *results_ table* = ϕ
3: **for** each user tuple $< u, a, t_u >$ in chronological order **do**
4: **if** $u \in$ *results_ table* **then**
5: set $perform_u$ flag to 1
6: **else**
7: add u to *results_ table* with p_u=0 and $perform_u$=2
8: **end if**
9: **for** each user $v : (v, u) \in E^{t_u}$ **do**
10: **if** $v \notin$ *results_ table* **then**
11: add v in *results_ table* with p_v=0 and $perform_v$=0
12: **end if**
13: **end for**
14: **end for**
15: *sorted_ parents* = ϕ
16: **for** each entry $< u, p_u, perform_u, t_u >$ in *results_ table* **do**
17: **for** each user $v : perform_v != 0, (v, u) \in E^{t_v}$ **do**
18: add v to *sorted_ parents*
19: **end for**
20: **for** each neighbor $v_i \in$ *sorted_ parents* list **do**
21: compute $p_u(t_{v_i})$ at time t_{v_i} considering neighbors from v_1 to v_i
22: **if** $p_u(t_{v_i}) > p_u$ **then**
23: update p_u
24: **end if**
25: **end for**
26: **end for**
27: **for** each entry $< u, p_u, perform_u, t_u >$ in *results_ table* **do**
28: if ($perform_u == 1$ && $p_u \geq \theta_u$) it is TP
29: if ($perform_u == 1$ && $p_u < \theta_u$) it is FN
30: if ($perform_u == 0$ && $p_u \geq \theta_u$) it is FP
31: if ($perform_u == 0$ && $p_u < \theta_u$) it is TN
32: **end for**
33: **end for**

As an example, consider the social graph and influence probabilities in Figure 4.2. Let us assume there is an action $a4$ in test set whose action log is given in Figure 4.3(a). The corresponding propagation graph is shown in Figure 4.3(b). Finally, the probability of R performing the action w.r.t. is plotted in 4.3(c).

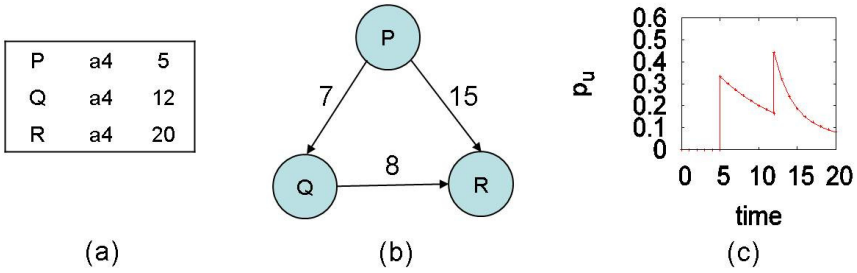


Figure 4.3: (a) Action Log; (b) Propagation Graph for action a4 (c) $p_R(.)$ w.r.t time for Continuous Time Model with Bernoulli.

4.5.3 Predicting Time

Time conscious models like CT and DT enable us not only to predict whether a user performs a particular action or not, but also to predict the time interval $[b, e]$ in which she is most likely to do it. We next explain how.

Whenever a new neighbor activates, there is a sharp increase in the joint influence probability which makes it a piecewise continuous function. We assume that user u is most likely to get activated in the first region where $p_u^t(.) \geq \theta_u$. In other terms, we take the first local maximum of the joint probability function which is not less than θ_u. We say the user has now entered into the contagious interval and can activate anytime. We label this time t as the left-bound of the interval b. For the right-bound e, we add to b the *half life period*[32] of the influence of v over u, or more specifically, $\tau_{v,u} * ln(2)$, where v is the neighbor of u by virtue of which u entered the contagious zone. Intuitively, by time $\tau_{v,u} * ln(2)$, half the actions that are propagated from w to u have indeed been picked up by u. Thus, we predict that given an action a that did propagate from v to u, by the half life period after v performs a, u would have performed a.

Since it would be complex to asses prediction accuracy w.r.t. an interval $[b, e]$ we decide for our experiments to predict an exact point in time. In viral marketing applications, tightness of lower bound is not critical, as in case the user performs the action early, it does not hurt. Hence, we decide to perform our experiments on the upper bound e.

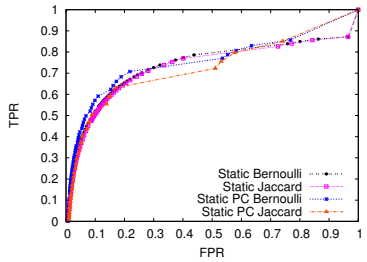


Figure 4.4: ROC comparisons of Static Models.

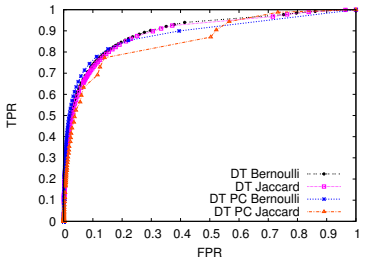


Figure 4.5: ROC comparisons of Discrete Time Models.

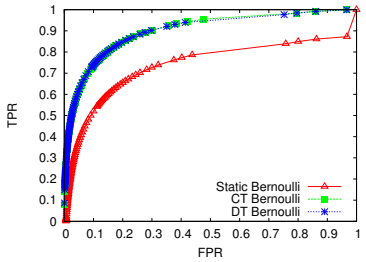


Figure 4.6: ROC comparisons of Static, CT, and DT models.

4.6 Experimental Evaluation

Dataset preparation. In our problem setting, we need both the social network and the actions log. We consider Flickr social network and we consider "joining a group" as the action[33]. We started with 6,2 millions users having 71 millions edges. We projected this graph on the subset of users who is a member of at least one group. This gave us 1,450,347 users with 40,562,923 edges among them.

This social graph has 34,766 connected components where the largest connected component consist of 1,319,573 users with 40,450,082 edges (99.72%). Rest of the components have less than 75 users, so we ignore them. Total number of tuples in action log after the filtering are 35,967,169. As in any machine learning approach, we split the dataset into a training and a test set. In our experiments, we split the dataset based on actions such that each action can appear completely either in training or test dataset.

Qualitative Evaluation. We compare the different models by means of ROC curves. Each point in ROC curve corresponds to one possible value of activation threshold θ_u which is same for all users. Figure 4.4 compares the four Static Models introduced in Section 4.4.1. Similarly, Figure 4.5 examines the 4 variants of Discrete Time (DT) models (Section 4.4.3). These figures show that Bernoulli is slightly better than Jaccard model and among two Bernoulli variants, Partial Credit wins by a small margin. In the rest of the section, we only use Bernoulli model to compare different classes of models.

Figure 4.6 shows the comparisons between the three different classes of models. It verifies the claim that time conscious models work far better than Static Models. Among time conscious models, CT and DT perform equally well.

Figure 4.7 reports the ROC curves for different slices of users influenceability (a) and actions influenceability (b). The plots confirm the intuition that larger influenceability leads to an easier prediction of influence.

Predicting Time. While for DT models it is inexpensive to compute both lower and upper bounds of the contagious time interval, we found the upper bound computed is not tight. Hence, we conclude that while DT models are good in predicting the activation state of the

[32] `http://en.wikipedia.org/wiki/Half-life`
[33] `http://www.flickr.com/groups`

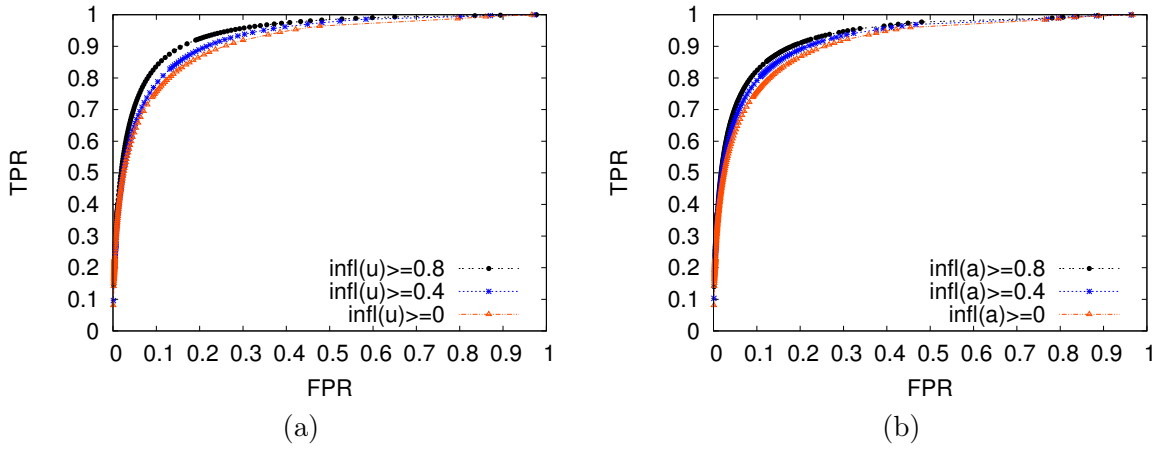


Figure 4.7: CT Bernoulli model: ROC curves for different slices of users influenceability (a) and actions influenceability (b).

users w.r.t. actions, they are not capable of predicting the time range in which the user is most likely to perform the action. Therefore in the following we focus on CT models (in particular Bernoulli).

In figures 4.8, 4.9 and 4.10, we show the power of CT models to predict the time by which users are likely to perform actions. The model attempts to predict the upper bound on time only when the user is active and prediction of activation state is correct, i.e. the TP cases. Figure 4.8 shows the root mean square error (RMSE) in days against the accuracy of predicting upper bound. Accuracy of time prediction is defined as the ratio between the number of cases when the prediction of upper bound by the model is correct, over the total number of cases it examines. In this plot we removed the 2.5% outliers both from the negative and positive side. The RMSE value revolves around 70-80 days. If we try to increase accuracy beyond 85%, RMSE increases sharply. In the subsequent figures, we choose an operating point for CT model corresponding to 82.5% TPR and 17.5% FPR. This is chosen as the intersection point of the ROC curve and diagonal line from [1,0] to [0,1]. At this point, Accuracy is 73% and RMSE is 75 days. In figure 4.9, X-axis is the error in predicting time and Y-axis is the number of times the CT model makes that error. It clearly shows that most of the time, the error in prediction is very small. Finally, Figure 4.10 shows the coverage of CT model. Here, a point (x, y) means that for y% of cases, the error is within $\pm$x. In particular, for 95% cases, the error is within 20 weeks.

Scalability evaluation. Figure 4.11 (a) and (b) shows the scalability of our algorithms. In the training phase, the runtime is a linear function of number of the tuples read. Memory usage (not reported in the figure) remains constant and independent of the number of tuples read, both in training and testing phases. Since Static and DT models are incremental in nature, they are far more efficient than CT models for testing. In conclusion, DT models achieves the same quality as CT models but much more efficiently.

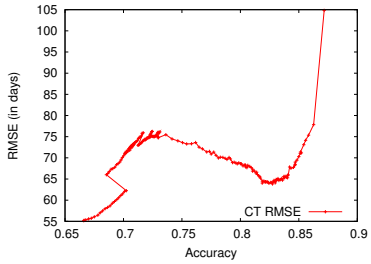


Figure 4.8: Root Mean Square Error (in days) vs Accuracy in predicting time for CT models.

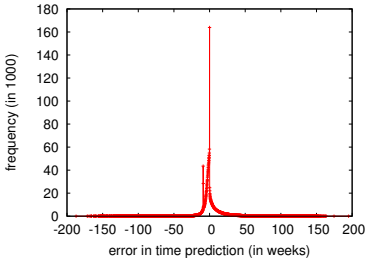


Figure 4.9: Distribution of error in time prediction.

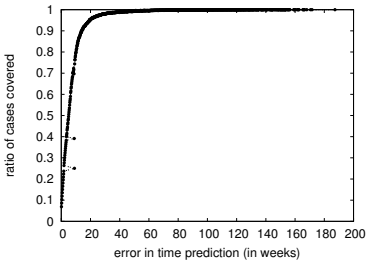
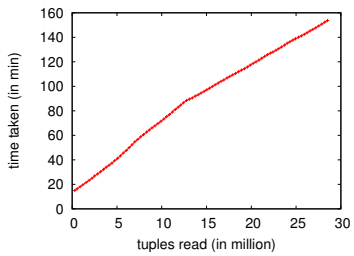
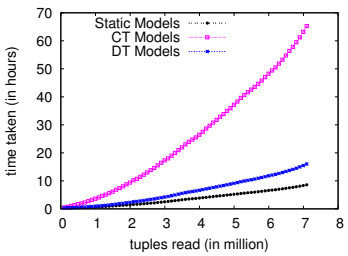


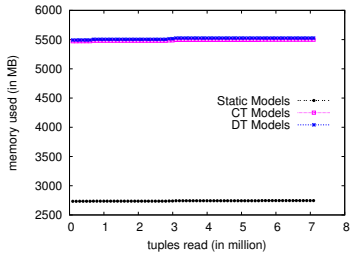
Figure 4.10: Coverage of CT models as a function of time prediction error.



(a)



(b)



(c)

Figure 4.11: (a) Training runtime. (b) Testing runtime comparison of all 3 classes of models. (c) Memory usage comparison of all 3 models in testing.

4.7 Conclusions and Discussion

Previous works about influence propagation in social networks typically assume the social graph which is input to the problem has its edges labeled with probabilities of influence. In this chapter we studied how to learn such probabilities form a log of past propagations. We proposed both static and time-dependent models for capturing influence, presented algorithms for learning the parameters of the various models and for testing the models. Our algorithms are optimized to minimize the scans over the action log, a key input to the problem of inferring probabilities of influence. This is significant since the action log tends to be huge. We ran an extensive set of experiments to test our learning algorithms. One of the highlights is that in addition to predicting whether a user will perform an action, we also observed that the predictions of our algorithms on users with a high influenceability score tend to have a high precision. In addition, we are able to predict, to within tight margins, the time by which an influenced user will perform an action after its neighbors have performed the action. In addition to demonstrating these aspects, our experiments also show that while testing the proposed continuous time model is very expensive, the discrete time model can be tested much more efficiently and yet can yield accuracy levels very close to that of the continuous

time model.

Several challenges remain. While this work was motivated by the assumptions of viral marketing, it's interesting to consider the impact of this work in the reverse direction. For instance, viral marketing works essentially ignore the effect of time and assume edges have constant influence probabilities as labels. It is important to formulate and solve viral marketing taking into account the time varying nature of influence. Similarly, factoring in user influenceability and action influence quotient is important. Indeed, a user with low influenceability might "attenuate" some of the incoming influence from neighbors. The same user may be more influenced by neighbors on actions with high influence quotient than on other actions. Accounting for these phenomena in viral marketing is an interesting direction for future work. Last but not the least, learning optimal user activation thresholds would be an interesting future work.

Chapter 5

A Data-Based Approach to Social Influence Maximization

Influence Maximization is the problem of finding a set of users in a social network, such that by targeting this set, one maximizes the expected spread of influence in the network. Most of the literature on this topic has focused exclusively on the social graph, overlooking historical data, i.e., traces of past action propagations. In this chapter, we study Influence Maximization from a novel data-based perspective. In particular, we introduce a new model, which we call *credit distribution*, that directly leverages available propagation traces to learn how influence flows in the network and uses this to estimate expected influence spread. Our approach also learns the different levels of influenceability of users, and it is time-aware in the sense that it takes the temporal nature of influence into account.

We show that Influence Maximization under the credit distribution model is NP-hard and that the function that defines expected spread under our model is submodular. Based on these, we develop an approximation algorithm for solving the Influence Maximization problem that at once enjoys high accuracy compared to the standard approach, while being several orders of magnitude faster and more scalable.

This chapter is based on our VLDB 2011 paper [57]. This investigation was done in collaboration with Francesco Bonchi and Laks V. S. Lakshmanan.

5.1 Introduction

Motivated by applications such as viral marketing [42], personalized recommendations [124], feed ranking [72], and the analysis of Twitter [13, 139], the study of the propagation of influence exerted by users of an online social network on other users has received tremendous attention in the last years. One of the key problems in this area is the identification of influential users, by targeting whom certain desirable outcomes can be achieved. Here, targeting could mean giving free (or price discounted) samples of a product and the desired outcome may be to get as many customers to buy the product as possible. Kempe et al. [78] formalized this as the Influence Maximization problem: find k "seed" nodes in the network, for a given number k, such that by activating them we can maximize the *expected influence spread*, i.e., the expected number of nodes that eventually get activated, according to a chosen *propagation model*. The propagation model governs how influence diffuses or propagates through the network (see Chapter 2.1 for background on the most prominent propagation models adopted by [78]). Following this seminal paper, there has been substantial work in this area (see Chapter 2). In this chapter we study Influence Maximization as defined by Kempe et al., but from a

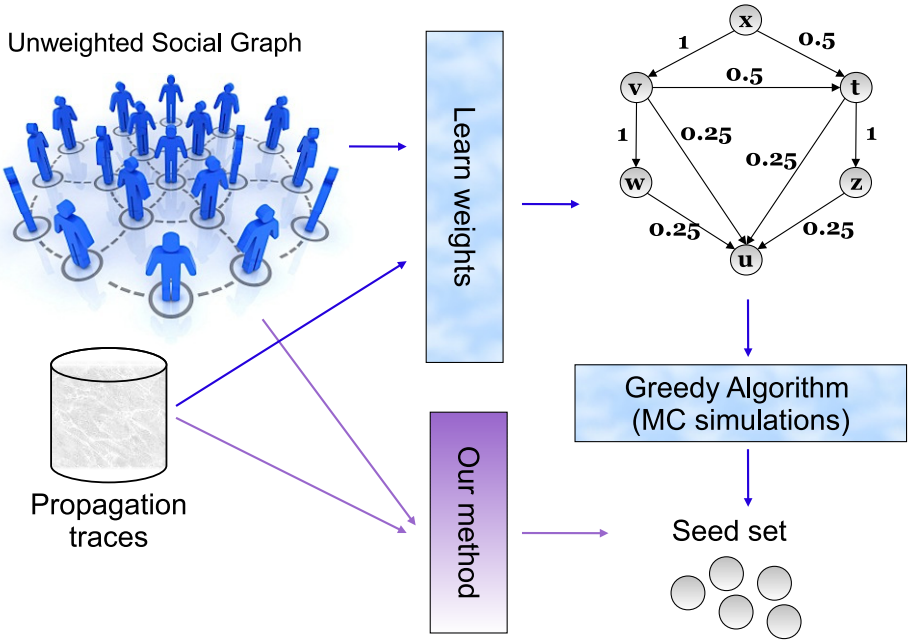


Figure 5.1: The standard Influence Maximization process (in light blue), and our approach (in magenta).

novel, data-based perspective.

Influence Maximization requires two kinds of data – a directed graph G and an assignment of probabilities (or weights) to the edges of G, capturing degrees of influence. E.g., in Figure 5.1, the probability of the edge (v, u) is 0.25 and it says there is a probability 0.25 with which user v influences u and thus v's actions will propagate to u with probability 0.25. In real life, while the digraph representing a social network is often explicitly available, edge probabilities are not. Facing difficulties in gathering real action propagation traces from which to "learn" edge probabilities, previous work has resorted to simply making assumptions about these probabilities. The methods adopted for assignment of probabilities to edges include the following: (i) treating them as constant (e.g., 0.01), (ii) drawing values uniformly at random from a small set of constants, e.g., $\{0.1, 0.01, 0.001\}$ in the so-called trivalency "model", or (iii) defining them to be the reciprocal of a node's in-degree, in the so-called weighted cascade "model" (see e.g., [35, 36, 78]). Only recently researchers have shown how to *learn* the edge probabilities from real data on past propagation traces of actions performed by users (nodes) [56, 114].

Given that there have been several ad hoc assumptions about probability assignment as well as recent techniques for learning edge probabilities from real data, some natural questions arise. What is the relative importance of the graph structure and the edge probabilities in the Influence Maximization problem? To what extent different methods of edge probability assignment accurately describe the influence propagation phenomenon? In particular, how do the various edge probability assignments considered in earlier literature compare with proba-

bilities learned from real data when it comes to accurately predicting the expected influence spread? Learning edge probabilities from real data is prone to error either owing to noise in the data or to the inherent nature of mining these probabilities. How robust are solutions to Influence Maximization against such noise?

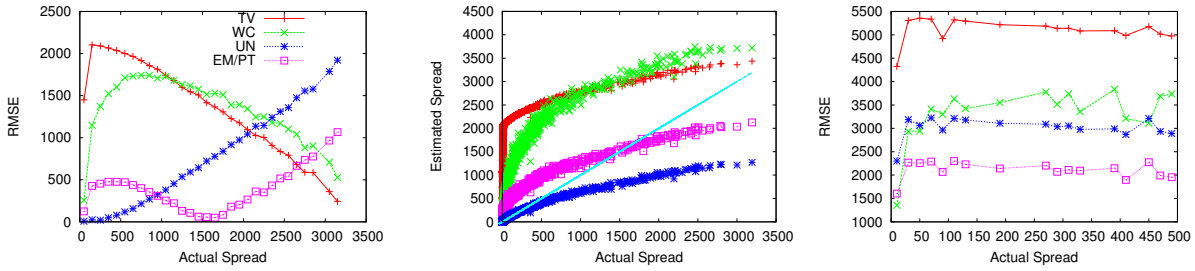
As we discuss in the Chapter 2, the Influence Maximization process based on Monte Carlo (MC) simulation is computationally expensive, even when the edge probabilities are given as input. Having to learn these probabilities, from a large database of traces, only adds to the complexity. *Can we avoid the costly learning and simulation approach, and directly mine the available log of past action propagation traces to build a model of the spread of any given seed set?*

Our research is driven by the questions above, and it achieves the following contributions.

- We conduct a detailed empirical evaluation of different methods of edge probability assignment as well probabilities learned from real propagation traces and show that methods that don't learn probabilities from real data end up choosing very different seed sets than those that do. Secondly, we show the spread predicted by methods based on edge probability assignment suffers from large errors, compared to methods that learn edge probabilities from real data. This offers some evidence that the former class of methods risk choosing poor quality seeds (Section 5.2).
- We develop a new model called *credit distribution*, built on top of real propagation traces that allows us to directly predict the influence spread of node sets, without any need for learning edge probabilities or conducting MC simulations (Section 5.3).
- We show that Influence Maximization under credit distribution is NP-hard. However, we show the function defining influence spread under this model is monotone and submodular. Using this, we develop a greedy algorithm that guarantees a $(1-1/e)$-approximation to the optimal solution and is scalable (Section 5.4).
- We conduct a comprehensive set of experiments on large real-world datasets (Section 5.5). We compare our proposal against the standard approach of [78] with edge probabilities learned from real data, and show that the credit distribution model provides higher accuracy. We also demonstrate the scalability of our approach by showing our results on very large real world networks, on which the standard approach is not practical.

5.2 Why Data Matters

What is the relative importance of the network structure and the edge probabilities in determining influence propagation? How important is it to accurately learn probabilities from real propagation traces? We have seen that a large majority of the literature assumes edge probabilities to be randomly chosen from an arbitrary fixed set or to be determined by node degrees. How do these methods compare with that of learning edge probabilities from real data, in terms of the quality of seeds selected? To answer this, we compare the performance of Algorithm 1 under the IC model, with different methods of assigning edge probability. To this end, we present two kinds of experiments that, to the best of our knowledge, have never been reported before.



(a) (b) (c)

Figure 5.2: Error as a function of Actual Spread on (a) FLIXSTER_SMALL, (c) FLICKR_SMALL; (b) Scatter plot of predicted spread vs. actual spread on FLIXSTER_SMALL. The legend in all the plots follows from (a).

Datasets. We take two real world datasets: Flixster and Flickr, both consisting of an unweighted directed social graph, along with an associated *action log*. An action log is a set of triples (u, a, t) which say user u performed action a at time t. We refer to the set of triples in the action log corresponding to a specific action a as the *propagation trace* (propagation for short) associated with a. Flixster (`www.flixster.com`) is one of the main players in the mobile and social movie rating business [74]. Here, an action is a user rating a movie. In other words, if user v rates "The King's Speech", and later on v's friend u does the same, we consider the action of rating "The King's Speech" as having propagated from v to u. Flickr is a popular photo sharing platform. Here, an action is a user joining an interest group (e.g., *"Nikon Selfportrait"*, *"HDR Panoramas"*). The raw versions of both datasets are very large and as a result, experiments that require repeated MC simulations cannot be run within any reasonable time on the full data set. While the large version of the datasets are useful for testing scalability of our proposal, for other experiments we have to sample smaller datasets. In what follows, we use samples that correspond to taking a unique "community", obtained by means of graph clustering performed using *Graclus*[34]. The resulting datasets are named FLIXSTER_SMALL and FLICKR_SMALL (statistics in Table 5.1).

	FLIXSTER LARGE	FLICKR LARGE	FLIXSTER SMALL	FLICKR SMALL
#Nodes	1M	1.32M	13K	14.8K
#Dir. Edges	28M	81M	192.4K	1.17M
Avg.degree	28	61	14.8	79
#propagations	49K	296K	25K	28.5K
#tuples	8.2M	36M	1.84M	478K

Table 5.1: Statistics of datasets.

One of the goals of the experiments is to determine which method more accurately predicts the expected spread of node sets. So we split the action log into two sets of propagation traces

[34] `http://www.cs.utexas.edu/users/dml/Software/graclus.html`

– training and test sets. The edge probabilities are learnt from the training set and thus, it is crucial that the splitting is performed in such a way that a propagation trace in its entirety falls into training or test set. Taking care that similar distributions of propagation sizes are maintained in the two sets, we place 80% and 20% of the propagations in training and test set respectively. Precisely, we sorted the propagation traces based on their size and put every fifth propagation in this ranking in the test set. As a result, the number of propagations in the training set are 5.1K and 5.7K for FLIXSTER_SMALL and FLICKR_SMALL respectively. The number of tuples in the training set are 1.5M and 385.3K respectively. The training set is used to learn the edge probabilities according to the EM-based method of Saito *et* al. [114]. One issue in using their method is that in their work, Saito *et* al. assume that the input action log data is as though it was generated by an IC model: i.e., time is discrete, and if user u activates at time t, then at least one of the neighbors of u was activated at time $t-1$. In real-world propagations this is not the case. To close this gap between their model and the real data, we let all previously activated neighbors of a node be its possible influencers.

Methods experimented. In both our experiments, we consider the IC model together with the following methods of edge probability assignment based on previous work [35, 36, 78, 114]:

WC: probability on an edge (v, u) is $1/\text{in-degree}(u)$ (known as weighted cascade);

TV: probabilities are selected uniformly at random from the set $\{0.1, 0.01, 0.001\}$ (trivalency).

UN: all edges are uniformly assigned probability $p = 0.01$.

EM: probabilities are learned from the training set using the EM-based method [114].

PT: Finally, in order to assess how robust the greedy method is to noise in the probability learning phase, we take EM-learnt probability and add noise. More precisely, for each edge (v, u) we randomly pick a percentage from the interval $[-20\%, 20\%]$ to perturb $p_{v,u}$, rounding to 0 or 1 in cases that go below 0 or over 1 respectively. We call this method **PT** (EM *perturbed*).

Experiment 1: Seed set intersection. The goal of this experiment is to understand the extent to which choice of edge probabilities affects the decisions of different methods in seed set selection. We run Algorithm 1 under the IC model[35] with the various methods of probability assignment above, as well as with edge probabilities learned from the training data set using EM. In each case, we used the algorithm to produce a seed set of size $k = 50$. Table 5.2 reports the size of the intersection for each pair of seed sets. We can see that EM, the method using real data to learn the influence probabilities has a very small, almost empty, intersection with all other methods, with the exception of its own perturbed version PT. Thus, we conclude *all methods that use edge probabilities based on ad hoc assumptions select seed sets very different from the method that uses propagation trace data to learn edge probabilities. Secondly, noise in the learned edge probabilities does not affect the seed set selection too drastically, as shown by the intersection between* EM *and* PT. What can we say about the *quality of seed sets* chosen by the methods UN, TV, and WC, compared to EM? This is addressed next.

[35]We found the greedy algorithm on FLICKR_SMALL is too slow to complete in a reasonable time, even with CELF optimization. Hence we use the PMIA heuristic [35] (discussed in Section 2.2) in order to speed up IC

	UN	WC	TV	EM	PT
UN	50	25	5	6	6
WC		50	9	3	2
TV			50	3	2
EM				50	44
PT					50

FLIXSTER_SMALL

	PT	EM	TV	WC	UN
UN	0	0	44	19	50
WC	0	0	17	50	
TV	0	0	50		
EM	44	50			
PT	50				

FLICKR_SMALL

Table 5.2: Size of seed set intersection for $k = 50$ on FLIXSTER_SMALL (left) and FLICKR_SMALL (right).

Experiment 2: Spread prediction. In the second experiment, we address the question, how good each of the methods is at predicting the actual spread. For that end, for a given seed set S, we compute the expected spread $\sigma_{IC}(S)$ predicted by each of the methods and compare it with the actual spread of S according to ground truth. For ground truth, for each propagation (movie in Flixster and group in Flickr) in the test set, we take the set of users that are the first to rate the movie (or join the group in case of Flickr) among their friends, i.e., the set of "initiators" of the action, to be the seed set. The actual spread is the number of users who performed that action, also called propagation size. This allows for a fair comparison of all methods from a neutral standpoint, which is a first in itself.

Figures 5.2(a) and (c) report the root mean squared error (RMSE) between predicted and actual spread on the two datasets: propagations in the test set are grouped in bins with respect to their size[36] and RMSE is computed inside each bin. On FLIXSTER_SMALL, uniform method (UN) works well but only for small propagations, and trivalency (TV) and weighted cascade (WC) work well but only for very large propagations (which are only few cases, i.e., outliers), and this is explainable with the fact that they *always* tend to predict the spread as very high. This is clearly shown in Figure 5.2(b), which shows a scatter plot between predicted and actual spread on FLIXSTER_SMALL.

In the case of FLICKR_SMALL, EM clearly outperforms all the other methods for all sizes of actual spread (Figure 5.2(c)). In all cases, the performance of EM and PT are so close that they are almost indistinguishable.

Overall, even if EM tends to underestimate the spread when this gets larger, *it is by far the most accurate method with respect to the ground truth*. The other methods that do not use the real propagation traces to learn influence probabilities are found to be unreliable in predicting the true spread.

By putting the results of the two experiments together, we can draw the first conclusion of this chapter: methods UN, TV, and WC select seed sets that are very different from EM and since they can be quite inaccurate in predicting the true spread, they can end up selecting seed sets of poor quality. It is thus extremely important to exploit available past propagation traces to learn the probabilities right. This finding strengthens the motivation for the rest of our work.

computation. [35] empirically showed PMIA produces results very close to greedy.

[36] In FLIXSTER_SMALL bins are defined at multiples of 100, in FLICKR_SMALL at multiples of 20.

5.3 Credit Distribution Model

The propagation models discussed in Section 2.1 are probabilistic in nature. In the IC model, coin flips decide whether an active node will succeed in activating its peers. In the LT model it is the node threshold chosen uniformly at random, together with the influence weights of active neighbors, that decides whether a node becomes active. Under both models, we can think of a propagation trace as a *possible world*, i.e., a possible outcome of a set of probabilistic choices.

Given a propagation model and a directed and edge-weighted social graph $G = (V, E, p)$, let $\mathbb{G}$ denote the set of all possible worlds. Independently of the model m chosen, the expected spread $\sigma_m(S)$ can be written as:

$$\sigma_m(S) = \sum_{X \in \mathbb{G}} Pr[X] \cdot \sigma_m^X(S) \tag{5.1}$$

where $\sigma_m^X(S)$ is the number of nodes reachable from S in the possible world X. The number of possible worlds is clearly exponential. Indeed, computing $\sigma_m(S)$ under the IC and LT models is #P-hard [35, 37], and the standard approach (see [78]) tackles influence spread computation from the perspective of Eq. (5.1): sample a possible world $X \in \mathbb{G}$, compute $\sigma_m^X(S)$, and repeat until the number of sampled worlds is large enough. We now develop an alternative approach for computing influence spread, by rewriting Eq. (5.1), giving a different perspective. Let $path(S, u)$ be an indicator random variable that is 1 if there exists a directed path from the set S to u and 0 otherwise. Moreover let $path_X(S, u)$ denote the outcome of the random variable in a possible world $X \in \mathbb{G}$. Then we have:

$$\sigma_m^X(S) = \sum_{u \in V} path_X(S, u) \tag{5.2}$$

Substituting in (5.1) and rearranging the terms we have:

$$\sigma_m(S) = \sum_{u \in V} \sum_{X \in \mathbb{G}} Pr[X]\, path_X(S, u) \tag{5.3}$$

From the definition of expectation, we can rewrite this to

$$\sigma_m(S) = \sum_{u \in V} E[path(S, u)] = \sum_{u \in V} Pr[path(S, u) = 1] \tag{5.4}$$

That is, the expected spread of a set S is the sum over each node $u \in V$, of the probability of the node u getting activated given that S is the initial seed set.

The standard approach samples possible worlds from the perspective of Eq. (5.1). To leverage available data on real propagation traces, we observe that these traces are similar to possible worlds, except they are *"real available worlds"*. Thus, in this chapter, we approach the computation of influence spread from the perspective of Eq. (5.4), i.e., we estimate directly $Pr[path(S, u) = 1]$ using the propagation traces that we have in the action log.

Data Model. We are given a social graph $G = (V, E)$, with nodes V corresponding to users and directed (unweighted) edges E corresponding to social ties between users, and an *action*

log, i.e., a relation $\mathbb{L}(User, Action, Time)$ where a tuple $(u, a, t) \in \mathbb{L}$ indicates that user u performed action a at time t. It contains such a tuple for every action performed by every user of the system. We will assume that the projection of $\mathbb{L}$ on the first column is contained in the set of nodes V of the social graph G. We let $\mathcal{A}$ denote the universe of actions, i.e., the projection of $\mathbb{L}$ on the second column. Moreover, we assume that a user performs an action at most once, and define the function $t(u, a)$ to return the time when user u performed action a (the value of $t(u, a)$ is undefined if u never performed a, and $t(u, a) < t(v, a)$ is false whenever either of $t(u, a), t(v, a)$ is undefined).

We say that a propagates from node u to v iff u and v are socially linked, and u performs a before v (we also say that u influences v on a). This defines a *propagation graph* of a as a directed graph $G(a) = (V(a), E(a))$, with $V(a) = \{v \in V \mid \exists t : (v, a, t) \in \mathbb{L}\}$ and $E(a) = \{(u, v) \in E \mid t(u, a) < t(v, a)\}$. Note that the propagation graph of an action a is the graph-representation of the propagation trace of a, and it is always a DAG: it is directed, each node can have zero or more parents, and cycles are impossible due to the time constraint. The action log $\mathbb{L}$ is thus a set of these DAGs representing propagation traces through the social graph. We denote by $N_{\text{in}}(u, a) = \{v \mid (v, u) \in E(a)\}$ the set of *potential influencers* of u for action a and $d_{\text{in}}(u, a) = |N_{\text{in}}(u, a)|$ to be the *in-degree* of u for action a. Finally, we call a user u an *initiator* of action a if $u \in V(a)$ and $d_{\text{in}}(u, a) = 0$, i.e., u performed action a but none of its neighbors performed it before u did. Table 5.3 summarizes the notation used.

The Sparsity Issue. In order to estimate $Pr[path(S, u) = 1]$ using available propagation traces, it is natural to interpret such quantity as the fraction of the actions initiated by S that propagated to u, given that S is the seed set. More precisely, we could estimate this probability as

$$\frac{|\{a \in \mathcal{A} | initiate(a, S) \,\&\, \exists t : (u, a, t) \in \mathbb{L}\}|}{|\{a \in \mathcal{A} | initiate(a, S)\}|}$$

where $initiate(a, S)$ is true iff S is precisely the set of initiators of action a. Unfortunately, this approach suffers from a *sparsity issue* which is intrinsic to the Influence Maximization problem [78]. If we need to be able to estimate $Pr[path(S, u) = 1]$ for any set S and node u, we will need an enormous number of propagation traces corresponding to various combinations, where each trace has as its initiator set precisely the required node set S. It is clearly impractical to find a real action log where this can be realized. To overcome this obstacle, we propose a different approach to estimating $Pr[path(S, u) = 1]$ by taking a "u-centric" perspective: we assign "credits" to the possible influencers of a node u whenever u performs an action. The model is formally described next.

Credit Distribution. When a user u performs an action a, we want to give *direct influence credit*, denoted by $\gamma_{v,u}(a)$, to all $v \in N_{\text{in}}(u, a)$, i.e., all neighbors of u that have performed the same action a before u. We constrain the sum of the direct credits given by a user to its neighbors to be no more than 1. We can have various ways of assigning direct credit: for ease of exposition, we assume for the moment to give equal credits to each neighbor v of u, i.e., $\gamma_{v,u}(a) = 1/d_{\text{in}}(u, a)$ for all $v \in N_{\text{in}}(u, a)$. Later we will see a more sophisticated method of assigning direct credit.

Intuitively, we also want to distribute influence credit transitively backwards in the propagation graph $G(a)$, such that not only u gives credit to the users $v \in N_{\text{in}}(u, a)$, but they in

$\mathcal{A}_u$	Number of actions performed by u.
$N_{\text{in}}(u,a)$	Neighbors of u which activated on action a before. i.e., u's potential influencers on action a.
$\gamma_{v,u}(a)$	Direct influence credit given to v for influencing u for action a.
$\Gamma_{v,u}(a)$	Total credit given to v for influencing u for action a.
$\kappa_{v,u}$	Total credit given to v for influencing u for all actions.
$\Gamma^W_{x,u}(a)$	Total credit given to x for influencing u for action a considering the paths that are completely contained in $W \subseteq V$.
$\tau_{v,u}$	Average time taken by actions to propagate from user u to user v.

Table 5.3: Notation adopted in the next sections in this chapter.

turn pass on the credit to their predecessors in $G(a)$ and so on. This suggests the following definition of *total credit* given to a user v for influencing u on action a, corresponding to multiple propagation paths:

$$\Gamma_{v,u}(a) = \sum_{w \in N_{\text{in}}(u,a)} \Gamma_{v,w}(a) \cdot \gamma_{w,u}(a) \tag{5.5}$$

where the base of the recursion is $\Gamma_{v,v}(a) = 1$. Sometimes, when the action is clear from the context, we can omit it and simply write $\gamma_{v,u}$ and $\Gamma_{v,u}$. From here on, as a running example, we consider the influence graph in Figure 5.1 as the propagation graph $G(a)$ with edges labeled with direct credits $\gamma_{v,u}(a) = 1/d_{in}(u,a)$. For instance,

$$\begin{aligned}\Gamma_{v,u} &= \Gamma_{v,v} \cdot \gamma_{v,u} + \Gamma_{v,t} \cdot \gamma_{t,u} + \Gamma_{v,w} \cdot \gamma_{w,u} + \Gamma_{v,z} \cdot \gamma_{z,u} \\ &= 1 \cdot 0.25 + 0.5 \cdot 0.25 + 1 \cdot 0.25 + 0.5 \cdot 0.25 = 0.75.\end{aligned}$$

We next define the total credit given to a set of nodes $S \subseteq V(a)$ for influencing user u on action a as follows:

$$\Gamma_{S,u}(a) = \begin{cases} 1 & \text{if } v \in S; \\ \sum_{w \in N_{\text{in}}(u,a)} \Gamma_{S,w}(a) \cdot \gamma_{w,u}(a) & \text{otherwise} \end{cases}$$

Consider again the propagation graph $G(a)$ in Figure 5.1. Let $S = \{v, z\}$. Then, $\Gamma_{S,u}$ is the fraction of flow reaching u that flows from either v or z:

$$\begin{aligned}\Gamma_{S,u} &= \Gamma_{S,w} \cdot \gamma_{w,u} + \Gamma_{S,v} \cdot \gamma_{v,u} + \Gamma_{S,t} \cdot \gamma_{t,u} + \Gamma_{S,z} \cdot \gamma_{z,u} \\ &= 1 \cdot 0.25 + 1 \cdot 0.25 + 0.5 \cdot 0.25 + 1 \cdot 0.25 = 0.875.\end{aligned}$$

Aggregating Over All Actions and All Nodes. The next question is how to aggregate the influence credit over the whole action log $\mathbb{L}$. Consider two nodes v and u: the total influence credit given to v by u for all actions in $\mathcal{A}$, is simply obtained by taking the total credit over all actions and normalizing it by the number of actions performed by u (denoted $\mathcal{A}_u$). This is justified by the fact that credits are assigned by u backward to its potential influencers. We define:

$$\kappa_{v,u} = \frac{1}{\mathcal{A}_u} \sum_{a \in \mathcal{A}} \Gamma_{v,u}(a) \tag{5.6}$$

Intuitively, it denotes the average credit given to v for influencing u, over all actions that u performs. Similarly, for the case of a set of nodes $S \subseteq V$, we can define the total influence credit for all the actions in $\mathcal{A}$ as:

$$\kappa_{S,u} = \frac{1}{\mathcal{A}_u} \sum_{a \in \mathcal{A}} \Gamma_{S,u}(a) \tag{5.7}$$

Note that $\kappa_{S,u}$ corresponds, in our approach, to $Pr[path(S,u) = 1]$ in Eq. 5.4. Finally, inspired by Eq. 5.4, we define the influence spread $\sigma_{cd}(S)$ as the total influence credit given to S from the whole social network:

$$\sigma_{cd}(S) = \sum_{u \in V} \kappa_{S,u} \tag{5.8}$$

In the spirit of Influence Maximization (Problem 1), this is the objective function that we want to maximize. In the next section we formally state the problem of maximizing influence under the CD model. We prove that the problem is NP-hard and that the function $\sigma_{cd}(.)$ is submodular, paving the way for an approximation algorithm.

Assigning Direct Credit. We now revisit the problem of defining the direct credit $\gamma_{v,u}(a)$ given by a node u to a neighbor v for action a. In our previous work [56], we observed that influence decays over time in an exponential fashion and that some users are more influenceable than others. Motivated by these ideas, we propose to assign direct credit as:

$$\gamma_{v,u}(a) = \frac{infl(u)}{N_{\text{in}}(u,a)} \cdot \exp\left(-\frac{t(u,a) - t(v,a)}{\tau_{v,u}}\right) \tag{5.9}$$

Here, $\tau_{v,u}$ is the average time taken for actions to propagate from user v to user u. The exponential term in the equation achieves the desired effect that influence decays over time. Moreover, $infl(u)$ denotes the user influenceability, that is, how prone the user u is to influence by the social context [56]. Precisely, $infl(u)$ is defined as the fraction of actions that u performs under the influence of at least one of its neighbors, say v, i.e., u performs the action, say a, such that $t(u,a) - t(v,a) \leq \tau_{v,u}$; this is normalized by $N_{\text{in}}(u,a)$ to ensure that the sum of direct credits assigned to neighbors of u for action a is at most 1. Note that both $infl(u)$ and $\tau_{\cdot,u}$ are learnt from (the training subset of) $\mathbb{L}$.

Discussion. It should be pointed out that unlike classical models such as IC and LT, the credit distribution model is *not* a propagation model. Instead, it is a model that, based on available propagation data, learns the total influence credit accorded to a given set S by any node u and uses this to predict the influence spread of S. It is not susceptible to the sparsity issue discussed above, and it obviates the need to perform expensive MC simulations for the purpose of estimating influence spread.

5.4 Influence Maximization

We next formally define the problem studied in this chapter.

Problem 2 (Influence Maximization- CD model)**.** *Given a directed social graph $G = (V,E)$, an action log $\mathbb{L}$, and an integer $k \leq |V|$, find a set $S \subseteq V$, $|S| = k$, that maximizes $\sigma_{cd}(S)$.*

Theorem 4. Influence Maximization *under the credit distribution model is NP-hard.*

Proof. We prove the hardness by reducing the well-known **NP**-complete problem Vertex Cover [48] to our problem. Given an instance $\mathcal{I}$ of Vertex Cover, consisting of an undirected graph $G = (V, E)$ and a number k, create an instance $\mathcal{J}$ of the Influence Maximization problem under CD as follows. The directed social graph $G' = (V, E')$ associated with $\mathcal{J}$ has the same node set as G. E' two directed edges (u, v) and (v, u) in place of every undirected edge $(u, v) \in E$. We express the action log $\mathbb{L}$ associated with $\mathcal{J}$ in terms of propagation graphs, for convenience. For each edge $(v, u) \in E$, create two propagation graphs, corresponding to two actions a_1 and a_2 in $\mathbb{L}$, consisting of only two nodes v and u. In the propagation graph $G(a_1)$, create an edge from v to u indicating that the corresponding action is being propagated from v to u. In the propagation graph $G(a_2)$, create an edge from u to v. Assign direct credits $\gamma_{v,u}(a_1) = \gamma_{u,v}(a_2) = \alpha$ where $\alpha \in (0, 1]$. For instance, if we assign direct credits simply as $\gamma_{v,u}(a) = 1/d_{\text{in}}(u, a)$, then $\alpha = 1$. Similarly, if we assign direct credits as in equation 5.9, then $\alpha = 1/e$. The reduction clearly takes polynomial time. We next prove that a set $S \subseteq V$, with $|S| \leq k$, is a vertex cover of G if and only if its influence spread $\sigma_{cd}(S)$ in the instance $\mathcal{J}$ is at least $k + \alpha \cdot (|V| - k)/2$.

Only if: Suppose S is a vertex cover of G in the instance $\mathcal{I}$. Consider any arbitrary node u. If $u \in S$, then $\kappa_{S,u} = 1$ by definition. On the other hand, if $u \notin S$, then $\kappa_{S,u} = \sum_a \Gamma_{S,u}(a)/(2 \cdot deg(u))$. Since u is not in the vertex cover, all its neighbors must be in S and thus, for exactly half of the actions a that u performs, $\Gamma_{S,u}(a) = \alpha$. These are the actions that u performs after its neighbor. Hence, $\sum_a \Gamma_{S,u}(a) = \alpha \cdot deg(u)$ and $\kappa_{S,u} = \alpha/2$, where $deg(u)$ is u's degree in G. This implies $\sigma_{cd}(S) = \sum_{u \in V} \kappa_{S,u} = k + \alpha \cdot (|V| - k)/2$.

If: Let S be any seed set whose spread is at least $k + \alpha \cdot (|V| - k)/2$ in instance $\mathcal{J}$. Let $N(u)$ be the set of neighbors of u in G, that is, $N(u) = \{v \in V | (v, u) \in E\}$. Consider an arbitrary node $u \notin S$. For each node v in $N(u) \cap S$, v has a credit of α over u, for the unique action whose propagation graph is the edge (v, u), and a null credit for all the other actions. Therefore $\kappa_{v,u} = \alpha/(2 \cdot deg(u))$. Aggregating over the whole seed set S we have that $\kappa_{S,u} = \alpha \cdot |N(u) \cap S|/(2 \cdot deg(u))$. Hence, $\sigma_{cd}(S) = \sum_{u \in V} \kappa_{S,u} = k + \sum_{u \in V \setminus S} \alpha \cdot |N(u) \cap S|/(2 \cdot (deg(u))$.

From our assumption, $\sigma_{cd}(S) \geq k + \alpha \cdot (|V| - k)/2$, it follows that $\sum_{u \in V \setminus S} |N(u) \cap S|/deg(u) \geq |V| - k$. As $|N(u) \cap S| \leq deg(u)$, this is possible only when $\forall u \in V \setminus S : |N(u) \cap S| = deg(u)$, implying that all neighbors of u must be in S and therefore, S is a vertex cover in instance $\mathcal{I}$. □

Since the problem is NP-hard, we are interested in developing an approximation algorithm. We prove that the influence spread function is submodular under the CD model, paving the way for efficient approximation.

Theorem 5. $\sigma_{cd}(S)$ *is monotone and submodular.*

Proof. It suffices to show that $\Gamma_{S,u}(a)$ is monotone and submodular as a positive linear combination of monotone, submodular functions is also monotone and submodular [103]. Clearly it is monotone. We prove submodularity by induction on path lengths. Note that the propagation graph for a given action is acyclic and hence the maximum path length is $|V| - 1$. Let

$\Gamma_{S,u}(a,\ell)$ denote the total credit obtained by the set S for influencing u, restricting attention to paths of length $\leq \ell$. Thus, $\Gamma_{S,u}(a) = \Gamma_{S,u}(a, |V|-1)$.

Let S and T be two node sets such that $S \subseteq T$ and let $x \notin T$. Recall that the function Γ is submodular iff $\Gamma_{S+x,u}(a) - \Gamma_{S,u}(a) \geq \Gamma_{T+x,u}(a) - \Gamma_{T,u}(a)$. We call the left hand side of the inequality the marginal gain of x with respect to S (implicitly understood to be on u) and similarly for the right hand side.

Base Case: In the base case, $\ell = 0$. Depending on u, the base case can be split into various sub-cases: (a) If $u \in S$, then the marginal gain of x with respect to both S and T is 0; (b) If $u \in T, u \notin S$, then while x's marginal gain with respect to T is 0, its marginal gain with respect to S is no less than 0 as the function $\Gamma(\cdot)$ is monotone; (c) If $u = x$, then the marginal gain of x with respect to both S and T is exactly 1; (d) If $u \neq x$ and $u \notin T$, then the total credits on u from S, $S+x$, T and $T+x$ are 0. This proves the base case.

Induction Step: Assume that the function Γ is submodular when restricted to path lengths $\leq \ell$, that is, $\forall w \in V$:

$$\Gamma_{S+x,w}(a,\ell) - \Gamma_{S,w}(a,\ell) \geq \Gamma_{T+x,w}(a,\ell) - \Gamma_{T,w}(a,\ell) \tag{5.10}$$

We will prove that the function remains submodular for paths of length $\ell+1$ for any node $u \in V$. Consider the marginal gain of x with respect to S on u when restricted to paths of length $\leq \ell+1$, that is, consider $\Gamma_{S+x,u}(a,\ell+1) - \Gamma_{S,u}(a,\ell+1)$. By definition, this is equal to $\sum_{w \in N_{\text{in}}(u,a)} \Gamma_{S+x,w}(a,\ell) \cdot \gamma_{w,u}(a) - \sum_{w \in N_{\text{in}}(u,a)} \Gamma_{S,w}(a,\ell) \cdot \gamma_{w,u}(a)$. Taking the common factor $\gamma_{w,u}(a)$ out and applying induction hypothesis (Eq. 5.10), this is

$$\begin{aligned} &\geq \sum_{w \in N_{\text{in}}(u,a)} (\Gamma_{T+x,w}(a,\ell) - \Gamma_{T,w}(a,\ell)) \cdot \gamma_{w,u}(a) \\ &= \Gamma_{T+x,u}(a,\ell+1) - \Gamma_{T,u}(a,\ell+1) \end{aligned}$$

This was to be shown. □

In the remaining sub-sections, we develop an efficient approximation algorithm to solve the Influence Maximization problem under the CD model.

5.4.1 Overview of our method

In the previous section, we show that while Influence Maximization under the CD model is NP-hard, the influence spread function is monotone and submodular. Consequently, the greedy algorithm (Algorithm 1) provides a $(1-1/e)$-approximation to the optimum solution [103]. However, the greedy algorithm by itself does not guarantee efficiency as it requires to compute the marginal gain of a candidate seed node with respect to the current seed set, i.e., $\sigma_{cd}(S+w) - \sigma_{cd}(S)$ (line 3 of Algorithm 1). For the IC and LT models, this is done by expensive MC simulations. For the CD model, the marginal gain can be directly computed from the action log $\mathbb{L}$. A naive way to do this would be to scan $\mathbb{L}$ in each iteration. But this approach would be very inefficient. Hence, we focus our attention on computing the marginal gain efficiently by carefully exploiting some properties of our model.

From here on, with a superscript $W \subseteq V$ on the function $\Gamma(\cdot)$, we denote the function to be evaluated on the sub-graph induced by nodes in W. For example, $\Gamma^{W}_{x,u}(a)$ is the total credit given to node x for influencing node u to perform action a considering the paths that are contained completely in the sub-graph induced by $V(a) \cap W$. That is, the sub-graph of the propagation graph for action a, induced by the nodes $W \cap V(a)$. When the superscript is not present the graph considered is the whole propagation graph for action a, i.e., $\Gamma_{x,u}(a) = \Gamma^{V(a)}_{x,u}(a)$. It should be noted that the direct credit $\gamma_{x,u}$ is always assigned considering the whole propagation graph. The following result is key to the efficiency of our algorithm.

Theorem 6.

$$\sigma_{cd}(S + x) - \sigma_{cd}(S) = \sum_{a \in \mathcal{A}} \left((1 - \Gamma_{S,x}(a)) \cdot \sum_{u \in V} \frac{1}{\mathcal{A}_u} \cdot \Gamma^{V-S}_{x,u}(a) \right)$$

Intuitively, the theorem says that the marginal gain of a node x equals the sum of *normalized* marginal gain of x on all actions. We give more insights into this equation in the proof. The theorem provides us an efficient method to compute the marginal gain, given values of $\Gamma_{S,x}(a)$ and $\Gamma^{V-S}_{x,u}(a)$: this is the key idea behind the efficiency of our algorithm, which can be abstractly summarized as follows:

1. Initially, scan the action log $\mathbb{L}$ and compute $\Gamma_{v,u}(a)$ for all combinations of v, u and a (Algorithm 12). Note that at the beginning, $S = \emptyset$ and hence $\Gamma_{S,x}(a) = 0$ for all combinations of x and a.

2. In each iteration of the greedy method, a node that provides the maximum marginal gain is added to the seed set. For this step we adopt the CELF [91] optimization idea (Algorithm 13).

3. To compute the marginal gain of a node x efficiently, we use Theorem 6. It requires values of $\Gamma^{V-S}_{v,u}(a)$ and $\Gamma_{S,x}(a)$ (Algorithm 14).

4. Once a node is added to the seed set, $\Gamma^{V-S}_{v,u}(a)$ and $\Gamma_{S,x}(a)$ are updated using Lemmas 2 and 3 (Alg. 15).

5.4.2 Proof of Theorem 6

The proof of Theorem 6 is non-trivial and we need to prove a few auxiliary claims first. In the process, we also derive equations to update the total credit (step 4 in the outline of our algorithm above).

Lemma 1. $\Gamma_{S,u}(a) = \sum_{v \in S} \Gamma^{V-S+v}_{v,u}(a)$

We first explain the claim by means of an example by taking the influence graph in Figure 5.1 as a propagation graph $G(a)$ with direct credit $\gamma_{v,u}(a) = 1/d_{in}(u, a)$. Let $S = \{v, z\}$, according to Lemma 1 (dropping the argument a), $\Gamma_{S,u} = \Gamma^{V-z}_{v,u} + \Gamma^{V-v}_{z,u} = (0.25 + 0.25 + 0.5 \cdot 0.25) + 0.25 = 0.875$. Note that the credit given to v via the path $v \to t \to z \to u$ is ignored. Next, we formally prove the claim.

Proof of Lemma 1: By induction on path lengths. Recall that $\Gamma_{S,u}(a,l)$ denotes the total credit given to set S for influencing node u over paths of length no more than l.

Base Case: $l = 0$ implies only u can get credit for influencing itself. Therefore if $u \notin S$, both sides of the equality become 0. When $u \in S$, the total credit given to S for influencing u (left hand side) is 1 by definition, while in the right hand side all terms in the summation are 0 except the case $v = u$, that is $\Gamma^{V-S+u}_{u,u}(a,0) = 1$.

Induction Step: Assume that the lemma is true for path lengths no more than l. We prove it for path length up to $l+1$. We start with definition of $\Gamma_{S,u}(a,l+1)$ (Eq. 5.5). $\Gamma_{S,u}(a,l+1) = \sum_{w \in N_{\text{in}}(u,a)} \Gamma_{S,w}(a,l) \cdot \gamma_{w,u}(a)$. Applying induction hypothesis, the right hand side becomes:

$$= \sum_{w \in N_{\text{in}}(u,a)} \left(\sum_{v \in S} \Gamma^{V-S+v}_{v,w}(a,l) \right) \cdot \gamma_{w,u}(a) =$$

$$\sum_{v \in S} \left(\sum_{w \in N_{\text{in}}(u,a)} \Gamma^{V-S+v}_{v,w}(a,l) \cdot \gamma_{w,u}(a) \right) = \sum_{v \in S} \Gamma^{V-S+v}_{v,u}(a,l+1)$$

This concludes the proof. □

Next, we show how the total credit can be updated incrementally, when the induced sub-graph under consideration changes. Consider the sub-graph induced by the nodes $W = V - S$ where S is the current seed set. Let $\Gamma^{W}_{v,u}(a)$ be the total credit given to node v for influencing u in this sub-graph. Suppose node x is added to the seed set, then we are interested in computing $\Gamma^{W-x}_{v,u}(a)$. This is clearly the total credit given to v minus the total credit given to v via paths that go through x. More precisely, we have:

Lemma 2. $\Gamma^{W-x}_{v,u}(a) = \Gamma^{W}_{v,u}(a) - \Gamma^{W}_{v,x}(a) \cdot \Gamma^{W}_{x,u}(a)$.

As an example, consider again the propagation graph in Figure 5.1 with $S = \{t, z\}$. The total credit given to v for influencing u on the subgraph induced by nodes in $V - \{t, z\}$ is $1 \cdot 0.25 + 0.25 = 0.5$. Suppose w is added to the seed set S, then $\Gamma^{V-S-w}_{v,u} = 0.5 - 1 \cdot 0.25 = 0.25$.

The next lemma shows how to update incrementally the total credit of influence given to a set S by a node u, after x is added to the set. This is needed in step 4 of our method as sketched in previous section.

Lemma 3. $\Gamma_{S+x,u}(a) = \Gamma_{S,u}(a) + \Gamma^{V-S}_{x,u} \cdot (1 - \Gamma_{S,x}(a))$

Proof: Since we refer to a single action, we drop the argument a and assume it implicitly. We use Lemma 1 to expand $\Gamma_{S+x,u}$ and $\Gamma_{S,u}$:

$$\Gamma_{S+x,u} - \Gamma_{S,u} = \sum_{v \in S+x} \Gamma^{V-S-x+v}_{v,u} - \sum_{v \in S} \Gamma^{V-S+v}_{v,u}$$

$$= \Gamma^{V-S}_{x,u} - \sum_{v \in S} (\Gamma^{V-S+v}_{v,u} - \Gamma^{V-S-x+v}_{v,u})$$

Applying Lemma 2 to the terms inside the summation (with $W = V - S + v$), the right hand side becomes

$$= \Gamma^{V-S}_{x,u} - \sum_{v \in S} (\Gamma^{V-S+v}_{v,x} \cdot \Gamma^{V-S+v}_{x,u})$$

The terms inside the summation denote the total credit given to v for influencing u considering the paths that go through x in the sub-graph induced by the nodes $V - S + v$. Since the graph is acyclic, if $\Gamma_{v,x}^{V-S+v}$ is non-zero, then any path from x to u cannot pass through v. Hence, $\Gamma_{v,x}^{V-S+v} \cdot \Gamma_{x,u}^{V-S+v} = \Gamma_{v,x}^{V-S+v} \cdot \Gamma_{x,u}^{V-S}$. Note that this equality holds even when $\Gamma_{v,x}^{V-S+v} = 0$ as both sides would be 0 in that case. Thus, $\Gamma_{S+x,u} - \Gamma_{S,u} = \Gamma_{x,u}^{V-S} - \sum_{v \in S}(\Gamma_{v,x}^{V-S+v} \cdot \Gamma_{x,u}^{V-S})$.

The term $\Gamma_{x,u}^{V-S}$ can be taken out of the summation and applying Lemma 1 gives $\sum_{v \in S} \Gamma_{v,x}^{V-S+v} = \Gamma_{S,x}$, from which the lemma follows. □

Finally, we are ready to prove Theorem 6.

Proof of Theorem 6: By definition,

$$\sigma_{cd}(S + x) - \sigma_{cd}(S) = \sum_{u \in V} \frac{1}{\mathcal{A}_u} \sum_{a \in \mathcal{A}} (\Gamma_{S+x,u}(a) - \Gamma_{S,u}(a))$$

Applying lemma 3, the right hand side becomes

$$= \sum_{u \in V} \frac{1}{\mathcal{A}_u} \sum_{a \in \mathcal{A}} (\Gamma_{x,u}^{V-S}(a) \cdot (1 - \Gamma_{S,x}(a)))$$

Rearranging the terms, we get $\sigma_{cd}(S + x) - \sigma_{cd}(S) = \sum_{a \in \mathcal{A}} \Big((1 - \Gamma_{S,x}(a)) \cdot \sum_{u \in V} \frac{1}{\mathcal{A}_u} \cdot \Gamma_{x,u}^{V-S}(a) \Big)$, which was to be shown. □

5.4.3 Algorithms

In this section, we present our algorithm which builds on the properties developed in previous sections and whose outline was given in Section 5.4.1. Initially, we scan the action log $\mathbb{L}$ and then, we use the greedy algorithm with CELF optimization to select the seed set. While scanning the action log, we maintain all the information needed to select k seeds later. In particular, we compute total credit given to each node v for influencing any other node u for all actions a and record it into the data structure UC (User Credits). Each entry $UC[v][u][a]$ corresponds to $\Gamma_{v,u}^{V-S}(a)$, that is, total credit given to v for activating u on the graph induced by $V - S$ where S is the current seed set. We also maintain another data structure SC (Set Credits) where each entry $SC[x][a]$ refers to the total credit given to the current seed set S by a node x for an action a, that is, $\Gamma_{S,x}(a)$. Since S is empty in the beginning, SC is not used in the first iteration.

Algorithm 12 describes the first step of our method that scans $\mathbb{L}$. $\mathbb{L}$ is maintained sorted, first by action and then by time. It processes one action at a time and in chronological order. We use *current_table* to maintain the list of users who have performed the current action and have been seen so far, and $\mathcal{A}_u$ to denote the number of actions performed by user u in $\mathbb{L}$, and $Parents(u)$ for the list of parents of each user u with respect to the current action a, that is, $N_{\text{in}}(u, a)$. For each action a and for each user u that performs it, we scan *current_table* to find its neighbors that already performed a and add them to the list of parents of u. Then for each parent v of u, we compute the direct credit $\gamma_{v,u}(a)$ appropriately (line 9). For the ease of exposition, here we assume the simple definition $\gamma_{v,u}(a) = 1/N_{\text{in}}(u, a)$, that can be implemented as $\gamma = 1/|Parents(u)|$. If we want to use the more complex definition of

Algorithm 12 *Scan*

Input: $G, \mathbb{L}, \lambda$
Output: UC

1: $UC \leftarrow \emptyset$
2: **for** each action a in $\mathbb{L}$ **do**
3: $current\_table \leftarrow \emptyset$
4: **for** each tuple $\langle u, a, t_u \rangle$ in chronological order **do**
5: $Parents(u) \leftarrow \emptyset$; $\mathcal{A}_u \leftarrow \mathcal{A}_u + 1$; $UC[*][u][a] \leftarrow 0$
6: **while** $\exists v : (v, u) \in G, v \in current\_table$ **do**
7: $Parents(u) \leftarrow Parents(u) \cup \{v\}$
8: **end while**
9: **for** each $v \in Parents(u)$ **do**
10: compute $\gamma_{v,u}$
11: **if** $\gamma_{v,u} \geq \lambda$ **then**
12: $UC[v][u][a] \leftarrow UC[v][u][a] + \gamma_{v,u}$
13: **for** each w such that $UC[w][v][a] \cdot \gamma_{v,u} \geq \lambda$ **do**
14: $UC[w][u][a] \leftarrow UC[w][u][a] + \gamma_{v,u} UC[w][v][a]$
15: **end for**
16: **end if**
17: **end for**
18: **end for**
19: $current\_table \leftarrow current\_table \cup \{u\}$
20: **end for**

direct credit given in Eq. (5.9), we need to learn the parameters $\tau_{v,u}$ for all edges and $infl(u)$ for all nodes in advance and pass them on to Algorithm 12 as input, similarly to what the standard method does for influence probabilities. Although it is straightforward to learn these parameters by means of a preliminary scan of $\mathbb{L}$, we refer the reader to [56] for an efficient way to learn them. The total credit given to various nodes for influencing u is then computed using equation 5.5 (lines 10-13). For the sake of reducing memory requirements, we use a truncation threshold λ and discard credits that are below the threshold. In the experiments we will assess the effect of this truncation.

Algorithm 13 *Greedy* with CELF

Input: UC, k
Output: seed set S

1: $SC \leftarrow \emptyset$; $S \leftarrow \emptyset$; $Q \leftarrow \emptyset$
2: **for** each $u \in V$ **do**
3: $x.mg \leftarrow computeMG(x)$; $x.it \leftarrow 0$; add x to Q
4: **end for**
5: **while** $|S| < k$ **do**
6: $x \leftarrow pop(Q)$
7: **if** $x.it = |S|$ **then** $S \leftarrow S \cup \{x\}$; $update(x, UC, SC)$
8: **else**
9: $x.mg \leftarrow computeMG(x, UC, SC)$;
10: $x.it \leftarrow |S|$; Reinsert x into Q and heapify
11: **end while**

Once the first phase is completed, we use the standard greedy algorithm with the CELF optimization [91] to select the seeds (Algorithm 13). The algorithm maintains a queue Q where an entry for a user x is stored in the form $\langle x, mg, it\rangle$, where mg represents the marginal gain of user x with respect to seed set in iteration it. Q is always kept sorted in decreasing order of mg. Initially, the influence spread of each node is computed and Q is built (lines 2-3). In each iteration, the top element x of Q is analyzed. If x is analyzed before in the current iteration (that is, $x.it = |S|$), then it is picked as the next seed node and the subroutine *update* is called (line 6). On the other hand, if $x.it < |S|$, we recompute the marginal gain of x with respect to S by calling the subroutine *computeMG* (line 8). Then $x.it$ is set appropriately and x is re-inserted into Q (line 9).

Algorithm 14 *computeMG*

Input: x, UC, SC
Output: mg
1: $mg = 0$
2: **for** each action a such that $\exists u : UC[x][u][a] > 0$ **do**
3: $\quad mg_a \leftarrow 1/A_x$
4: $\quad$ **for** each user u such that $UC[x][u][a] > 0$ **do**
5: $\quad\quad mg_a \leftarrow mg_a + UC[x][u][a]/\mathcal{A}_u$
6: $\quad$ **end for**
7: $\quad mg \leftarrow mg + mg_a(1 - SC[x][a])$
8: **end for**

Algorithm 15 *update*

Input: x, UC, SC
1: **for** each action a such that $\exists u : UC[x][u][a] > 0$ **do**
2: $\quad$ **for** each u such that $UC[x][u][a] > 0$ **do**
3: $\quad\quad$ **for** each v such that $UC[v][x][a] > 0$ **do**
4: $\quad\quad\quad UC[v][u][a] \leftarrow UC[v][u][a] - UC[v][x][a] \cdot UC[x][u][a]$
5: $\quad\quad$ **end for**
6: $\quad\quad SC[u][a] \leftarrow SC[u][a] + UC[x][u][a] \cdot (1 - SC[x][a])$
7: $\quad$ **end for**
8: **end for**

Algorithm 14 computes the marginal gain of a node x with respect to the current seed set S. It leverages Theorem 6 to do this efficiently. For an action a, lines 3-5 compute the term $\sum_{u \in V} \frac{1}{\mathcal{A}_u} \cdot \Gamma^{V-S}_{x,u}(a)$ and line 6 multiplies it by the term $(1 - \Gamma_{S,x}(a))$. Finally, whenever a user x is added to the seed set, the subroutine *update* is invoked and Algorithm 15 updates both UC and SC, using Lemmas 2 and 3.

Memory requirements: Our algorithm requires to maintain the data structure UC whose size is potentially of the order of $\sum_{a \in \mathcal{A}} |V(a)|^2$. In reality, total credit decreases sharply with the length of paths. Thus by ignoring values that are below a given truncation threshold λ, the memory usage by our algorithm can be kept reasonable. We study the effect of λ in the experiments in the next section.

5.5 Experimental Evaluation

The goals of our experiments are manifold. At a high level, we want to evaluate the different models and the optimization algorithms based on them with respect to accuracy of spread prediction, quality of seed selection, running time, and scalability. We perform additional experiments on the CD model and on the influence maximization algorithm based on it, to explore the impact of training data size on the quality of the solution and the impact of truncation threshold on the quality, running time, and memory usage. The source of the code used in our experiments is available at `http://people.cs.ubc.ca/~goyal/code-release.php`.

We experiment on the same two real world datasets of Section 5.2 (Table 5.1). While we use the "large" versions of the datasets only to study the scalability of our method, "small" versions of the datasets are used to compare our algorithm with other methods (that do not scale to the large versions).

Methods Compared. Since methods based on arbitrarily assigning edge probabilities are dominated by those that learn them from the past propagation traces (see Section 5.2), in our evaluation, we focus only on the following models.

IC model with edge probabilities learnt from the training set by means of the EM method [114]. In all the experiments we run 10k MC simulations.

LT model with 10k MC simulations. We take ideas from [78] and [56] and learn weights as $p_{v,u} = A_{v2u}/N$ where A_{v2u} is the number of actions propagated from v to u in the training set and N is the normalization factor to ensure the sum of incoming weights on each node is 1.

CD model with direct credit assigned as described in Equation (5.9). Unless otherwise mentioned, the truncation threshold λ is set to 0.001 (see section 5.4.3). Later we also study effect of different truncation thresholds.

Accuracy of Spread Prediction. In Section 5.2 ("Experiment 2"), we evaluated methods using IC and LT models where edge probabilities are arbitrarily assigned and methods that learn them from available data, with respect to the accuracy of spread prediction. We conduct a similar experiment to compare the IC, LT, and CD models. Fig. 5.3 shows the RMSE (computed exactly in the same way as in Section 5.2) in the spread predicted by the IC, LT, and CD models, as a function of actual spread for both datasets. An interesting observation from the figure is that while IC beats LT by a large margin on Flixster_Small, it loses to LT on Flickr_Small by a considerable margin. On the other hand, the CD model performs very well on both the datasets.

In order to have a better understanding of the results, we conduct a detailed analysis. Fig. 5.4 depicts the proportion of propagation traces captured within a given absolute error, which is the absolute difference between the estimated spread and actual spread. More precisely, for a given method, a point (x, y) on its plot says that the fraction of propagation traces (in the test set) on which the (absolute) prediction error of that method is $\leq x$, is y. For instance, on Flixster_Small, for absolute error ≤ 30, CD model captures 67% of propagations (that is, 3391 out of 5128 propagations). On the other hand, the percentages of propagations captured within the same error by IC and LT model are 46% and 26% respec-

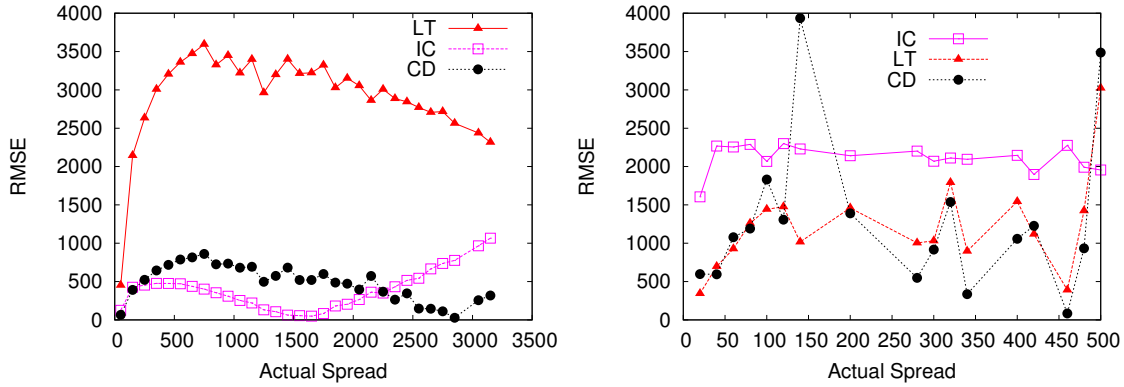


Figure 5.3: RMSE vs Propagation Size on Flixster_Small (left) and Flickr_Small (right).

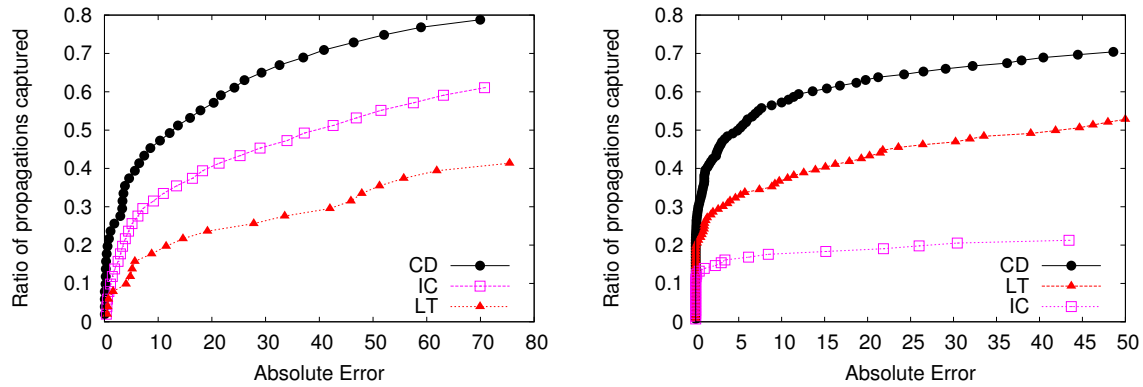


Figure 5.4: Number of propagations captured against Absolute Error on Flixster_Small (left) and Flickr_Small (right).

tively. Once again, it can be seen that while IC performs better than LT on Flixster_Small, it's the other way round on Flickr_Small. This plot shows conclusively that within any given error tolerance, CD is able to capture a much higher fraction of propagation traces than IC and LT, on both data sets, confirming that CD model is more accurate when it comes to predicting the influence spread of a given seed set.

Seed Set Intersection. Having established that CD is much more accurate in predicting actual spread, we next examine the question, how close to each other are the (near) optimal seed sets for the influence maximization problem, obtained by running the greedy algorithm under different models. Fig. 5.5 shows that the intersection of seed sets obtained from IC model with the seed sets obtained from LT and CD models is empty. On the other hand, there is a significant (∼50%) overlap between CD and LT models. We note since the greedy algorithm with MC simulations runs too slow on Flickr_Small (more on this later), in Fig. 5.5 we use PMIA [35] (for IC model) and LDAG [37] (for LT model) heuristics to obtain the seed set (only for Flickr_Small) in order to finish this experiment within a reasonable

	IC	LT	CD
IC	50	0	0
LT		50	26
CD			50

	CD	LT	IC
IC	0	0	50
LT	28	50	
CD	50		

Figure 5.5: Size of seed set intersection for $k = 50$ on FLIXSTER_SMALL (left) and FLICKR_SMALL (right).

time.[37] This shows that the seed sets obtained from IC and LT models are very different from CD model. The difference is much more pronounced in the case of IC model. These findings are significant since together with the results of the previous experiment, they offer some evidence that the seeds chosen by IC and LT models run the risk of being poor with respect to the actual spread they achieve. We strengthen this evidence by conducting the next experiment.

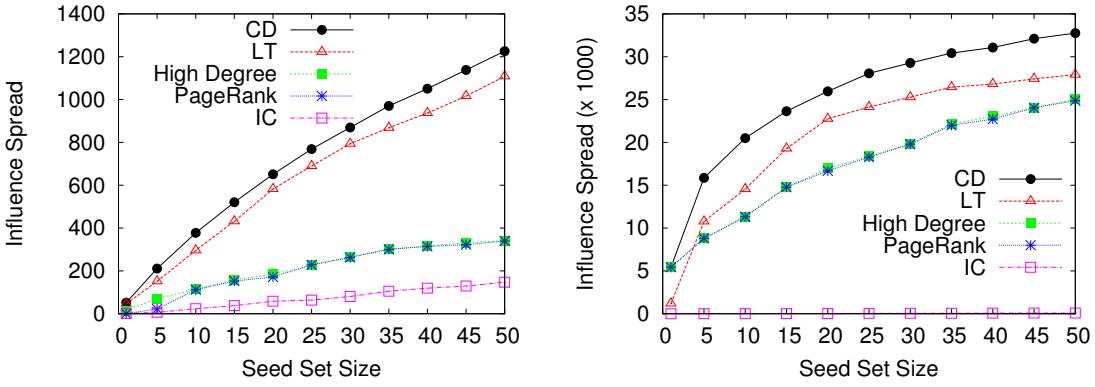


Figure 5.6: Influence spread achieved under CD model by seed sets obtained by various models on Flixster_Small (left) and Flickr_Small (right).

Spread Achieved. In this experiment, we compare the influence spread achieved by the seed sets obtained from the three methods. For the sake of completeness, we also include the heuristics High Degree and PageRank which select as seeds the top-k nodes with respect to degree and PageRank score respectively, as done in [35, 78].

One issue we face is that due to the sparsity issue, we cannot determine the actual spread of an arbitrary seed set from the available data. The next best thing we can do is pick a model that enjoys the least error in spread prediction and treat the spread predicted by it as actual spread. In this way, for any given seed set, we can use that model to tell us (its best estimate of) the actual spread. Given that CD model is found to be closest to reality in predicting the spread of a seed set (see Fig. 5.3 and 5.4), we use the spread predicted by it as the actual spread. The results of this experiment, depicted in Fig. 5.6, confirm that on both data sets the spread achieved by the seed sets found by methods using the IC and LT

[37] Chen et al.[35, 37] have shown the spread obtained from PMIA and LDAG are very close to those obtained via MC simulations for IC and LT.

models falls far short of the actual spread, which is best approximated using the CD model. A surprising observation is that IC model performs poorly, even worse than heuristics like High Degree and PageRank. We looked in the data and found that the seeds picked by IC model are nodes which perform a very small number of actions, often just one action, and should not be considered as high influential nodes. We investigate the reasons below.

For instance, on Flixster_Small, the first seed picked by the IC model is the user with Id 168766. While its influence spread under IC model is 499.6, it is only 1.08 under CD model. In the data, the user 168766 performs only one action and this action propagates to 20 of its neighbors. As a result, the EM method [114] ends up assigning probability 1.0 to the edges from 168766 to all its 20 neighbors, making it a high influence node, so much that it is picked as the first seed. Obviously, in reality, 168766 cannot be considered as a highly influential node since its influence is not statistically significant. In an analogy with Association Rules, the influence of user 168766 can be seen as a maximum *confidence* rule, but which occurs only once (absolute *support* = 1).

A deeper analysis tells us that most of the seeds picked by the IC model are of this kind: not very active nodes that, in the few cases they perform an action, do have some followers. We checked and found that the average number of actions performed by these seeds is 30.3 (against the global average of 167). On the other hand, the average number of actions performed by seed set obtained from our CD model is 1108.7. We found a similar behavior in Flickr_Small.

Running Time. In this experiment, we first show results on the small versions of the data sets, for all three models, as a function of number of seed nodes selected. All the experiments are run on an Intel(R) Xeon(R) CPU X5570 @ 2.93GHz machine with 64GB RAM running Opensuse 11.3. The algorithms are implemented in C++.

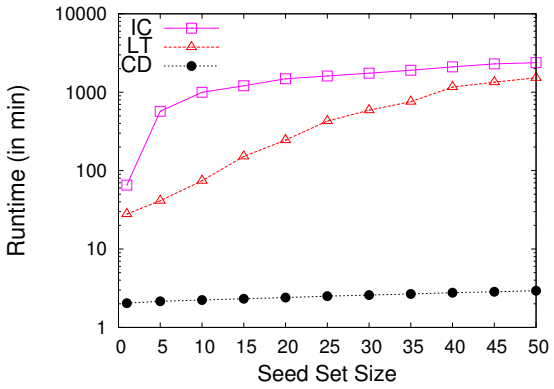


Figure 5.7: Running Time Comparison.

Fig. 5.7 reports the time taken (in minutes, on log scale) by the various models. It can be seen that our method is several orders of magnitude faster. For instance, to select 50 seeds on Flixster_Small, while the greedy algorithm (with Celf optimization) takes 40 and 25 hours under IC and LT model respectively, our algorithm takes only 3 minutes.

We do not show a similar plot for Flickr_Small as the experiment takes too long to complete (for IC and LT models). At the time of writing the corresponding paper for this chapter [57], while the experiment for IC model ran for 27 days without even selecting a single seed, the experiment for LT model took the same time to pick only 17 seeds. On the other hand, our algorithm takes only 6 minutes to pick 50 seeds.

Scalability. Next, we show the scalability of our algorithm with respect to the size of the

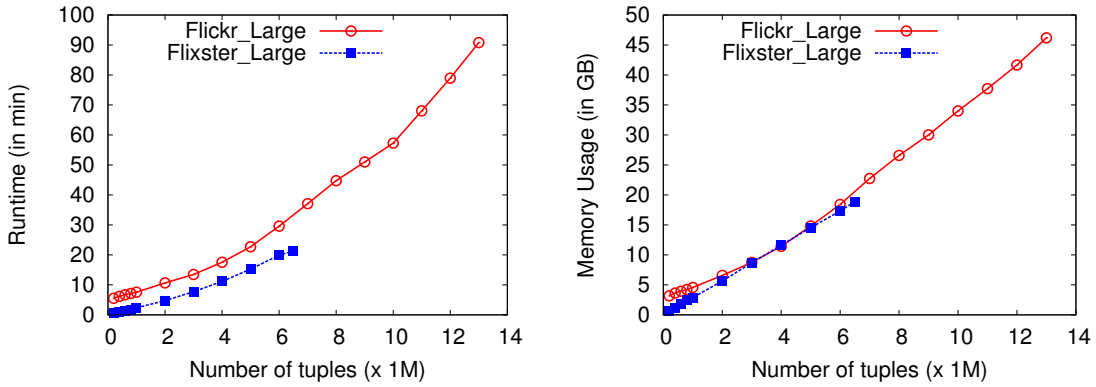


Figure 5.8: Runtime (left) and memory usage (right) against number of tuples.

action log, in number of tuples. For this purpose, we created the training data set by randomly choosing propagation traces from the complete action log and selecting all the corresponding action log tuples. In Fig. 5.8 and 5.9, the x-axis corresponds to the number of tuples in the training set.

Fig. 5.8 (left) shows the time taken by our algorithm to select 50 seeds against the number of tuples used. It should be noted that most of the time taken by our algorithm is consumed in scanning the action log. For example, it takes 15 minutes to select the seed set when 5M tuples are used on Flixster_Large, out of which, 11.6 minutes are spent on scanning the action log and only 3.4 minutes are incurred in selecting the seed set.

Fig. 5.8 (right) presents the memory usage with respect to the number of action log tuples used to select the seed set of size 50. Our algorithm's memory usage is proportional to the number of training tuples used: on Flixster_Large using 6.5M tuples, it requires approximately 16GB, while on 13M tuples on Flickr_Large, it requires approximately 46GB. This raises the question *how much training data is needed to select a good seed set*, which we study next.

Effect of Training Data Size. Fig. 5.9 shows the convergence of the output of our algorithm with respect to number of tuples used to select the seeds. Both plots have a double y-axis (left and right). On the left side, we have the spread of influence by the seed set obtained using a sample of training data, while on the right side, we have the overlap of the seed set found with the "true seeds", i.e., the seeds selected by using the complete action logs, i.e., all 6.5M tuples in Flixster_Large and all 13M tuples in Flickr_Large.

As can be seen, the quality of the seed set obtained by using only 1M tuples is as good as using all 6.5M tuples in case of Flixster_Large. Similarly, in Flickr_Large, the influence spread "converges" after 8M tuples.

These observations suggest that we need to use only a small sample of the propagation traces (or action log) to select the seed set and as a result, even though our algorithm can in principle be memory intensive when the action log is huge, in reality, the memory requirements are not that high.

Effect of truncation threshold. Finally, we show the effect of truncation threshold λ on

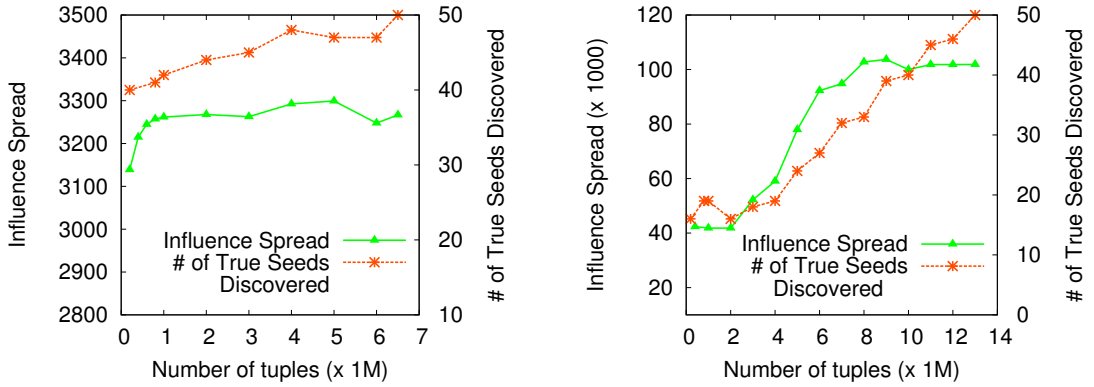


Figure 5.9: Influence spread achieved and number of "true" seeds with respect to number of tuples used on Flixster_Large (left) and Flickr_Large (right).

the accuracy, memory usage and running time of our algorithm in Table 5.4. As expected, as we decrease the truncation threshold, while accuracy (measured in terms of number of "true" seeds discovered and influence spread achieved) improves, memory requirement and running time increase. Both the influence spread and "true seeds discovered" essentially saturate at $\lambda = 0.001$. Note that in all our previous experiments, we used $\lambda = 0.001$ which is a good choice as can be seen from the table. The results on FLICKR_LARGE and on small versions of the datasets are similar.

λ	Influence Spread	True seeds discovered	Memory usage (GB)	Runtime (in min)
0.1	2959	38	2.1	5.25
0.01	3220	45	6	8.62
0.001	**3267**	**48**	**18.8**	**21.25**
0.0005	3267	49	26	25.9
0.0001	3270	50	51	46.7

Table 5.4: Effect of truncation threshold λ on FLIXSTER_LARGE. "True seeds" are the ones obtained when $\lambda = 0.0001$.

5.6 Conclusions and Discussion

While most of the literature on INFLUENCE MAXIMIZATION has focused mainly on the social graph structure, in this chapter we proposed a novel data-based approach, that directly leverages available traces of past propagations.

Our Credit Distribution model directly estimates influence spread by exploiting historical data, thus avoiding the need for learning influence probabilities, and more importantly, avoiding costly Monte Carlo simulations, the standard way to estimate influence spread. Based on this, we developed an efficient algorithm for INFLUENCE MAXIMIZATION. We demonstrated

the accuracy on real data sets by showing the CD model by far is closest to ground truth. We also showed that our algorithm is highly scalable.

Beyond the main contributions, this chapter achieves several side-contributions: (1) Methods which arbitrarily assign influence probabilities suffer from large error in their spread prediction compared with those that learn these probabilities from data. (2) The former methods end up choosing seed sets very different from the latter ones, suggesting the seeds they recommend may well have a poor spread. (3) The greedy algorithm using learned influence probabilities is robust against some noise in the probability learning step. (4) The IC and LT models, using learned influence probabilities, choose seed sets very different from each other, and in turn different from the CD model, which is by far closest to ground truth. These observations further highlight the need for devising techniques and benchmarks for comparing different influence models and the associated Influence Maximization methods.

Acknowledgments. Thanks to Jamali and Ester for sharing the Flixster dataset [74].

Chapter 6

On Minimizing Budget and Time in Influence Propagation over Social Networks

In recent years, study of influence propagation in social networks has gained tremendous attention. In this context, we can identify three orthogonal dimensions – the number of *seed* nodes activated at the beginning (known as *budget*), the expected number of activated nodes at the end of the propagation (known as *expected spread* or *coverage*), and the *time* taken for the propagation. We can constrain one or two of these and try to optimize the third. In their seminal paper, Kempe, Kleinberg and Tardos constrained the budget, left time unconstrained, and maximized the coverage: this problem is known as Influence Maximization.

In this chapter, we study alternative optimization problems which are naturally motivated by resource and time constraints on viral marketing campaigns. In the first problem, termed *Minimum Target Set Selection* (or MINTSS for short), a coverage threshold η is given and the task is to find the *minimum size seed set* such that by activating it, at least η nodes are eventually activated in the expected sense. This naturally captures the problem of deploying a viral campaign on a budget. In the second problem, termed MINTIME, the goal is to minimize the time in which a predefined coverage is achieved. More precisely, in MINTIME, a coverage threshold η and a budget threshold k are given, and the task is to find a seed set of size at most k such that by activating it, at least η nodes are activated in the expected sense, *in the minimum possible time*. This problem addresses the issue of *timing* when deploying viral campaigns. Both these problems are NP-hard, which motivates our interest in their approximation.

For MINTSS, we develop a simple greedy algorithm and show that it provides a bicriteria approximation. We also establish a generic hardness result suggesting that improving this bicriteria approximation is likely to be hard. For MINTIME, we show that even bicriteria and tricriteria approximations are hard under several conditions. We show, however, that if we allow the budget for number of seeds k to be boosted by a logarithmic factor and allow the coverage to fall short, then the problem can be solved *exactly* in PTIME, i.e., we can achieve the required coverage within the time achieved by the optimal solution to MINTIME with budget k and coverage threshold η.

Finally, we establish the value of the approximation algorithms, by conducting an experimental evaluation, comparing their quality against that achieved by various heuristics.

This chapter is based on our SNAM 2012 journal manuscript [58]. This study has been done in collaboration with Francesco Bonchi and Laks V. S. Lakshmanan and Suresh Venkata-

subramanian.

6.1 Introduction

The study of how influence and information propagate in social networks has recently received a great deal of attention [2, 13, 24, 32, 35–37, 42, 55, 56, 78, 79, 82, 109, 139]. One of the central problems in this domain is the problem of Influence Maximization [78]. Consider a social network in which we have accurate estimates of influence among users. Suppose we want to launch a new product in the market by targeting a set of influential users (e.g., by offering them the product at a discounted price), with the goal of starting a word-of-mouth viral propagation, exploiting the power of social connectivity. The idea is that by observing its neighbors adopting the product, or more generally, performing an action, a user may be influenced to perform the same action, with some probability. Influence thus propagates in steps according to one of the propagation models studied in the literature, e.g., the *independent cascade* (IC) or the *linear threshold* (LT) models [78]. The propagation stops when no new user gets activated.

In this context, we can identify three main dimensions – the number of *seed* nodes (or users) activated at the beginning (known as the *budget*), the expected number of nodes that eventually get activated (known as *coverage* or *expected spread*)[38], and the number of *time* steps required for the propagation. In their seminal paper Kempe, Kleinberg and Tardos [78] introduced the problem of Influence Maximization which asks for a seed set with a budget threshold k that maximizes the expected spread (time being left unconstrained). They showed that under the standard propagation models IC and LT, Influence Maximization is NP-hard, but that a simple greedy algorithm that exploits properties of the propagation function yields a $(1 - 1/e - \phi)$-approximation, for any $\phi > 0$ (as discussed in detail in Section 6.2).

In this chapter, we explore the other dimensions of influence propagation. The problem of Minimum Target Set Selection (MINTSS) is motivated by the observation that in a viral marketing campaign, we may be interested in the smallest budget that will achieve a desired outcome. The problem can therefore be defined as follows. We are given a threshold η for the expected spread and the problem is to find a seed set of *minimum size* such that activating the set yields an expected spread of at least η.

In both MINTSS and Influence Maximization, the time for propagation is not considered. Indeed, with the exception of a few papers [see e.g., 91], the temporal dimension of the social propagation phenomenon has been largely overlooked. This is surprising as the timeliness of a *viral marketing* campaign is a key ingredient for its success. Beyond viral marketing, many other applications in time-critical domains can exploit social networks as a means of communication to spread information quickly. This motivates the problem of Minimum Propagation Time (MINTIME), defined as follows: given a budget k and a coverage threshold η, find a seed set that satisfies the given budget and achieves the desired coverage in *as little time as possible*. Thus, MINTIME tries to optimize the propagation time required to achieve a desired coverage under a given budget.

[38] We use the terms coverage and expected spread interchangeably throughout the article.

6.1.1 Our Contributions

We now summarize the main results in this chapter.

- Firstly, we show (Section 6.4, Theorem 7) that for all instances of MINTSS where the coverage function is submodular, a simple greedy algorithm yields a bicriteria approximation: given a coverage threshold η and a shortfall parameter $\epsilon > 0$, the greedy algorithm will produce a solution S: $\sigma(S) \geq \eta - \epsilon$ and $|S| \leq (1 + \ln(\eta/\epsilon))OPT$, where OPT is the optimal size of a seed set whose coverage is at least η. That is, the greedy solution exceeds the optimal solution in terms of size (budget) by a logarithmic factor while achieving a coverage that falls short of the required coverage by the shortfall parameter. We prove a generic hardness result (Section 6.4, Theorem 9) suggesting that improving this approximation factor is likely to be hard.

- For MINTIME under IC and LT model (or any model with monotone submodular coverage functions), we show that when we allow the coverage achieved to fall short of the threshold and the budget k for number of seed nodes to be overrun by a logarithmic factor, then we can achieve the required coverage in the minimum possible propagation time, i.e., in the time achieved by the optimal solution to MINTIME with budget threshold k and coverage threshold η (Section 6.5, Theorem 12).

- On the other hand, for MINTIME under the IC model, we show that even bicriteria and tricriteria approximations are hard. More precisely, let R_{OPT} be the optimal propagation time required for achieving a coverage $\geq \eta$ within a budget of k. Then we show the following (Section 6.5, Theorem 10): there is unlikely to be a PTIME algorithm that finds a seed set with size under the budget, which achieves a coverage better than $(1 - 1/e)\eta$. Similarly, if we limit the budget overrun factor to less than $\ln(\eta)$, then it is unlikely that there is a PTIME algorithm that finds a seed set of size within the overrun budget which achieves a coverage better than $(1 - 1/e)\eta$. In both cases, the result holds even when we permit any amount of slack in the resulting propagation time.

- The above results are bicriteria bounds, in that they allow slack in two of the three parameters governing MINTIME problems. We also show a tricriteria hardness result (Section 6.5, Theorem 11). Namely, if we limit the budget overrun factor to be $\beta < \ln(\eta)$, then it is unlikely that there is a PTIME algorithm that finds a seed set with a size within a factor β of the budget that achieves a coverage better than $(1-1/e^{\beta})\eta$. Similar bounds hold if we place hard limits on the coverage approximation and try to balance overrun in the other parameters.

- Often, the coverage function can be hard to compute exactly. This is the case for both IC and LT models [78]. All our results are robust in that they carry over even when only estimates of the coverage function are available.

- We show the value of our approximation algorithms by experimentally comparing their quality with that of several heuristics proposed in other contexts, using two real data sets. We discuss our findings in Section 6.6.

Related work is discussed in Section 6.3. Section 6.7 concludes the chapter and discusses interesting open problems.

6.2 Problems Studied

The necessary background is covered in Chapter 2. We directly jump to the problems we study in this chapter. Let m stand for any propagation model with a submodular coverage function $\sigma_m(.)$.

Problem 3 (MINTSS)**.** *Let $G = (V, E)$ be a social graph. Given a real number $\eta \leq |V|$, find a set $S \subseteq V$ of the smallest size $|S|$, such that the expected spread, denoted $\sigma_m(S)$, is no less than η.*

Problem 4 (MINTIME)**.** *Let $G = (V, E)$ be a social graph. Given an integer k, and a real number $\eta \leq |V|$, find a set $S \subseteq V$, $|S| \leq k$, and the smallest $t \in \mathbb{N}$, such that the expected spread at time t, denoted $\sigma_m^t(S)$, is no less than η.*

The MINTSS problem is closely related to the real-valued submodular set cover (RSSC) problem. In the submodular set cover (SSC) problem we are given a monotone submodular set function f over a ground set $\mathcal{X}$ and a modular cost function c over $\mathcal{X}$ and are asked to find a set $S \subseteq \mathcal{X}$ minimizing $c(S) := \sum_{s \in S} c(s)$ subject to the constraint that $f(S) = f(\mathcal{X})$. The RSSC problem, which generalizes SSC, is defined as follows: given a submodular function $f : 2^{\mathcal{X}} \to \mathbb{R}$ and a threshold η, find a set $S \subseteq \mathcal{X}$ of the least size (or minimum cost, when elements of $\mathcal{X}$ are weighted) such that $f(S) \geq \eta$. MINTSS under any propagation model such as IC and LT, for which the coverage function is submodular is clearly a special case of RSSC, an observation we exploit in Section 6.4.

MINTIME is closely related to the Robust Asymmetric k-center (RAKC) problem in directed graphs, defined as follows: given a digraph $G = (V, E)$, a (possibly empty) set of forbidden nodes and thresholds k and η, find k or fewer nodes S such that they cover at least η non-forbidden nodes in the minimum possible radius, i.e., each of the η nodes are reachable from some node in S in the minimum possible distance.

6.3 Related Work

While to the best of our knowledge MINTIME has never been studied before, some work has been devoted to MINTSS [16, 33]. Both studies focus on a variant of LT model (DLT model) where a deterministic threshold θ_u is chosen for each node. While coverage under the LT model is submodular, it is not submodular under the DLT model. There have been some recent works that evaluate classical propagation models like IC and LT against real world datasets [17, 57]. We are not aware of any such studies on the DLT model. In this chapter, we study both MINTSS and MINTIME under the classic propagation models IC and LT, under which the coverage function is submodular.

Chen [33] shows that under the DLT propagation model, MINTSS cannot be approximated within a factor of $O(2^{\log^{1-\delta} n})$ unless $NP \subseteq DTIME(n^{O(\log \log n)})$, and also gives a polynomial

time algorithm for MINTSS on trees. Ben-Zwi et al. [16] build upon [33] and develop a $O(n^{O(w)})$ algorithm for solving MINTSS exactly under the DLT model, where w is the tree width of the graph. They show the problem cannot be solved in $n^{O(\sqrt{w})}$ time unless all problems in SNP can be solved in sub-exponential time.

A few classical cover-problems are related to the problems we study. One such problem is Maximum Coverage (MC): given a collection of sets $\mathcal{S}$ over a ground set $\mathcal{U}$ and budget k, find a subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $|\mathcal{C}| \leq k$ and $|\bigcup \mathcal{C}|$ is maximized. The problem can be approximated within a factor of $(1 - 1/e)$ and it cannot be improved [44, 80]. Similar results by Khuller et al. [80] and Sviridenko [126] exist for the weighted case.

Another relevant problem is Partial Set Cover (PSC): given a collection of sets $\mathcal{S}$ over the ground set $\mathcal{U}$ and a threshold η, the goal is to find a subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $|\bigcup \mathcal{C}| \geq \eta$ and $|\mathcal{C}|$ is minimized. While PSC can be approximated within a factor of $\lceil \ln \eta \rceil$, Feige [44] showed that it cannot be approximated within a factor of $(1 - \delta) \ln \eta$, for any fixed $\delta > 0$, unless $NP \subseteq DTIME(n^{O(\log \log n)})$.

Our results on MINTSS exploit its connection to the real-valued submodular set cover (RSSC) problem. There has been substantial work on submodular set cover (SSC) in the presence of integer-valued submodular functions[14, 44, 46, 47, 123]. Relatively much less work has been done on real-valued SSC. For non-decreasing real-valued submodular functions, Wolsey [140] has shown, among other things, that a simple greedy algorithm yields a solution to a special case of SSC where $\eta = f(\mathcal{X})$, that is within a factor of $\ln[\eta/(\eta - f(S_{t-1})]$ of the optimal solution, where t is the number of iterations needed by the greedy algorithm to achieve a coverage of η and S_i denotes the greedy solution after i iterations. Unfortunately, this result by itself does *not* yield an approximation algorithm with any guaranteed bounds: in Section 6.3.1 we give an example to show that the greedy solution can be arbitrarily worse than the optimal one. Furthermore, Wolsey's analysis is restricted to the case $\eta = f(\mathcal{X})$. Along the way to establishing our results on MINTSS, we show the greedy algorithm yields a bicriteria approximation for real-valued SSC that extends to the general case of partial cover with $\eta \leq f(\mathcal{X})$, and where elements are weighted.

Our results on MINTIME leverage its connection to the robust asymmetric k-center problem (RAKC). It has been shown that, while asymmetric k-center problem can be approximated within a factor of $O(\log^* n)$ [104], RAKC cannot be approximated within any factor unless P = NP [92].

6.3.1 Example Illustrating Performance of Wolsey's solution

The following example shows the greedy solution with threshold η can be arbitrarily worse than the optimum.

Example (Illustrated also in Figure 6.1). Consider a ground set $\mathcal{X} = \{w_1, w_2, v_1, v_2, ..., v_l\}$ with elements having unit costs. Figure 6.1 geometrically depicts the definition of a function $f : 2^{\mathcal{X}} \rightarrow \mathbb{R}$, where for any set $S \subset \mathcal{X}$, $f(S)$ is defined to be the area (shown shaded) covered by the elements of S. Specifically, $f(w_1) = f(w_2) = 1 - 1/2^{l+1}$ and $f(v_i) = 1/2^{i-1}$, $1 \leq i \leq l$. Notice, $f(\{v_1, ..., v_l\}) = \Sigma_{i=1}^{l} 1/2^{i-1} = 2 - 1/2^{l-1} < 2 - 1/2^l = f(\{w_1, w_2\})$. The greedy algorithm will first pick v_1. Suppose it picks $S = \{v_1, ..., v_i\}$ in i rounds. Then $f(S \cup \{v_{i+1}\}) - f(S) = 1/2^i > 1 - 1/2^{l+1} - 1 + 1/2^i = 1 - 1/2^{l+1} - 1/2(2 - 1/2^{i-1}) = f(S \cup \{w_1\}) - f(S)$.

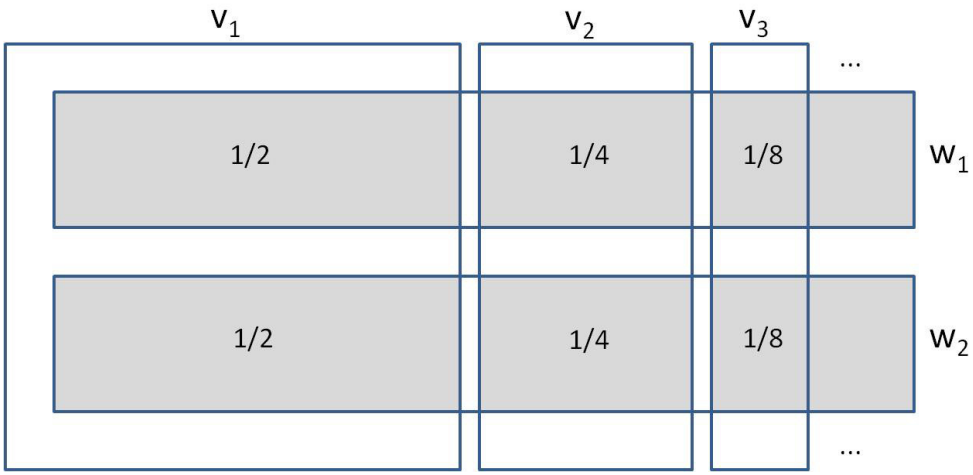


Figure 6.1: Example. Rectangles represent the elements in the universe. The shaded area within a rectangle represents the coverage function f for the element. e.g., $f(v_1) = 1/2+1/2 = 1$.

Thus, greedy will never pick w_1 or w_2 before it picks $v_1, ..., v_l$. Suppose $\eta = 2 - 1/2^l$. Clearly, the greedy solution is $\mathcal{X}$ whereas the optimal solution is $\{w_1, w_2\}$. Here l can be arbitrarily large.

6.4 Minimum Target Set Selection

6.4.1 A Bicriteria Approximation

Our main result of this section is that a simple greedy algorithm, Algorithm Greedy-Mintss, yields a bicriteria approximation to (weighted) MINTSS, for any propagation model whose coverage function is monotone and submodular.

Algorithm 16 Greedy-Mintss

Input: $G, \eta, \epsilon, \sigma_m$
Output: seed set S

1: $S \leftarrow \emptyset$
2: **while** $\sigma_m(S) < \eta - \epsilon$ **do**
3: $\quad u \leftarrow \arg\max_{w \in V \setminus S}(\frac{\min(\sigma_m(S \cup \{w\}), \eta) - \sigma_m(S)}{c(w)})$;
4: $\quad S \leftarrow S \cup \{u\}$
5: **end while**

In order to prove the results in the most general setting, we consider digraphs $G = (V, E)$ which have non-negative node weights: we are given a cost function $c : V \to \mathbb{R}^+$ in addition to the coverage threshold η, and need to find a seed set S such that $\sigma_m(S) \geq \eta$ and $c(S) = \sum_{x \in S} c(x)$ is minimum. Clearly, this generalizes the unweighted case.

Theorem 7. *Let $G = (V, E)$ be a social graph, with node weights given by $c : V \to \mathbb{R}^+$. Let m be any propagation model whose coverage function $\sigma_m(.)$ is monotone and submodular. Let $\S^*$ be a seed set of minimum cost such that $\sigma_m(\S^*) \geq \eta$. Let $\epsilon > 0$ be any shortfall and let S be the greedy solution with chosen threshold $\eta - \epsilon$. Then, $c(S) \leq c(S^*) \cdot (1 + \ln(\eta/\epsilon))$.*

In the rest of this section, we prove this result. We first observe that every instance of MINTSS where the coverage function $\sigma_m(.)$ is monotone and submodular is an instance of RSSC. Thus, it suffices to prove Theorem 7 for RSSC, for which we adapt a bicriterion approximation technique by [123].

Let $\mathcal{X} = \{x_1, x_2, ..., x_m\}$ be a ground set, $c : \mathcal{X} \to \mathbb{R}^+$ be a cost function, $f : 2^{\mathcal{X}} \to \mathbb{R}$ a non-negative monotone submodular function and η a given threshold. Apply the greedy algorithm above to this instance of RSSC. Let S_i be the (partial) solution obtained by the greedy algorithm after i iterations. Let t be the smallest number such that $f(S_t) \geq \eta$. We define $g(S) = \min(f(S), \eta)$. Clearly, g is also monotone and submodular. In each iteration, the greedy algorithm picks an element which provides the maximum marginal gain per unit cost (w.r.t. g), i.e., it picks an element x for which $\frac{g(S \cup \{x\}) - g(S)}{c(x)}$ is positive and is maximum.

Let $c(S^*) = \kappa$ and define $\eta_i = \eta - g(S_i)$, i.e., the shortfall in coverage after i iterations of the greedy algorithm.

Lemma 4. *At the end of iteration i, there is an element $x \in \mathcal{X} \setminus \S_i$: $\frac{g(S_i \cup \{x\}) - g(\S_i)}{c(x)} \geq \frac{\eta_i}{\kappa}$.*

Proof. Let $\S_i^* = \S^* - \S_i$. Let $\S_i^* = \{y_1, ..., y_t\}$ and $c(S_i^*) = \kappa_i$. Suppose $\forall x \in \mathcal{X} \setminus \S_i$: $\frac{g(\S_i \cup \{x\}) - g(\S_i)}{c(x)} < \frac{\eta_i}{\kappa}$. Consider adding the elements in $\S_i^*$ to $\S_i$ one by one. Clearly, at any step $j \leq t$, we have by submodularity that

$$g(\S_i \cup \{y_1, ..., y_j\}) - g(\S_i \cup \{y_1, ..., y_{j-1}\}) \leq g(\S_i \cup \{y_j\}) - g(\S_i) < c(y_j) \cdot \frac{\eta_i}{\kappa}$$

Iterating over all j, this yields $g(\S_i \cup \{y_1, ..., y_j\}) - g(\S_i) < \frac{\eta_i}{\kappa} \cdot (c(y_1) + ... + c(y_j))$ resulting in $g(\S_i \cup \{y_1, ..., y_t\}) < g(S_i) + \frac{\eta_i}{\kappa} \cdot \sum_{1 \leq j \leq t} c(y_j) \leq \eta$ which is a contradiction since the left hand side is no less than the optimal coverage. □

Proof of Theorem 7:

It follows from Lemma 4 that $\eta_i \leq \eta_{i-1}(1 - c_i/\kappa)$ where c_i is the cost of the element added in iteration i. Using the well known inequality $(1 + z) \leq e^z, \forall z$, we get $\eta_i \leq \eta_{i-1} \cdot e^{-c_i/k}$. Expanding, $\eta_i \leq \eta \cdot e^{-\frac{1}{k} \cdot \sum_i c_i}$. Let the algorithm take l iterations to achieve coverage $g(S_l) \geq \eta - \epsilon$ such that $g(S_{l-1}) < \eta - \epsilon$. At any step, $g(S_{i+1}) - g(S_i) \leq \eta_i$. Thus, $c_i \leq \kappa$, and in particular, the cost of the last element picked can be at most κ. So, $c(S_l) \leq \kappa + c(S_{l-1})$. $g(S_{l-1}) < \eta - \epsilon$ implies $\eta_{l-1} > \epsilon$. Hence, we have $\eta e^{-\frac{1}{\kappa} c(S_{l-1})} > \epsilon$ which implies $c(S_{l-1}) < \kappa \ln(\eta/\epsilon)$. Thus, $c(S_l) \leq \kappa(1 + \ln(\eta/\epsilon))$. □

Using a similar analysis, it can be shown that when the costs are uniform, the approximation factor can be improved to $\lceil \ln(\eta/\epsilon) \rceil$.

For propagation models like IC and LT, computing the coverage $\sigma_m(S)$ exactly is #P-hard [35, 37] and thus we must settle for estimates. To address this, we "lift" the above theorem to the case where only estimates of the function $f(.)$ are available. We can show:

Theorem 8. *For any $\phi > 0$, there exists a $\delta \in (0,1)$ such that using $(1-\delta)$-approximate values for the coverage function $\sigma_m(\cdot)$, the greedy algorithm approximates MINTSS under IC and LT models within a factor of $(1+\phi)\cdot(1+\ln(\eta/\epsilon))$.*

Proof. The proof involves a more careful analysis of how error propagates in the greedy algorithm if, because of errors, the greedy algorithm picks the wrong point.

Here, we give the proof for the unit cost version only. Consider any monotone, submodular function $f(\cdot)$. Thus, in the statement of theorem, $\sigma_m(\cdot) = f(\cdot)$. Let $f'(\cdot)$ be its approximated value. In any iteration, the (standard) greedy algorithm picks an element which provides maximum marginal gain. Let S_i be the set formed after iteration i.

As we did in Lemma 4, it is straightforward to show that there must exists an element $x \in \mathcal{X} \setminus S_i$ such that $f(S_i \cup \{x\}) - f(S_i) \geq \eta_i/k$ where $\eta_i = \eta - f'(S_i)$. Without loss of generality, let x be the element which provides the maximum marginal gain. Suppose that due to the error in computing $f(.)$, some other element y is picked instead. Then,

$$(1-\delta)f(S_i \cup \{x\}) \leq f'(S_i \cup \{x\}) \leq f'(S_i \cup \{y\})$$

Moreover, $f'(S_i) \leq f(S_i)$. Thus,

$$\begin{aligned}
&\frac{\eta_i}{k} \leq f(S_i \cup \{x\}) - f(S_i) \leq \frac{f'(S_i \cup \{y\})}{1-\delta} - f'(S_i) \\
\implies &\frac{\eta_i}{k} \leq \frac{\eta - \eta_{i+1}}{1-\delta} - \eta + \eta_i \\
\implies &\eta_{i+1} \leq \eta_i \cdot (1-\delta) \cdot \left(1 - \frac{1}{k}\right) + \delta \cdot \eta \\
\implies &\eta_{i+1} \leq \eta \cdot (1-\delta)^{i+1} \cdot \left(1 - \frac{1}{k}\right)^{i+1} \\
&\quad + \delta \cdot \eta \cdot \left(\frac{1 - (1-\delta)^{i+1}(1-1/k)^{i+1}}{1 - (1-\delta)(1-1/k)}\right)
\end{aligned}$$

Let $\delta' = \delta/(1-(1-\delta)(1-1/k))$. Let the greedy algorithm takes l iterations. Then,

$$\begin{aligned}
\eta_l &\leq \eta \cdot (1-\delta)^l \cdot \left(1 - \frac{1}{k}\right)^l \\
&\quad + \delta' \cdot \eta \cdot \left(1 - (1-\delta)^l \cdot \left(1 - \frac{1}{k}\right)^l\right) \\
&= \eta \cdot (1-\delta)^l \cdot \left(1 - \frac{1}{k}\right)^l (1-\delta') + \delta' \cdot \eta
\end{aligned}$$

Using $(1-\delta)^l \leq 1$ and $(1-1/k)^l \leq e^{-l/k}$,

$$\eta_l \leq \eta e^{-l/k}(1-\delta') + \delta' \cdot \eta$$

The algorithm stops when $\eta_l \leq \epsilon$. The maximum number of iterations needed to ensure this are

$$l \leq k\left(1 + \ln \frac{\eta(1-\delta')}{\epsilon(1-\delta'\eta/\epsilon)}\right)$$

Let $x = \eta/\epsilon$. To prove the lemma, we need to prove that for any $\phi > 0$, there exists $\delta \in [0, 1)$ such that

$$x^{1+\phi} = x\frac{1-\delta'}{1-\delta' x} \implies \delta' = \frac{x^{\phi}-1}{x^{1+\phi}-1}$$

Clearly, for any $\phi \geq 0$, $\delta' \in [0, 1)$. Hence,

$$\begin{aligned} & 0 \leq \delta < 1 - (1-\delta)(1-1/k) \\ \iff & 0 \leq \delta < 1 \end{aligned}$$

This completes the proof for unit cost case. Using the slight modification in the greedy algorithm (as we did in proving theorem 1), the same result can be obtained for weighted version. □

6.4.2 An Inapproximability Result

Recall that every instance of MINTSS where the coverage function is monotone and submodular is an instance of RSSC. Consider the unweighted version of the RSSC problem. Let S^* denote an optimal solution and let $OPT = |S^*|$.

Theorem 9. *For any fixed $\delta > 0$, there does not exist a PTIME algorithm for RSSC that guarantees a solution $\S : |\S| \leq OPT(1-\delta)\ln(\eta/\epsilon)$, and $f(\S) \geq \eta - \epsilon$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{O(\log\log n)})$.*

Proof. Case 1: $\epsilon \geq 1$. Suppose there exists an algorithm $\mathcal{A}$ that finds a solution S of size $\leq OPT(1-\delta)\ln(\eta/\epsilon)$ such that $f(S) \geq \eta - \epsilon$ for any $\epsilon \geq 1$. Consider an arbitrary instance $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \eta \rangle$ of PSC, which is a special case of RSSC. Apply the algorithm $\mathcal{A}$ to $\mathcal{I}$. It outputs a collection of sets $\mathcal{C}_1 : |\mathcal{C}_1| \leq OPT(1-\delta)\ln(\eta/\epsilon)$ that covers $\geq \eta - \epsilon$ elements in $\mathcal{U}$.

Create a new instance $\mathcal{J} = \langle \mathcal{U}', \mathcal{S}', \eta' \rangle$ of PSC as follows. Let $T = \bigcup \mathcal{C}_1$ be the set of elements of $\mathcal{U}$ covered by $\mathcal{C}_1$. Define $\mathcal{S}' = \{S \setminus T \mid \S \in \mathcal{S} \setminus \mathcal{C}_1\}$, $\mathcal{U}' = \mathcal{U} \setminus T$ and $\eta' = \epsilon$. Set the new shortfall $\epsilon' = 1$. Apply the algorithm $\mathcal{A}$ to $\mathcal{J}$. It will output another collection of sets $\mathcal{C}_2 : |\mathcal{C}_2| \leq OPT(1-\delta)\ln\epsilon$ which covers $\geq \epsilon - 1$ elements in $\mathcal{U}'$.[39] Let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. The number of elements covered by $\mathcal{C}$ is $\geq \eta - \epsilon + \epsilon - 1 = \eta - 1$. Clearly, $|\mathcal{C}| = |\mathcal{C}_1| + |\mathcal{C}_2| \leq OPT(1-\delta)\ln(\eta/\epsilon) + OPT(1-\delta)\ln(\epsilon) = OPT(1-\delta)\ln(\eta)$. Thus, we have a solution for PSC with the approximation factor of $(1-\delta)\ln(\eta)$, which is not possible unless $NP \subseteq DTIME(n^{O(\log\log n)})$ [44]. This proves Case 1.

Case 2: $\epsilon < 1$. Assume an arbitrary instance $\mathcal{I}$ of RSSC with monotone submodular function $f : 2^{\mathcal{X}} \to \mathbb{R}$. Let η' be the coverage threshold and $\epsilon' \geq 1$ be any given shortfall.

[39] If $\epsilon = 1$, $\mathcal{A}$ outputs an empty collection.

We now construct another instance $\mathcal{J}$ of RSSC as follows: Set the coverage function $g(S) = f(S)/x$, coverage threshold $\eta = \eta'/x$ and shortfall $\epsilon = \epsilon'/x$. Choose any value of $x > 1$ such that $\epsilon = \epsilon'/x < 1$. We now show that if a solution is a $(1-\delta)\ln(\eta/\epsilon)$-approximation to the optimal solution for $\mathcal{J}$ then it is a $(1-\delta)\ln(\eta'/\epsilon')$-approximation to the optimal solution for $\mathcal{I}$. Clearly, the optimal solution for both the instances are identical, so $OPT_{\mathcal{I}} = OPT_{\mathcal{J}}$.[40] Suppose there exists an algorithm for RSSC when the shortfall is $\epsilon \in (0,1)$, that guarantees a solution $\S : |\S| \leq OPT(1-\delta)\ln(\eta/\epsilon)$ and $f(\S) \geq \eta - \epsilon$. Apply this algorithm to instance $\mathcal{J}$ to obtain a solution $\S_{\mathcal{J}}$. We have: $g(\S_{\mathcal{J}}) \geq \eta - \epsilon = (\eta' - \epsilon')/x$. It implies $f(\S_{\mathcal{J}}) = x \cdot g(\S_{\mathcal{J}}) \geq \eta' - \epsilon'$. Moreover, $|\S_{\mathcal{J}}| \leq OPT_{\mathcal{J}}(1-\delta)\ln(\eta/\epsilon)$, implying $|\S_{\mathcal{J}}| \leq OPT_{\mathcal{I}}(1-\delta)\ln(\eta'/\epsilon')$. Thus we have the solution $S_{\mathcal{J}}$ for instance $\mathcal{I}$ whose size is $\leq OPT_{\mathcal{I}}(1-\delta)\ln(\eta'/\epsilon')$. The theorem follows. □

In view of this generic result, we conjecture that improving the approximation factor for MINTSS to $(1-\delta)\ln(\eta/\epsilon)$ for IC and LT is likely to be hard.

6.5 MINTIME

In this section, we study MINTIME under the IC model. Denote by $\sigma_m^R(S)$ the expected number of nodes activated under model m within time R, and let η be the desired coverage and k be the desired budget. Let R_{OPT} denote the optimal propagation time under these budget and coverage constraints. Our first result says that efficient approximation algorithms are unlikely to exist under two scenarios: (i) when we allow a coverage shortfall of less than η/e and (ii) when we allow a budget overrun less than $\ln \eta$. In the former scenario, we have a strict budget threshold and in the latter we have a strict coverage threshold. In both cases, we allow any amount of slack in propagation time.

Theorem 10. *Unless $NP \subseteq DTIME(n^{O(\log\log n)})$, there does not exist a PTIME algorithm for MINTIME that guarantees (for any $\alpha \geq 1$):*

1. *a (α, γ)-approximation, such that $|\S| \leq k$, $R = \alpha \cdot R_{OPT}$ and $\sigma_m^R(\S) \geq \gamma \cdot \eta$ where $\gamma = (1 - 1/e + \delta)$ for any fixed $\delta > 0$; or*
2. *a (α, β)-approximation, such that $|\S| \leq \beta \cdot k$, $R = \alpha \cdot R_{OPT}$ and $\sigma_m^R(\S) \geq \eta$ where $\beta = (1-\delta)\ln \eta$ for any fixed $\delta > 0$.*

Our second theorem says efficient approximation algorithms are unlikely to exist under more liberal scenarios than those given above: (i) when for a given budget overrun factor $\beta < \eta$, the fraction of the coverage we want to achieve is more than $1 - 1/e^{\beta}$ and (ii) when for a given fraction $\gamma \in (0, 1 - 1/\eta]$ of the coverage we want to achieve, the budget overrun factor we allow is less than $\ln(1/(1-\gamma))$. As before, we allow any amount of slack in propagation time.

Theorem 11. *Unless $NP \subseteq DTIME(n^{O(\log\log n)})$ there does not exist a PTIME algorithm for MINTIME that guarantees (α, β, γ)-approximation factor (for any $\alpha \geq 1$) such that $|\S| \leq \beta \cdot k$, $R = \alpha \cdot R_{OPT}$ and $\sigma_m^R(\S) \geq \gamma \cdot \eta$ where*

[40] Here, $OPT_{\mathcal{I}}$ and $OPT_{\mathcal{J}}$ represent the size of the optimal solution for instances $\mathcal{I}$ and $\mathcal{J}$ respectively.

1. $\beta \in [1, \ln \eta)$ *and* $\gamma = 1 - 1/e^{\beta} + \delta$ *for any fixed* $\delta > 0$*; or*

2. $\gamma \in \left(0, 1 - \frac{1}{\eta}\right]$ *and* $\beta = (1-\delta)\ln\left(\frac{1}{1-\gamma}\right)$ *for any fixed* $\delta > 0$.

Finally, on the positive side, we show that when a coverage shortfall of $\epsilon > 0$ is allowed and a budget boost of $(1 + \ln(\eta/\epsilon))$ is allowed, we can in PTIME find a solution which achieves the relaxed coverage under the relaxed budget in optimal propagation time. More precisely, we have:

Theorem 12. *Let the chosen coverage threshold be* $\eta - \epsilon$*, for* $\epsilon > 0$ *and chosen budget threshold be* $k(1+\ln(\eta/\epsilon))$*. If the coverage function* $\sigma_m^R(\cdot)$ *can be computed exactly, then there is a greedy algorithm that approximates the MINTIME problem within a* (α, β, γ) *factor where* $\alpha = 1$*,* $\beta = 1 + \ln(\eta/\epsilon)$ *and* $\gamma = 1 - \epsilon/\eta$ *for any* $\epsilon > 0$*. Furthermore, for every* $\phi > 0$*, there is a* $\delta > 0$ *such that by using a* $(1-\delta)$*-approximate values for the coverage function* $\sigma_m^R(\cdot)$*, the greedy algorithm approximates the MINTIME problem within a* (α, β, γ) *factor where* $\alpha = 1$*,* $\beta = (1+\phi)(1+\ln(\eta/\epsilon))$ *and* $\gamma = 1 - \epsilon/\eta$.

6.5.1 Inapproximability Proofs

We next prove Theorems 10 and 11. We first show that MINTIME under the IC model generalizes the RAKC problem. In a digraph $G = (V, E)$ and sets of nodes $S, T \subset V$, say that *R-covers* T if for every $y \in T$, there is a $x \in S$ such that there is a path of length $\leq R$ from x to y. Given an instance of RAKC, create an instance of MINTIME by labeling each arc in the digraph with a probability 1. Now, it is easy to see that for any set of nodes S and any $0 \leq R \leq n-1$, S R-covers a set of nodes T iff activating the seed nodes S will result in the set of nodes T being activated within R time steps. Notice that since all the arcs are labeled with probability 1, all influence attempts are successful by construction. It follows that RAKC is a special case of MINTIME under the IC model.

The tricriteria inapproximability results of Theorem 11 subsume the bicriteria inapproximability results of Theorem 10. Still, in our presentation, we find it convenient to develop the proofs first for bicriteria. Since we showed that MINTIME under IC generalizes RAKC, it suffices to prove the theorems in the context of RAKC. It is worth pointing out [92] proved that it is hard to approximate *RAKC* within any factor unless $P = NP$. Their proof only applies to (the standard) unicriterion approximation.

For a set of nodes S in a digraph we denote by $f^R(S)$ the number of nodes that are R-covered by S. Recall the problems MC and PSC (see Section 6.3).

Proof of Theorem 10: It suffices to prove the theorem for RAKC. For claim 1, we reduce Maximum Coverage (MC) to RAKC and for claim 2, we reduce PSC to RAKC. The reduction is similar and is as follows: Consider an instance of the decision version of MC (equivalently PSC) $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, k, \eta \rangle$, where we ask whether there exists a subcollection $\mathcal{C} \subseteq \mathcal{S}$ of size $\leq k$ such that $|\bigcup_{S \in \mathcal{S}} S| \geq \eta$. Construct an instance $\mathcal{J} = \langle \mathcal{G}, k', \eta' \rangle$ of RAKC as follows: the graph $\mathcal{G}$ consists of two classes of nodes – A and B. For each $S \in \mathcal{S}$, create a class A node v_S and for each $u \in U$, create a class B node v_u. There is a directed edge (v_S, v_u) of unit length iff $u \in S$. Notice, a set of nodes S in $\mathcal{G}$ R-covers another non-empty set of nodes iff S 1-covers the latter set. Moreover, x sets in $\mathcal{S}$ cover y elements in $\mathcal{U}$ iff $\mathcal{G}$ has a set of x nodes which

1-covers $y+x$ nodes. The only-if direction is trivial. For the if direction, the only way x nodes can 1-covers $y+x$ nodes in $\mathcal{G}$ is when the x nodes are from class A.

Next, we prove the first claim. Set $k' = k$ and $\eta' = \eta + k$. Assume there exists a PTIME (α, γ)-approximation algorithm $\mathcal{A}$ for RAKC such that $f^R(\S) \geq (1-1/e+\delta)\cdot(\eta')$ for any fixed $\delta > 0$, for some $R \leq \alpha R_{OPT}$. Apply algorithm $\mathcal{A}$ to the instance $\mathcal{J}$. Notice, for our instance, $R_{OPT} = 1$. The coverage by the output seed set $\S$ will be $f^R(\S) \geq (1-1/e+\delta)\cdot(\eta+k)$ nodes, for some $R \leq \alpha \cdot 1$, implying that the number of class B nodes covered is $\geq (1-1/e+\delta)\cdot(\eta+k)-k$ $= (1 - 1/e + \delta - (1/e - \delta)k/\eta)\eta$. Thus the algorithm approximates MC within a factor of $\left(1 - \frac{1}{e} + \delta - \left(\frac{1}{e} - \delta\right)\frac{k}{\eta}\right)$. Let $\delta' = \delta - \left(\frac{1}{e} - \delta\right)\frac{k}{\eta}$. If we show $\delta' > 0$, we are done, since MC cannot be approximated within a factor of $(1 - 1/e + \delta')$ for any $\delta' > 0$ unless $NP \subseteq DTIME(n^{O(\log\log n)})$ [44, 80]. Clearly, δ' is not always positive. However, for a given δ and k, δ' is an increasing function of η and reaches δ in the limit. Hence there is a value $\eta_0 : \forall \eta \geq \eta_0$, $\delta' > 0$. That is, there are infinitely many instances of PSC for which $\mathcal{A}$ is a $(1 - 1/e + \delta')$-approximation algorithm, where $\delta' > 0$, which proves the first claim.

Next, we prove the second claim. Set $k' = k$ and $\eta' = \eta + x$. The value of x will be decided later. Assume there exists a PTIME (α, β)-approximation algorithm $\mathcal{A}$ for RAKC where $\beta = (1 - \delta)\ln(\eta')$ for any fixed $\delta > 0$. Apply the algorithm to $\mathcal{J}$. It gives a solution S such that $|S| \leq k \cdot (1 - \delta)\ln(\eta + x)$ that covers $\geq \eta + x$ nodes. A difficulty arises here since δ can be arbitrarily close to 1 making $k \cdot (1 - \delta)\ln(\eta + x)$ arbitrarily small, for any given η and k. However, as we argued in the proof of claim 1, for sufficiently large η, we can always find an x: $k \leq x \leq k \cdot (1 - \delta)\ln(\eta + x)$. That is, on infinitely many instances of PSC, algorithm $\mathcal{A}$ finds a set of $|S|$ class A nodes which R-covers $\eta + x$ nodes, for some $R \leq \alpha \cdot 1$. Without loss of generality, we can assume $x \leq \eta$. Choose the smallest value of x such that the solution S covers $\geq \eta$ class B nodes. This implies the number of class A nodes covered is $\leq x$ and so $|S| \leq x$. Thus, on all such instances, algorithm $\mathcal{A}$ gives a solution S of size $\leq x$: $k \leq x \leq k \cdot (1 - \delta)\ln(\eta + x)$ that covers $\geq \eta$ nodes. If we show that the upper bound is equal to $k \cdot (1 - \delta')\ln\eta$ for some $\delta' > 0$, we are done, since PSC cannot be approximated within a factor of $(1 - \delta')\ln\eta$ unless $NP \subseteq DTIME(n^{O(\log\log n)})$ [44].

Let $(1-\delta')\ln\eta = (1-\delta)\ln(\eta+x)$, which yields $\delta' = 1-(1-\delta)\frac{\ln(\eta+x)}{\ln\eta}$. It is easy to see that by choosing sufficiently large η, we can make the gap between δ and δ' arbitrarily small and thus can always ensure $\delta' > 0$ on infinitely many instances of PSC, on each of which algorithm $\mathcal{A}$ will serve as an $(1 - \delta')\ln\eta$-approximation algorithm proving claim 2. □

Note, in the proofs of both claim 1 and 2 in the above theorem, by choosing η sufficiently large, we can always ensure for any given k and $\delta > 0$, the corresponding δ' is always greater than 0. To prove the tricriteria hardness results, we need the following lemma.

Lemma 5. *In the MC (or PSC) problem, let k be the minimum number of sets needed to cover $\geq \eta$ elements. Then, unless $NP \subseteq DTIME(n^{O(\log\log n)})$, there does not exist a PTIME algorithm that is guaranteed to select βk sets covering $\geq \gamma\eta$ elements where*

1. *$\beta \in [1, \ln\eta)$ and $\gamma > 1 - 1/e^{\beta}$; or*
2. *$\gamma \in \left(0, 1 - \frac{1}{\eta}\right]$ and $\beta = (1 - \delta)\ln\left(\frac{1}{1-\gamma}\right)$ for any fixed $\delta > 0$.*

Proof: Suppose there exists an algorithm $\mathcal{A}$ that selects βk sets which covers $\gamma\eta$ elements. Apply $\mathcal{A}$ to an arbitrary instance $\langle \mathcal{U}, \mathcal{S}, \eta \rangle$ of PSC. The output is a collection of sets $\mathcal{C}_1$ such that $|\mathcal{C}_1| \leq \beta k$ and $|\bigcup_{\S \in \mathcal{C}_1} \S| \geq \gamma\eta$. Next, discard the sets that have been selected and the elements they cover, and apply again the algorithm $\mathcal{A}$ on the remaining universe. Repeat this process until 1 or fewer elements are left uncovered.[41]

Let η_i denote the number of elements uncovered after iteration i. In iteration i, the algorithm picks βk sets and covers at least $\gamma\eta_{i-1}$ elements. Hence, $\eta_i \leq \eta_{i-1} \cdot (1-\gamma)$. Expanding, $\eta_i \leq \eta \cdot (1-\gamma)^i$. Suppose after l iterations, $\eta_l = 1$. The total number of sets picked is $l\beta k$. $\eta \cdot (1-\gamma)^l = 1$ implies $l = \frac{\ln \eta}{\ln \frac{1}{1-\gamma}}$.

We now prove the first claim. Let $\gamma > 1 - 1/e^\beta$, then $\ln\left(\frac{1}{1-\gamma}\right) > \beta$. This yields a PTIME algorithm for PSC which outputs a solution of size $l\beta k = \beta k \cdot \ln \eta / \ln \frac{1}{1-\gamma} \leq c \cdot k \ln \eta$ (for some $c < 1$) This yields an $c \cdot \ln \eta$-approximation for PSC for some $c < 1$, which is not possible unless $NP \subseteq DTIME(n^{O(\log\log n)})$ [44].

To prove the second claim, assume $\beta \leq (1-\delta)\ln\left(\frac{1}{1-\gamma}\right)$. This gives a PTIME algorithm for PSC which outputs a solution of size $l\beta k = \beta k \cdot \ln \eta / \ln \frac{1}{1-\gamma} \leq (1-\delta)k \cdot \ln \eta$ which is not possible unless $NP \subseteq DTIME(n^{O(\log\log n)})$. □

We are now ready to prove Theorem 11.

Proof of Theorem 11: Again, it suffices to prove the theorem for RAKC. For claim 1, we reduce MC to RAKC and for claim 2, we reduce PSC to RAKC. The reduction is the same as in the proof of Theorem 10 and we skip the details here. Below, we refer to instances $\mathcal{I}$ and $\mathcal{J}$ as in that proof.

We first prove claim 1. Given any β, set $k' = k$ and $\eta' = \eta + \beta k$. Assume there exists a PTIME (α, β, γ)-approximation algorithm $\mathcal{A}$ for RAKC which approximates the problem within the factors as mentioned in claim 1. Apply algorithm $\mathcal{A}$ to the instance $\mathcal{J}$. The coverage by the output seed set $\S$ will be $f^R(\S) \geq (1 - 1/e^\beta + \delta) \cdot (\eta + \beta k)$ nodes, implying the number of class B nodes covered is $\geq (1 - 1/e^\beta + \delta) \cdot (\eta + \beta k) - \beta k = (1 - 1/e^\beta + \delta - (1/e^\beta - \delta)\beta k/\eta)\eta$. Thus the algorithm approximates MC within a factor of $\left(1 - \frac{1}{e^\beta} + \delta - \left(\frac{1}{e^\beta} - \delta\right)\frac{\beta k}{\eta}\right)$.

If we show $\delta - \left(\frac{1}{e^\beta} - \delta\right)\frac{\beta k}{\eta} > 0$, then the claim follows, since MC cannot be approximated within a factor of $(1 - 1/e^\beta + \delta')$ for any $\delta' > 0$ unless $NP \subseteq DTIME(n^{O(\log\log n)})$, by Lemma 5. Let $\delta' = \delta - \left(\frac{1}{e^\beta} - \delta\right)\frac{\beta k}{\eta}$. For any $\beta \in [1, \ln \eta)$, δ' is an increasing function of η which approaches δ in the limit. Thus, given any fixed $\delta > 0$, there must exist some η_o such that for any $\eta \geq \eta_o$, $\delta' > 0$. This proves the first claim (by an argument similar to that in Theorem 10).

Next, we prove the second claim. Set $k' = k$ and $\eta' = \eta + x$. The value of x will be decided later. Assume that there exists a PTIME (α, β, γ)-approximation algorithm $\mathcal{A}$ for RAKC where the factors α, β and γ satisfy the conditions as mentioned in claim 2. Apply the algorithm to instance $\mathcal{J}$. For any $\gamma_j \in (0, 1 - 1/(\eta + x)]$, it gives a solution of size $\leq k \cdot (1-\delta)\ln(1/(1-\gamma_j))$ that covers $\gamma_j \cdot (\eta + x)$ nodes. There can be $|\mathcal{S}|$ possible choices of x.

[41] Instead of 1, we could be left with a constant number of elements. Asymptotically, it does not make a difference.

Pick the smallest x such that number of nodes covered in class B is at least $\gamma_j\eta$, implying that the number of nodes picked from class A is $\gamma_j x$. Thus, $\gamma_j x \leq k \cdot (1-\delta)\ln(1/(1-\gamma_j))$. The existence of x satisfying this inequality can be established as done for claim 2 in Theorem 10.

Thus, algorithm $\mathcal{A}$ gives the solution instance $\mathcal{I}$ of size $\leq k \cdot (1-\delta)\ln(1/(1-\gamma_j))$ that covers $\gamma_j\eta$ elements in $\mathcal{U}$ where $\gamma_j \in (0, 1-1/(\eta+x)]$. If we show that for any given $\delta > 0$ and γ_j in the range, there exists some $\delta' > 0$ and $\gamma_i \in (0, 1-1/\eta]$ such that $\gamma_i\eta \geq \gamma_j(\eta+x)$ and $(1-\delta')\ln(1/(1-\gamma_i)) = (1-\delta)\ln(1/(1-\gamma_j))$, then the claim follows. Let $Z = \left(\ln \frac{1}{1-\gamma_j}\right) / \left(\ln \frac{1}{1-\gamma_i}\right)$, then $\delta' = 1-(1-\delta)Z$.

Whenever $\gamma_j \leq 1-1/\eta$, we can always choose $\gamma_i \geq \gamma_j$ such that $\delta' > 0$. The non-trivial case is when $\gamma_j \in (1-1/\eta, 1-1/(\eta+x)]$. In this case, by choosing a large enough η, we can make Z arbitrarily close to 1 and make $\delta' > 0$. In other words, there exists some η_0: for all $\eta \geq \eta_0$, $\delta' > 0$, and by an argument similar to that for claim 2 in Theorem 10, the claim follows. □

6.5.2 A Tri-criteria Approximation

We now consider upper bounds for MINTIME. It is interesting to ask what happens when either the budget overrun or the coverage shortfall is increased. We show that under these conditions, a greedy strategy combined with linear search yields a solution with optimal propagation time. This proves Theorem 12.

Algorithm Greedy-Mintss computes a small seed set S that achieves coverage $\sigma_m(S) = \eta - \epsilon$. Recall that $\sigma_m^R(S)$ denotes the coverage of S under propagation model m within R time steps. It is easy to see that Greedy-Mintss can be adapted to instead compute a seed set that yields coverage $\eta - \epsilon$ within R time steps: we call this algorithm Greedy-Mintss^R.

Given such an algorithm, a simple linear search over $R = 0 \ldots n-1$ yields the bounds specified in Theorem 12, after setting coverage threshold as $\eta - \epsilon$ and the chosen budget threshold as $budg = k(1+\ln(\eta/\epsilon))$. The approximation factors in the theorem follow from Theorem 7 and Lemma 8. These bounds continue to hold if we can only provide estimates for the coverage function (rather than computing it exactly) and also extend to weighted nodes.

We conclude this section by noting that the algorithm above can be naturally adapted to the RAKC problem. The bounds in Theorem 12 apply to RAKC as well, since MINTIME under IC generalizes RAKC.

6.6 Empirical Assessment

We conducted several experiments to assess the value of the approximation algorithms by comparing their quality against that achieved by several well-known heuristics, as well as against the state-of-the-art methods developed for Influence Maximization that we adapt in order to deal with MINTSS and MINTIME. In particular, the goals of experimental evaluation are two-fold. First, we have previously established from theoretical analysis that the Greedy algorithm (Greedy-Mintss for MINTSS and Greedy-Mintss^R for MINTIME) provides the best possible solution that can be obtained in PTIME, which we would like to validate

	NetHEPT	Meme
$\#Nodes$	15233	7418
$\#Arcs$	62794	39170
$Avg.degree$	4.12	5.28
$\#CC$ (strong)	1781	4552
$max\,CC$ (strong)	6794 (44.601%)	2851 (38.434%)
clustering coefficient	0.31372	0.06763

Table 6.1: Networks statistics: number of nodes and directed arcs with non-null probability, average degree, number of (strongly) connected components, size of the largest one, and clustering coefficient.

empirically. Second, we study the gap between the solutions obtained from various heuristics against the Greedy algorithm, the upper bound, in terms of quality.

In what follows we assume the IC propagation model.

Datasets, probabilities and methods used. We use two real-world networks, whose statistics are reported in Table 6.1.

The first network, called NetHEPT, is the same used in [35–37]. It is an academic collaboration network extracted from "High Energy Physics - Theory" section of arXiv[42], with nodes representing authors and edges representing coauthorship. This is clearly an undirected graph, but we consider it directed by taking for each edge the arcs in both the directions. Following [35, 36, 78], we assign probabilities to the arcs in two different ways: *uniform*, where each arc has probability 0.1 (or probability 0.01) and *weighted cascade* (WC), i.e, the probability of an arc (v, u) is $p_{v,u} = 1/d_{in}(u)$, where $d_{in}(\cdot)$ indicates in-degree [78]. Note that WC is a special case of IC where probabilities on arcs are not necessarily uniform.

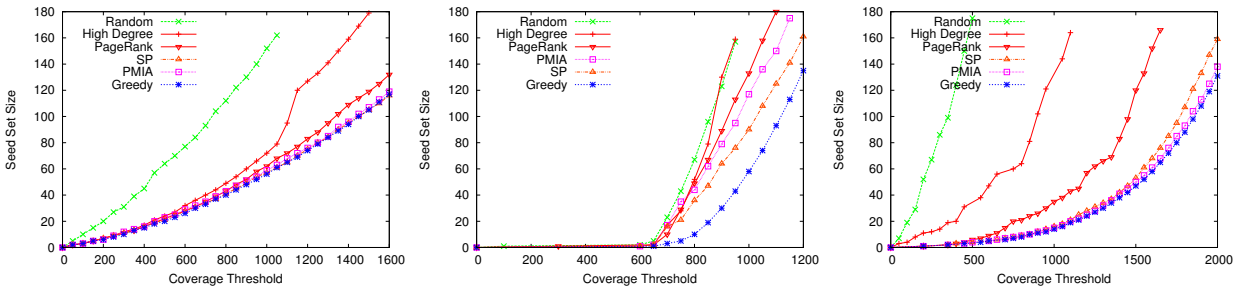
Random	Simply add nodes at random to the seed set, until the stopping condition is met.
High Degree	Greedily add the highest degree node to the seed set, until the stopping condition is met.
Page Rank	The popular index of nodes' importance. We run it with the same setting used in [35].
Sp	The shortest-path based heuristic for the greedy algorithm introduced in [82].
Pmia	The maximum influence arborescence method of [35] with parameter $\theta = 1/320$.
Greedy	Algorithm Greedy-Mintss for MINTSS and Algorithm Greedy-Mintss^R for MINTIME.

Table 6.2: The methods used in our experiments.

The second one, called Meme, is a sample of the social network underlying the Yahoo! Meme[43] microblogging platform. Nodes are users, and directed arcs from a node u to a node v indicate that v *"follows"* u. For this dataset, we also have the log of posts propagations during 2009. We sampled a connected sub-graph of the social network containing the users that participated in the most re-posted items. The availability of posts propagations is significant

[42] http://www.arXiv.org

[43] http://meme.yahoo.com/



(a) NetHEPT- WC (b) NetHEPT- uniform (c) Meme

Figure 6.2: Experimental results on MINTSS.

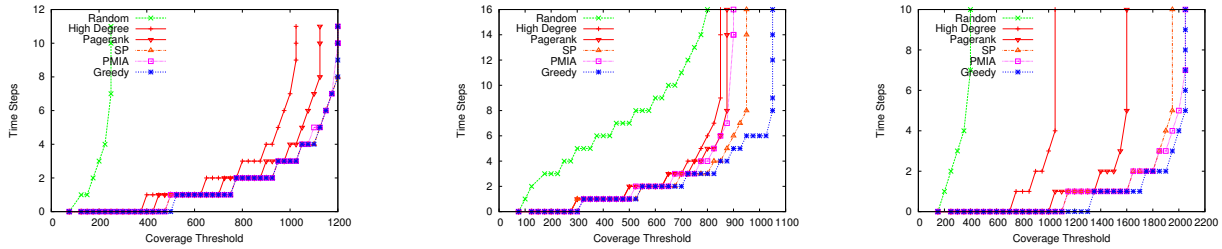
since it allows us to directly estimate actual influence.

In particular, here a propagation is defined based on reposts: a user posts a meme, and if other users like it, they repost it, thus creating cascades. For each meme m and for each user u, we know exactly from which other user she reposted, that is we have a relation $repost(u, v, m, t)$ where t is the time at which the repost occurs, and v is the user from which the information flowed to user u. The maximum likelihood estimator of the probability of influence corresponding to an arc is $p_{v,u} = M_{v2u}/M_{vu}$ where M_{vu} denotes the number of memes that v posted before u, and M_{v2u} denotes the number of memes m such that $repost(u, v, m, t)$.

For the sake of comparison, we adapt the state-of-the-art methods developed for Influence Maximization (also see Section 6.3) to deal with MINTSS and MINTIME. For most of the techniques the adaptation is straightforward. The methods that we use in the experimentation are succinctly summarized in Table 6.2. It is noteworthy that PMIA is one of the state-of-the-art heuristic algorithms proposed for Influence Maximization under the IC model by [35]. In all our experiments, we run 10,000 Monte Carlo simulations for estimating coverage.

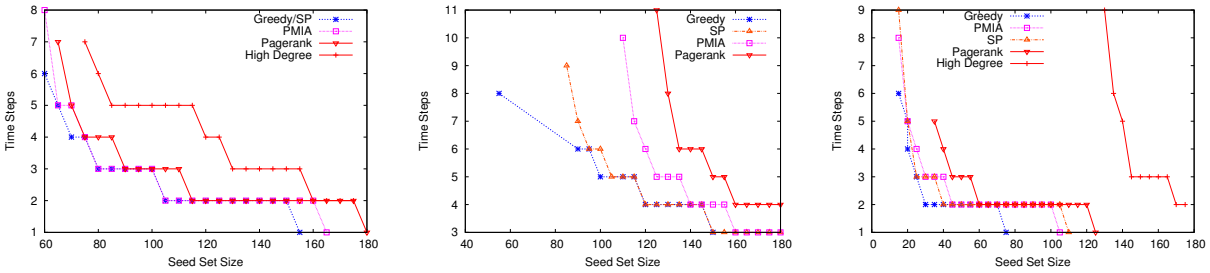
MINTSS - Our experimental results on the MINTSS problem are reported in Figure 6.2. In each of the three plots, we report, for a given coverage threshold (x-axis), the minimum size of a seed-set (budget, reported on y-axis) achieving such coverage. As Greedy provides the upper bound on the quality that can be achieved in PTIME, in all the experiments it outperforms the other methods, with Random and High Degree consistently performing the worst.

We analyzed the probability distributions of the various data sets we experimented with. At one extreme is the model with uniformly low probabilities (0.01). In Meme, about 80% of the probabilities are ≤ 0.05. In NetHEPT WC, on the other hand, approximately 83% of the probabilities are ≥ 0.05 and about 66% of the probabilities are ≥ 0.1. However, the combination of a power law distribution of node degrees in NetHEPT together with assignment of low probabilities for high degree nodes (since it's the reciprocal of in-degree) has the effect of rendering central nodes act as poor influence spreaders. And the arcs with high influence probability are precisely those that are incident to nodes with a very low degree. This makes for a low influence graph overall, i.e., propagation of influence is limited. Finally, at the other extreme is the model with uniformly high probabilities (0.1) which corresponds to a high



(a) NetHEPT- WC, Budget=75 (b) NetHEPT- Uniform, Budget=75 (c) Meme, Budget=150

Figure 6.3: Experimental results on MINTIME with fixed budget.



(a) NetHEPT- WC, η=1000 (b) NetHEPT- Uniform, η=1000 (c) Meme, η=1000

Figure 6.4: Experimental results on MINTIME with fixed Coverage Threshold.

influence graph.

We tested uniformly low probabilities (0.01), and we observed that with such low probabilities, there is limited propagation happening: for instance, in order to achieve a coverage of 150, even the best method requires more than 100 seeds. This forces the quality of all algorithms to look similar.

On data sets where there is a non-uniform mix of low and high probabilities, but the probabilities being predominantly low, as well as on data sets corresponding to low influence graphs, the PMIA method of [35] and the SP method of [82], originally developed as efficient heuristics for the INFLUENCE MAXIMIZATION problem, when adapted to the MINTSS problem, continue to provide a good approximation of the results achieved by the GREEDY algorithm (Figure 6.2(a), (c)). In these situations, the Random and HighDegree heuristics provide seed sets much larger than GREEDY. In NetHEPT WC (Figure 6.2(a)), PageRank has a performance that is close to the Greedy solution, while in Meme(Figure 6.2(c)), the seed set generated by PageRank is much larger than Greedy. In data sets with uniformly high probabilities (0.1), the gap between between GREEDY and other heuristics is substantial (Figure 6.2(b)). GREEDY can achieve a target coverage $\eta = 750$, with just 5 seeds, while PMIA and SP need 35 and 21 seeds respectively; similarly GREEDY can achieve a target coverage $\eta = 1000$, with just 58 seeds, while PMIA and SP need 117 and 90 seeds respectively. It is worth noting that Random, HighDegree, and the PageRank heuristic all generate seed sets much larger than Greedy on this data set. To sum, the gap between the sizes of the seed sets obtained by the heuristics one the one hand and the Greedy algorithm on the other, varies depending on the

influence probabilities on the edges. In general, on graphs with high influence, the gap can be substantial.

MINTIME - Our experimental results on the MINTIME problem are reported in Figures 6.3 and 6.4. In Figure 6.3, we report, for a coverage threshold given on the x-axis, and a fixed budget (75 for NetHEPT, 150 for Meme), the minimum time steps needed to achieve such coverage with the given budget (y-axis). As expected, Greedy outperforms all the heuristics. All the plots show that after a certain time, there is no further gain in the coverage, indicating the influence decays over time. Figure 6.3(a) compares the various heuristics with the Greedy on the NetHEPT dataset under WC model. On this data set, Pmia, Sp and Greedy exhibit comparable performance. The Pagerank heuristic comes close to them.

Figure 6.3(b) shows the results for the NetHEPT dataset under IC model with uniform probability 0.1. Here, Greedy outperforms all the other heuristics. For instance, when coverage threshold η is 900 and budget is 75, Greedy achieves the coverage in 5 time steps, and Sp in 6 time steps, Pmia in 14 time steps. Random, High Degree and Pagerank fail to find a solution. Similarly, when coverage threshold is 1000 and budget is 75, Greedy achieves the coverage in 6 steps whereas all other heuristics fail to find a solution with this coverage.

Finally, Figure 6.3(c) shows the results on Meme dataset. As we increase the target coverage, the other heuristics fail to give a solution, one by one. Beyond $\eta = 1600$, all but Sp, and Pmia fail and beyond $\eta = 2000$, all but Pmia fail. On this data set, Pmia provides a good approximation to the performance of Greedy.

In Figure 6.4, we fix the coverage threshold ($\eta = 1000$ for all the plots). The plots show the minimum time steps needed to achieve the coverage w.r.t. different seed set sizes (budget). In all the cases, Random fails to find a solution and hence is not shown in the plots. The performance of the High Degree algorithm is poor as well and it fails to find a solution in case of NetHEPT with uniform probabilities 0.1. As expected, Greedy outperforms all the heuristics and provides us the lower bound on time needed to achieve the required coverage with a given budget.

Overall, we notice that the performance quality of all other heuristics compared to Greedy follows a similar pattern to that observed in case of MINTSS: as the graph changes from a low influence graph to a high influence graph, the heuristics' performance drops substantially compared to Greedy.

Another key takeaway from the MINTIME plots is the following. *For a given budget, as observed above, the choice of the seed set plays a key role in determining whether a given coverage threshold can be reached or not, no matter how much time we allow for the influence to propagate. Even if the given coverage threshold is achieved, the choice of the seed set can make a big difference to the number of time steps in which the coverage threshold is reached.* Often, for a given budget, relaxing the coverage threshold can dramatically change the propagation time. E.g., In Figure 6.3(a) (budget fixed to 75), while Greedy takes 8 time steps to achieve a coverage of 1200, when we relax the threshold to 1100, the propagation time decreases by 50%, that is, to just 4 time steps. A similar phenomenon is observed when the budget is boosted w.r.t. a fixed coverage threshold. For instance, in Figure 6.4(c), while using 15 seeds, Greedy takes 6 time steps to achieve a coverage of 1000, it achieves the same

coverage by 30 seeds in 33% of the time, that is, in 2 time steps. These findings further highlight the importance of the MINTIME problem.

6.7 Conclusions

In this chapter, we study two optimization problems in social influence propagation: MINTSS and MINTIME. We present a bicriteria approximation for MINTSS which delivers a seed set larger than the optimal seed set by a logarithmic factor $(1+\ln(\eta/\epsilon))$, that achieves a coverage of $\eta - \epsilon$, which falls short of the coverage threshold by ϵ. We also show a generic tightness result that indicates improving the above approximation factor is likely to be hard.

Turning to MINTIME, we give a greedy algorithm that provides a tricriteria approximation when allowed a budget overrun by a factor of $(1+\ln(\eta/\epsilon))$ and a coverage shortfall by ϵ, and achieves the optimal propagation time under these conditions. We also provide hardness results for this problem. We conduct experiments on two real-world networks to compare the quality of various popular heuristics proposed in a different context (with necessary adaptations) with that the greedy approximation algorithms. Our results show that the greedy algorithms outperform the other methods in all the settings (as expected) but depending on the characteristics of the data, some of the heuristics perform competitively. These include the recently proposed heuristics Pmia [35] and Sp [82] which we adapted to MINTSS and MINTIME.

Several questions remain open, including proving optimal approximation bounds for MINTSS and MINTIME, as well as complexity results for these problems under other propagation models.

Chapter 7

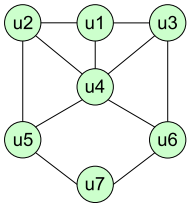
A Pattern Mining Framework to Discover Community Leaders

We introduce a novel frequent pattern mining approach to discover leaders and tribes in social networks. In particular, we consider social networks where users perform actions. Actions may be as simple as tagging resources (urls) as in del.icio.us, rating songs as in Yahoo! Music, or movies as in Yahoo! Movies, or users buying gadgets such as cameras, handhelds, etc. and blogging a review on the gadgets. The assumption is that actions performed by a user can be seen by their network friends. Users seeing their friends' actions are sometimes tempted to perform those actions. We are interested in the problem of studying the propagation of such "influence", and on this basis, identifying which users are leaders when it comes to setting the trend for performing various actions. We consider alternative definitions of leaders based on frequent patterns and develop algorithms for their efficient discovery. Our definitions are based on observing the way influence propagates in a time window, as the window is moved in time. Given a social graph and a table of user actions, our algorithms can discover leaders of various flavors by making one pass over the actions table. We run detailed experiments to evaluate the utility and scalability of our algorithms on real-life data. The results of our experiments confirm on the one hand, the efficiency of the proposed algorithm, and on the other hand, the effectiveness and relevance of the overall framework. To the best of our knowledge, this the first frequent pattern based approach to social network mining.

This chapter is based on our CIKM 2008 paper [55]. This study has been done in collaboration with Francesco Bonchi and Laks V. S. Lakshmanan. We also built a demo "Gurumine" using the techniques proposed in this chapter. It appeared in ICDE 2009 [62]. For brevity, we do not explain Gurumine in this dissertation and focus on core concepts that we proposed in [55].

7.1 Introduction

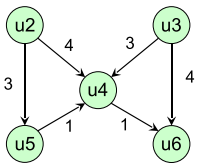
Consider a social network where users perform various actions. As an example, in del.icio.us (`http://del.icio.us/`), users bookmark urls and annotate them with various tags. Here, tagging a url with a tag such as *Ocean Kayaking* is an action. As another example, a user in Yahoo! Movies (`http://movies.yahoo.com/`) may decide to rate the movie *There will be blood*, which is also an action. As a third example, a user in facebook (`http://www.facebook.com/`) may decide to buy a new camera and decide to write a review on it in her personal blog. In each of these examples, users may choose to let their network friends see their actions. Seeing actions performed by their friends may make users curious and may sometimes tempt some



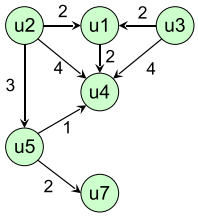
(a)

User	Action	Time
$u3$	b	13
$u2$	b	13
$u2$	a	15
$u1$	b	15
$u3$	a	16
$u5$	b	16
$u4$	b	17
$u5$	a	18
$u7$	b	18
$u4$	a	19
$u6$	a	20

(b)



(c)



(d)

Figure 7.1: (a) Example social graph; (b) A log of actions; (c) Propagation of action a and (d) of action b.

fraction of the users to perform those actions themselves, some of the time. Informally, we can think of this as an influence (for performing certain actions) propagating from users to their network friends, potentially recursively. If such influence patterns repeat with some statistical significance, that can be of interest to companies, say for targeted advertising. E.g., if we know that there are a small number of "leaders" who set the trend for various actions, then targeting them for adoption of new products or technology could be profitable to the companies.

Figure 7.1(a)-(b) shows an example of a social graph and a log of actions performed by users. By linking the actions log in Figure 7.1(b) to the social graph in Figure 7.1(a), we can trace the propagation of influence for performing actions. The result, for actions a and b, is shown in Figure 7.1(c)-(d). Observe that the graphs in Figure 7.1(c)-(d) are directed and labelled even though the original social graph is undirected and unlabelled. Direction is dictated by the order in which actions were performed. If x performed action a before y and there is a social tie between x and y in Figure 7.1(a), then Figure 7.1(c) contains a directed edge from x to y; while the label on the edge is given by the time elapsed between the two actions.

Given a social graph together with a log of user actions, we can determine the propagation of influence for performing each of the actions. How can we leverage this for determining who are the leaders in setting the trend for performing actions? It turns out there are a number of options for defining leadership. First off, intuitively, we want to think of user u as a *leader* w.r.t. an action a provided u performed a and within a chosen time bound after u performed a, a sufficient number of other users performed a. Furthermore, we require that these other users are reachable from u thus capturing the role social ties may have played. Finally, we may want to extract leadership w.r.t. performing actions in general or performing a class of actions. For example, in Figure 7.1, if we choose the time bound to be 5 units, number of users to be influenced by a leader to be *three*, then user $u2$ is a leader w.r.t. both actions a and b, since users $u4, u5, u6$ are influenced by $u2$ in Figure 7.1(c), while $u1, u4, u5, u7$ are influenced by $u2$ in Figure 7.1(d). Notice that $u3$ is not a leader w.r.t. either action. If user $u5$ had performed b after $u4$, then $u3$ would be regarded a leader w.r.t. b, in addition to $u2$. A stronger notion

of leadership might be based on requiring that w.r.t. each of a class of actions of interest, the set of influenced users must be the same. To distinguish from the notion of leader illustrated above, we refer to this notion as *tribe leader*, meaning the user leads a fixed set of users (tribe) w.r.t. a set of actions. To appreciate the difference between leader and tribe leader, consider that we require *two* users to be influenced by a leader and these two users must be the same for both actions a and b. Then $u2$ is a tribe leader for the set of actions $\{a, b\}$ since users $u4$ and $u5$ are influenced by $u2$ w.r.t. both a and b. Clearly, tribe leaders are leaders but not vice versa.

The notion of leader admits several candidates. However, the influence emanating from some of them may be "subsumed" by others. As an example, in Figure 7.1(c), both $u2$ and $u5$ are leaders w.r.t. a when the required number of influenced users is two. However, it is clear that user 5's influence is subsumed by that of user $u2$, as $u5$ is one of those influenced by $u2$. Thus, it is interesting to ask whether we can discover "genuine" leaders, i.e., leaders whose influence is not subsumed by others. Notice that a tribe leader may be genuine or not just as a leader may be genuine or not.

In this chapter, we make the following contributions:

- We formally define the problem of extracting leaders from a database consisting of a social graph plus a table representing the log of user actions. We define various notions of leader, tribe leader, and their confident and genuine variants (Section 7.3).
- We develop efficient algorithms for extracting leaders of various flavors from an input data consisting of a social graph and a table of user actions. Our algorithms make one pass over the actions table. This is significant since we expect the table to be very large (Section 7.4).
- We demonstrate the utility and scalability of our algorithms, via an extensive set of experiments (Section 7.5) on a real-world dataset obtained by joining data coming from Yahoo! Messenger (the social graph) and Yahoo! Movies (the action log, where the action corresponds to rating a movie).

Influence propagation and leadership have been considered in the context of *viral marketing* [42, 78, 109]. There, the model is based on probabilistic causation: there is a probability with which a user will perform an action if his neighbors have performed it. These works focus on the optimization problem of finding the k best people to target an ad to in order to maximize the influence. The framework is entirely based on the social graph together with edge weights representing the probabilities of influence. Our problem setting is considerably different. The input includes not just a graph (which is not edge-weighted) but an actions table which plays a central role in the definition of leaders. Secondly, our focus is on finding all leaders based on the frequent pattern mining approach, which is different from the approach employed in the above works. To the best of our knowledge, our notions of leaders and our algorithms are novel. Related work appears in the next section. We summarize the chapter in Section 7.6 and discuss future work.

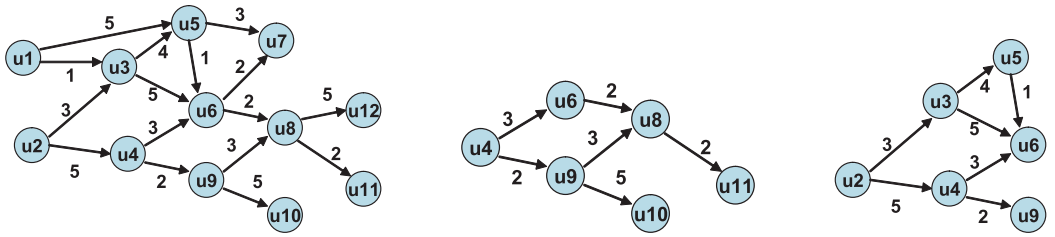


Figure 7.2: The propagation graph of an action $PG(a)$ in fig.(a), $Inf_8(u4, a)$ in fig.(b), $Inf_8(u2, a)$ in fig.(c).

7.2 Related Work

As discussed in the introduction (Section 7.1), even if the general goal of this work and the problem of Influence Maximization is very similar, our problem setting is considerably different, as we have a different input and as we adopt a completely different mining approach, i.e., pattern discovery.

Among the various kinds of patterns that we study, there is also *tribe leaders*, i.e., users that lead a *fixed* set of users for different actions. As a side effect of mining tribe leaders, we also mine tribes which can be considered as small communities. The problem of identifying communities in social networks has been studied extensively in the data mining literature, especially focusing on how to model community formation and evolution in time [11, 12, 45, 69, 70, 87, 130]. Again our problem statement is very different in that we focus on actions log, and in terms of the pattern discovery approach that drives our work. [93] considers a particular form of social network, called friendship-event network, made by the usual social ties, plus a series of events in which the various nodes of the network may be *organizers* or *participants*. Thus also in their context there is a kind of actions log, that is the events organization and participation. Our work is different as we consider actions in a more general sense than the organizer-participant relationship. Also the objective is different: [93] studies the notion of *social capital* based on the actor-organizer friendship relationship and the notion of *benefit*, based on event participation.

7.3 Problem Definition

A *social graph* is an undirected graph $G = (V, E)$ where the nodes are users. There is an undirected edge between users u and v representing a social tie between the users. The tie may be explicit in the form of declared friendship, or it may be derived on the basis of shared interests between users. Our discussion and algorithms are neutral to the source that gives rise to these edges. An *action log* is a relation $Actions(User, Action, Time)$, which contains a tuple (u, a, t) indicating that user u performed action a at time t. It contains such a tuple for every action performed by every user of the system. We will assume that the projection of $Actions$ on the first column is contained in the set of nodes V of the social graph G. In

other words, users in the *Actions* table correspond to nodes of the graph. We let $\mathcal{A}$ denote the universe of actions. We next define propagation of actions. In the following, we assume for ease of exposition that a user performs an action at most once. Our algorithms do not assume this. In Section 7.5, we explain how our algorithms handle the case where users may perform an action multiple times.

Definition 3 (Action propagation)**.** *We say that an action $a \in \mathcal{A}$ propagates from user v_i to v_j iff $(v_i, v_j) \in E$ and $\exists (v_i, a, t_i), (v_j, a, t_j) \in Actions$ with $t_i < t_j$.*

Notice that there must be a social tie between v_i and v_j, both must have performed the action, one strictly before the other. This leads to a natural notion of a propagation graph, defined next.

Definition 4 (Propagation graph)**.** *For each action a, we define a propagation graph $PG(a) = (V(a), E(a))$, defined as follows. $V(a) = \{v \mid \exists t : (v, a, t) \in Actions\}$; there is a directed edge $v_i \xrightarrow{\Delta t} v_j$ whenever a propagates from v_i to v_j, with $(v_i, a, t_i), (v_j, a, t_j) \in Actions$, where $\Delta t = t_j - t_i$.*

The propagation graph consists of users who performed the action, with edges connecting them in the direction of propagation. Observe that the propagation graph is a DAG. Each node can have more than one parent; it is directed, and cycles are impossible due to the time constraint which is the basis for the definition of propagation. Note that the propagation graph can possibly have disconnected components. In other words, the propagation of an action is just a directed instance (a flow) of the undirected graph G, and the log of actions $Actions(User, Action, Time)$ can be seen as a collection of propagations. Influence can propagate transitively. Thus, a definition of leader w.r.t. setting the trend for performing an action should take this into account. To aid in the definition of leaders, we define the notion of an influence graph next. The elapsed time along a (directed) path in a propagation graph is the sum of edge labels along the path. E.g., in Figure 7.1(c), the elapsed time on the path $u2 \rightarrow u4 \rightarrow u6$ is $4 + 1 = 5$.

Definition 5 (User influence graph)**.** *Given action a, a user u, and a maximum propagation time threshold π, we define the influence graph of the user u, denoted $Inf_\pi(u, a)$ as the subgraph $PG(a)$ rooted at u, such that it consists of those nodes of $PG(a)$ which are reachable from u in $PG(a)$ and such that every path from u to any other node in $Inf_\pi(u, a)$ has an elapsed time at most π.*

In Figure 7.2 we show an example of propagation graph $PG(a)$, and two user influence graphs.[44] The propagation time threshold π allows us to set limits on how long after an action is performed, we regard another user performing that action as influenced by the previous user. Given a threshold ψ, we say that user u acted as a *leader w.r.t. action a* whenever the size (in number of nodes) of $Inf_\pi(u, a)$ is at least ψ. The threshold ψ ensures sufficiently many users are influenced by the given user in the context of action a. Note that the two constraints defined by the thresholds π and ψ are conflicting constraints. In fact, π limits the influence

[44] W.r.t. some underlying social graph and action log, not shown.

graph rooted at u to be not grown after a certain elapsed time, while the second constraint requires to have an influence graph larger than ψ.

A user to be identified as a leader must act as such sufficiently often, i.e., for a number of actions larger than a given action threshold σ. This may be seen as the *minimum frequency* constraint in pattern discovery and association rule mining [4].

Problem 5 (Leaders)**.** *Given a set of actions $I \subseteq A$, and three thresholds π, ψ and σ, a user $v \in V$ is a* leader *iff:*

$$\exists S \subseteq I, |S| \geq \sigma \, : \, \forall a \in S \, . \, size(Inf_\pi(v, a)) \geq \psi$$

The intuition behind the above definition is that for sufficiently many actions ($\geq \sigma$) at least ψ other users are influenced to perform the action within π time units from when the given user v performed it. In this case, we regard v as a leader. Notice the definition does not force the set of influenced users for different actions to be the same.

Problem 6 (Tribe-leaders)**.** *Given a set of actions $I \subseteq A$, and three thresholds π, ψ and σ, a user $v \in V$ is a* tribe leader *iff:*

$$\exists S \subseteq I, |S| \geq \sigma, \exists U \subset V, |U| \geq \psi \, : \, \forall a \in S \, . \, U \subseteq Inf_\pi(v, a).$$

The notions leader and tribe leader were illustrated in Section 7.1. Clearly, every tribe leader is a leader, but not vice versa.

7.3.1 Additional constraints

In addition to using an absolute lower bound on the number of actions in which a user acts as a leader, we could apply a "confidence threshold" (similarly to the classical measure of confidence in association rules [4]), requiring that a user is a leader for a large fraction of the actions he performs.

More precisely, for a user $v \in V$, let $P(v) = \{a \in \mathcal{A} \mid v \text{ performed } a\}$ and $L(v) = \{a \in \mathcal{A} \mid v \text{ is a leader w.r.t. } a\}$. Then the *leadership confidence* of v is the ratio $conf(v) = |L(v)|/|P(v)|$. We define:

Problem 7 (Leaders with confidence measure)**.** *Given a set of actions $I \subseteq \mathcal{A}$, and a confidence threshold $0 < \varphi \leq 1$, a user v is said to be a* confidence leader *if it is a leader and $conf(v) \geq \varphi$.*

It may happen that one user acts as a leader according to the given definitions, but in concrete he is always a follower of the same leader. In some sense he benefits from the influence of a true leader, so that he also may seem a leader. To avoid this kind of "fake" leaders, we propose the following.

Given the usual three thresholds π, ψ and σ, for a user v, let $gen(v)$ denote the ratio

$$\frac{|\{a \in L(v) \mid \nexists u \in V : u \; is \; leader \; for \; a \wedge v \in Inf_\pi(u, a)\}|}{|L(v)|}$$

i.e., the fraction of actions led by v for which v's leadership is genuine, in that it is not a consequence of v being present in the influence graph of some other leader w.r.t. that action. We call $gen(v)$ the genuineness score of v. The notion of genuine leaders is defined next.

Problem 8 (Genuine leaders)**.** *Given a set of actions $I \subseteq \mathcal{A}$, and a threshold $0 < \gamma \leq 1$, we define a leader v to be a* genuine leader *provided the genuineness score of v is above the threshold, i.e., $gen(v) \geq \gamma$.*

Both confidence and genuineness constraints can be applied to tribe leaders as well. Note that when we focus on a single action a, the notions of leader for a and tribe leader for a coincide. Thus we use the exact same definitions. So a genuine tribe leader is a tribe leader whose genuineness score is above a threshold, and similarly for confidence. Our goal is to efficiently extract leaders and tribe leaders, possibly with each of the other criteria (confidence and genuineness) or even with both. This leads to eight distinct problems: leaders and tribe leaders without additional constraints, with one of the two, or with both additional constraints.

We provided the basic pattern discovery framework for mining leaders given the social graph and an action log. However, it should be noted that it is not mandatory for the final analyst to provide all the thresholds: she can query our system asking, e.g., the top-k users w.r.t. average number of followers per action given π, or the top-k tribe leaders w.r.t. confidence. In fact, our algorithms by scanning the actions log, produce a summary of the influence for which only the π threshold is needed. The various kinds of leaders then can be selected from this summary by a simple post-processing, as described in the next section.

7.4 Algorithms

Any algorithm for extracting leaders must scan the action log table and traverse the graph. Our assumption is that the graph is large (in the order of hundreds of thousands to millions of nodes) and the action log contains millions of tuples. While the graph is large, given a (maximum propagation) time threshold π, and a specific time interval with length π, we assume the subgraph corresponding to users performing actions within the interval is small. This is confirmed by the real data sets we used for our experiments, where π was set to several weeks. Thus, our focus is on minimizing the scans of the action log table.

Our algorithms are able to extract the patterns (leaders and tribe leaders) in no more than one scan of the action log table Actions. The key insight behind this is as follows. We assume that the action log is stored in chronological order of actions. Given the threshold π, we scan the *Actions* table by means of a window of width π. Thus at any position of the window, a subset of tuples in *Actions* falls in the window. Let W denote the current position of the window. The position W induces a subgraph of the social graph G: $G_W = (V_W, E_W)$, where V_W is the set of users who performed some action within the window W and an edge is present in E_W iff it is present in E and both its endpoints are in V_W. We call G_W the subgraph of G visible from the window.

By sliding the window chronologically backwards we can compute an *influence matrix* $IM_\pi(U, A)$, where U is the number of users and A is the number of actions. The entry $IM_\pi(u, a)$ is the number of users/nodes, influenced by u w.r.t. action a within time π. This number includes u. So, a user u performed action a iff $IM_\pi(u, a) > 0$. Then leaders can be computed from the IM_π easily. When it comes to compute tribe leaders, influence matrix is inadequate: for tribe leaders, we need to check that a fixed set of $\geq \psi$ users were influenced by the leader on sufficiently many actions. To address this problem, we propose the *influence*

cube. The influence cube is a $User \times Action \times User$ cube with cells containing boolean entries: $IC_\pi(u, a, v) = 1$ if user v was influenced by user u w.r.t. action a, w.r.t. an underlying time threshold π. From this, we can determine whether u is a tribe leader, as described later in Section 7.4.3.

7.4.1 Computing Influence Matrix

We now describe the major steps in detail and discuss their efficient implementation. To illustrate our algorithms, we will use the running example shown in Figure 7.3. Figure 7.3(a) shows a schematic action log and shows the window at the bottom-most position, i.e., the most recent actions fall in the window. Figure 7.3(b) shows a possible subgraph of G visible from the window. Nodes in light grey and their incident edges are not yet visible.

Algorithm 17 gives the algorithm at a high level. The steps are explained in detail next. We start by positioning the window at the bottom of the action log, thus seeing only most recent actions (step 1). We visit the nodes of G_W by exploring G and computing necessary state at the visited nodes, e.g., the nodes influenced by a given node (steps 2-3). We move the window backward in time. Clearly, as the window moves existing action tuples may fall off the window (view) and other tuples may enter the window. This in turn causes parts of the visible graph G_W to disappear and new parts to appear. Our algorithms make use of two key operations – *update* and *propagate*. For simplicity of exposition, let us focus on one action a, although our algorithms simultaneously keep track of the influence propagation information for several actions. When a node of G_W disappears as the window moves, we need to update the state of existing nodes. When a new node appears in the window, we need to propagate the state from existing nodes to the new node so that its state can be computed efficiently (steps 6-8).

By repeated application of update and propagate as the window is moved backward in time, we can compute the *influence matrix* $IM_\pi(U, A)$.

Algorithm 17 Compute Influence Matrix

Input: Graph G; Action log *Actions*; Threshold π.
Output: Influence matrix $IM_\pi(U, A)$.

1: Position a window of size π at the end of table *Actions*.
2: Discover the visible subgraph G_W of G on the fly from the tuples in the window.
3: Compute the state of every node by starting from the most recent tuples and working backward up the graph.
4: Fill in the cells $IM_\pi(u, a)$ whenever we have the state for node u w.r.t. action a computed.
5: **while** top of *Actions* not reached **do**
6: Move the window from the most recent tuple in the window to the next earlier tuple.
7: For every tuple that drops off the window, **update** the state of every other node.
8: For every new tuple that appears in the window, compute the state of the corresponding node by **propagating** the state from its children in the visible graph.
9: Update the IM_π entries as needed.
10: **end while**

Next we describe the update and propagate procedures in detail. We start with the representation of node (user) state. We use a bit vector to track which users are influenced by

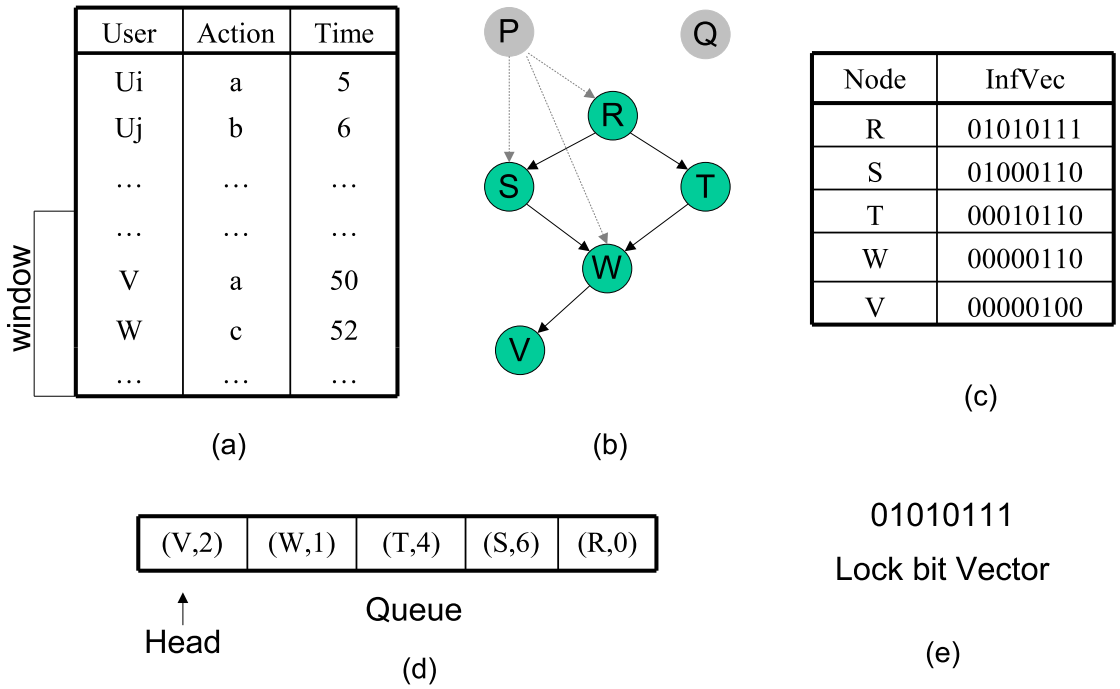


Figure 7.3: (a) Action log; (b) Propagation graph visible in current window for a action; (c) Influence Vector for the current nodes; (d) Queue for the current action; (e) Lock Bit Vector for the current action.

a given user. Since the graph is huge and only a relatively small number of nodes are visible from any window, we would like to optimize the number of bits. However, as the window moves, the bits need to be *dynamically allocated*. Moreover, when we adapt the algorithm for computing tribe leaders, we would want to know for any user and action, not just the number of users influenced but *which ones were influenced*. We solve this problem of dynamic allocation of bits for user state, by maintaining a *Lock Map* for each position of the window. The lock map keeps track of which bit is allocated to which user in the current window. As shown in Figure 7.3(d), we keep the lock map as entries in a queue for efficient manipulation. E.g., it shows user R is assigned the right-most bit (0-th bit), user S is assigned the 6-th bit (from right), etc. To facilitate quick allocation of "free" bits, we keep a *Lock Bit Vector* in which set bits correspond to bits or locks owned by nodes and 0 bits correspond to "free" bit positions. E.g., bit positions $0, 1, 2, 4, 6$ are owned by nodes R, W, V, T, S. When the window moves, the lock map entries and the Lock Bit Vector are kept updated as will be explained shortly. We represent the state of a node as an *influence vector* (IV). There is one influence vector per action performed by the user in the current window. E.g., the influence vector of user S is 01000110. This says the users influenced by S correspond to bit positions $1, 2, 6$. Positions $1, 2$ correspond to users W, V, which are the users influenced by S.

Algorithm 2 describes the procedure *Propagate*. When a new tuple is inserted, we issue a free lock to its associated node (step 1). Searching a free lock is as simple as finding the

Algorithm 18 Propagate

Input: A tuple τ in the form of $< u, a, t >$.

1: Issue the first free lock to u. Let its index be i.
2: Initialize the IV for u with only i-th bit set to 1.
3: **for** each element v of Queue(a) **do**
4: **if** there is an edge from u and v **then**
5: $IV(u) = IV(u)\ OR\ IV(v)$.
6: **end if**
7: **end for**
8: Add the node for u at the tail of Queue(a)

first 0 bit in the Lock Bit Vector (from the right). The IV for the new element u is initialized with its own index set to 1 (step 2). We then traverse the queue and search for a child of u. If there is an edge from u to the current element in the queue, then the IV of u is updated as the bit OR of IV of u and the IV of the current element. This way, we can compute the IV of every new node in the window (steps 3-5). When the time window moves up, a node may drop off the window. In such a case, we need to update the states of the remaining nodes. To do this, we traverse the queue and reset the lock index bit in IV of each element. Then the lock is released by resetting the bit in the Lock Bit Vector. Procedure *Update* describes the algorithm.

Algorithm 19 Update

Input: A tuple $\tau =< u, a, t >$ that has become invisible in the window.

1: i = lock index of u
2: **for** each element v in Queue(a) **do**
3: reset the index i in $IV(v)$.
4: **end for**
5: Delete the element u from the Queue(a).
6: Release the lock.

When the window is at its bottom-most position, we compute the state of every user by bit OR operations. Initialize the IV (for a given action) for every node to be the bit vector where only the bit owned by the node is set. In our running example, the IV of W is obtained as the bit OR of IV of V and the current IV of W. This way, we can compute the IV of every node in the window by bit OR'ing it with the IV of its children. Suppose the window now moves such that node V drops off the window and nodes P and Q enter the window (along with their edges linking to nodes in the window). Notice the new edges can only be directed from nodes that just entered the window to nodes that are present in the window. First, we update the state of nodes R, S, T, W which are present in the old and new positions of the window. This is done by simply zeroing the bit corresponding to V, i.e., bit 2 (from the right) in all their states. Next, by examining the Lock Bit Vector, we allocate the first "free" bit to each new node – bit 3 to P and bit 5 to Q. Then, in order to compute the state of new nodes P and Q, we propagate the states from their children to those nodes. Node Q has no children in the window so its IV is just 100000. For node P, we can compute its IV as a bit OR with its default IV and the current IV of R, S, W.

7.4.2 Computing Leaders

Once the influence matrix IM_π is available, we can compute leaders as follows. Given a user u and thresholds ψ, σ, let $L(u) = \{a \mid IM_\pi(u,a) > \psi\}$. Then u is a leader iff the count $|L(u)| \geq \sigma$. Thus, leaders are computed by scanning the rows of the influence matrix. By the same token, confidence leaders are evaluated as follows. Given a confidence threshold φ, a user is a confidence leader iff $|\{a \mid IM_\pi(u,a) > \psi|/|\{a \mid IM_\pi(u,a) > 0\}| \geq \varphi$. The denominator corresponds to the number of actions performed by u. Recall, $IM_\pi(u,a)$ keeps track of the number of users influenced by u, *including* u, for performing action a within the window π.

The genuineness score is computed by maintaining a $fake$ counter for every user: $fake(u)$ is incremented whenever u is found to be a leader for some action a and in the influence graph $Inf_\pi(v,a)$ of some other leader w.r.t. a. Then the numerator in the definition of $gen(u)$ is simply $|L(u)| - fake(u)$.

7.4.3 Computing Tribe Leaders

As discussed previously influence matrix is inadequate for computing tribe leaders, we need the *influence cube*, i.e. a $User \times Action \times User$ cube with cells containing boolean entries: $IC_\pi(u,a,v) = 1$ if user v was influenced by user u w.r.t. action a, w.r.t. an underlying time threshold π. From this, we can determine whether u is a tribe leader by means of frequent itemsets mining. In fact, we can view each array $IC_\pi(u,a,*)$ (where '$*$' denotes "don't care") as a transaction of items where items correspond to candidate influenced users. More precisely, for a given user u, we have a transaction corresponding to every action. Then we can see a tribe as an itemset, and finding tribe leaders as finding frequent itemsets larger than a given threshold ψ on the minimum acceptable tribe size, where we ensure the itemset does not include u when we are trying to determine if u is a tribe leader.

Lemma 6. *A user u is a tribe leader iff there is at least one itemset of size ψ (not containing u) that has a frequency of σ or more in the transaction database $IC_\pi(u,*,*)$.*

In our implementation, we do not explicitly compute the influence cube. The reason is that real data sets tend to be quite sparse w.r.t. users performing actions. This is a common phenomenon found in del.icio.us (for tagging actions) and Yahoo! Movies (w.r.t. rating movies) and similar social collaborative sites. Thus, a direct implementation of the cube will be inefficient w.r.t. space. We make the observation that when Algorithm 17 computes the influence matrix, it does so using influence vectors. Recall that for each user and each action we have an influence vector, for every position of the time window. Also, for every position, we have a lock map which "decodes" the bit positions in the influence vectors into corresponding node id's.

Consider a user u and action a. Let t be the time u performed a. Consider the position of the time window when its beginning corresponds to time t (and end to $t + \pi$). The influence vector of u w.r.t. a for this position tells us exactly which are the various bit positions corresponding to users influenced by user u on action a. Thus, the *combination of influence vector of u for a for window position t and the lock map* provides a compact representation of the transaction, i.e., row $IC_\pi(u,a,*)$ in the influence cube.

We then compute frequent itemsets using a special algorithm optimized for our needs. Notice that we are *not* interested in all frequent itemsets, but only in those ones whose size is ψ or more. This problem has been deeply studied in the literature [21, 25] and a large variety of optimizations have been developed in this context. Here we adopt the *ExAMiner* implementation [21]. The idea is that a transaction whose size is $< \psi$ can be discarded since none of its subsets will have size $\geq \psi$ either, and therefore the transaction cannot support any valid itemsets. This data reduction, in turn reduces the support of other frequent and invalid itemsets thus reducing the search space and creating new room for removal of other transactions. This has a recursive rippling effect that reduces the computation considerably. The overall approach for computing tribe leaders is summarized in Algorithm 20.

Algorithm 20 Tribe Leaders

Input: Graph G; Action log *Actions*; Thresholds π, ψ, σ.

1: Position a time window of size π at the end of table *Actions* and move it backward in time just as in Algorithm 17.
2: Compute the influence vector for every node and every action as before.
3: When the IV of user u w.r.t. action a for window position t is available, where u performed a at t, update the influence cube row $IC_\pi(u, a, *)$ using the IV and the current lock map.
4: When the influence cube is computed, compute σ-frequent itemsets of size $\geq \psi$ (not containing u) by means of *ExAMiner* [21].
5: Determine the tribe leaders and tribe led by them using the frequent itemsets computed above, using Lemma 6.

7.4.4 Complexity Analysis

Suppose the maximum number of nodes visible in any position of the time window is n. We expect n to be a small fraction of the total number of nodes in the graph G. At any stage, we need n bits to be used for both the Lock Bit Vector and for each of the influence vectors. The key atomic operations in our algorithm are bit operations (AND and OR) and entering or removing a lock map entry into the queue. Each update operation might involve the deletion of $O(n)$ nodes from the queue. Updating the Lock Bit Vector as nodes get deleted costs at most $O(n^2)$ *bit level operations*.[45] Similarly, when a node gets deleted, updating the influence vector of other nodes can cost $O(n)$ bit AND operations, which corresponds to $O(n^2)$ bit level operations. Suppose on an average when the window moves k nodes drop off the window and k nodes enter the window, where $k < n$ and k is $O(n)$. Then the worst case cost for updating all existing nodes in the window when k nodes drop off is $O(kn^2)$. Let T be the number of tuples in the *Actions* table. Then the number of distinct positions of the window is $O(T/k)$. Let A be maximum number of actions performed by a user. Since we need to maintain influence vectors separately for each action, the total number of bit level operations for update is $O(TAn^2)$.

By a similar analysis, we can show that propagate, which involves the atomic steps of finding a "free" bit in the Lock Bit Vector, entering a node in the queue, finding children of a

[45] As opposed to word level.

node among nodes in the window, and modifying the influence vector of a node by bit OR'ing with influence vectors of its children cost $O(TAn^2)$ bit level operations. A node is inserted into the queue or deleted from the queue at most once per tuple in the action log table. We thus have:

Lemma 7. *The influence matrix can be computed using $O(TAn^2)$ bit level operations, and using $O(T)$ insert and delete operations on the queue. Here, T, A, n are parameters as explained above.*

The complexity of computing tribe leaders is determined by: (a) computing the influence cube and (b) finding the existence of frequent itemsets. Influence cube can be computed with the same number of bit operations as influence matrix (Lemma 7). Finding frequent itemsets is well-known to be of exponential complexity. But it's a thoroughly studied problem and the use of several algorithmic tricks makes it quite efficient in practice.

7.5 Experimental Evaluation

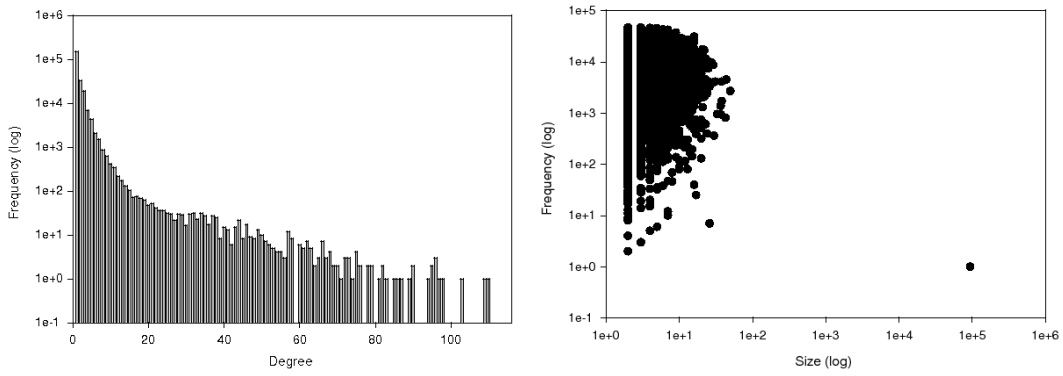
In this section, we describe the comprehensive empirical evaluation we conducted in order to assess both the efficiency of our algorithms and the relevance and usefulness of the overall proposal. All the experiments are performed on a 3 GHz Intel Pentium 4 CPU with 2 Gb main memory, running Suse Linux 10.1 with kernel 2.6.16.54. The algorithms were implemented in C++ with STL and compiled with gnu g++ compiler version 4.1.0. We also exploited the bitstring handling library proposed in [5].

7.5.1 Dataset used

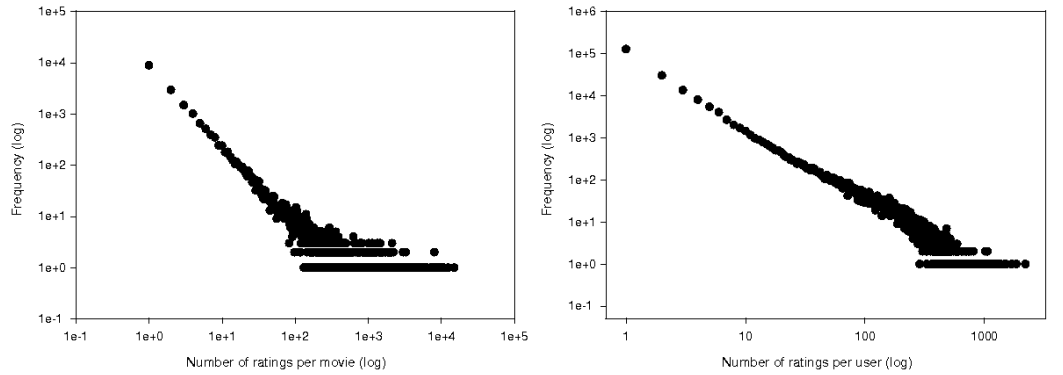
In our problem setting, we need both the social network and the actions database. Using the Yahoo! Movies data we have the actions (ratings of movies) but we do not have any social network information. Thus for producing the social network we "crossed-over" the Yahoo! Movies ratings with a subgraph of the Yahoo! Instant Messenger graph, as described in the following.

For the sake of privacy, all the user data in this chapter were provided to us after anonymization. We started from a subgraph of the Yahoo! Instant Messenger friends-relationship graph, containing the most active users (approx. 110 M nodes), and through common user identifiers, we projected this graph on the subset of users that have also rated a movie in Yahoo! Movies (or more precisely, that have a rating in the set of 21 million ratings provided to us). We then removed isolated nodes, i.e., nodes that were not connected to other nodes, after this projection. Accordingly, we then selected only the ratings belonging to the users present in the resulting subgraph. This data preparation made the graph shrink to a smaller graph containing approximately 217,494 nodes, and 212,373 edges. Accordingly the Yahoo! Movies data reduced from more than 21 millions ratings, to 1.8 millions ratings.

In Figure 7.4 we report some statistics about the dataset we generated. The social graph is quite disconnected: it contains 46,650 connected components, one of which has 94,032 nodes (see the bottom-right corner of Figure 7.4(b)), while all the others have less than 50 nodes.



(a) `nodes degree` (b) `connected components size`



(c) `ratings per movie` (d) `ratings per user`

Figure 7.4: The dataset used in the experimentation: (a) nodes degree distribution, (b) distribution of the size of the connected components, (c) distribution of ratings per movie, and (d) distribution of ratings per user.

In Figure 7.4(a) the distribution of nodes degree is reported. It should be noted that our graph exhibits a quite low density w.r.t. what is expected from a social network graph: this is natural as our graph is not a social network in its whole, but it is just a projection of a social network on a subset of its nodes, i.e., the set of users which performed actions.

Moreover, due to the sparsity of the Yahoo! graph that we constructed, in the following, we report experiments conducted not at the granularity of the movie, but going a step up in the hierarchy, namely at the granularity of *genres*. When we roll up to the level of genre, we have an average of 7.62 actions per user, and an average of 46093 ratings per genre. Note that when we go up in the granularity we have to face the following situation: the same user may execute the same actions more than once at different timestamps (e.g., rating different movies corresponding to the same genre). This situation is not only practically relevant but it also theoretically interesting as also discussed later in Section 7.6. When a user performs an action multiple times, in principle we may use any occurrence of action as a basis for defining propagation of influence. In the experiments we used the first occurrence of a given action,

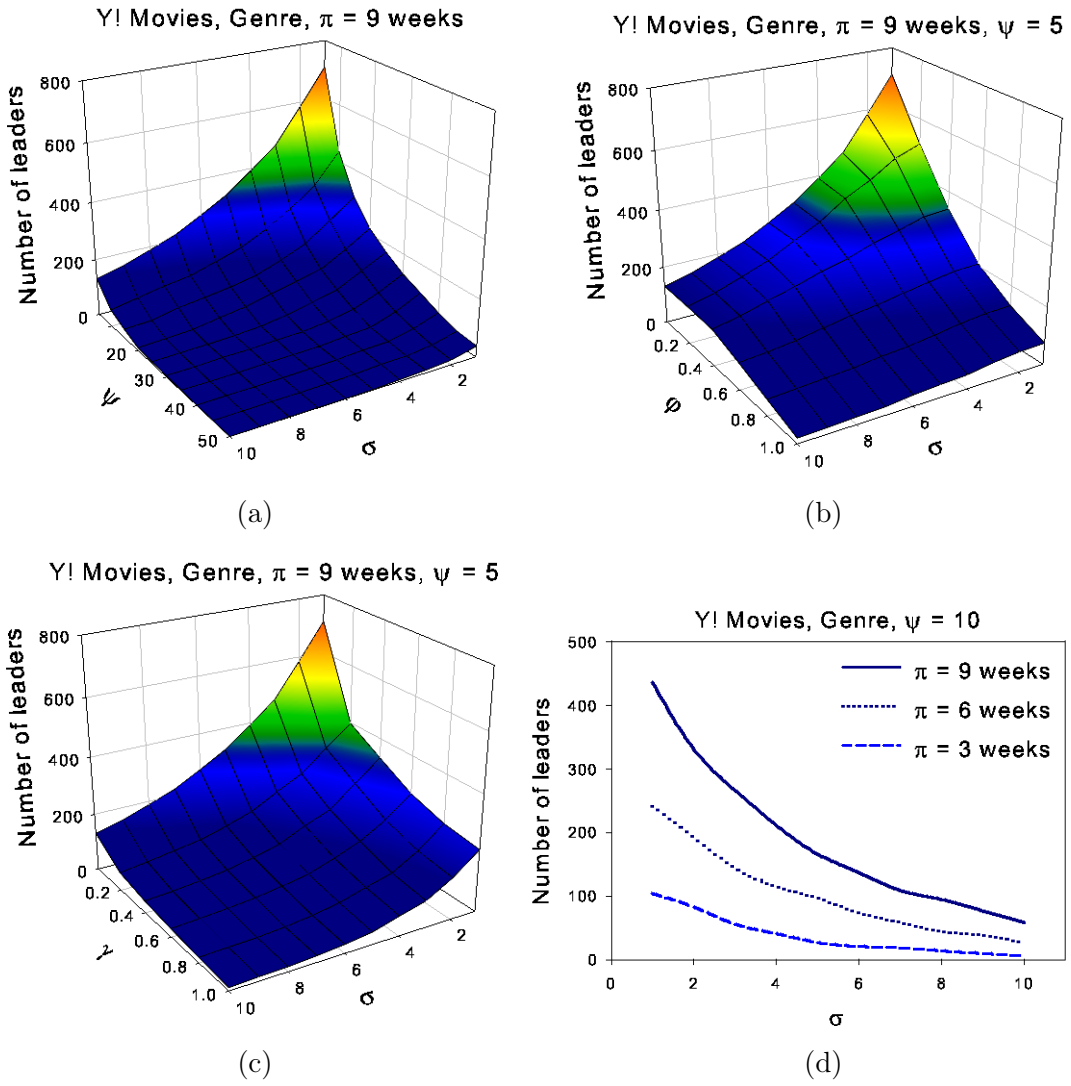


Figure 7.5: (a) Number of leaders found on Yahoo! Movies dataset, with $\pi = 9$ weeks, for $\sigma \in [1, 10]$ and $\psi \in [5, 50]$; (b) number of confidence leaders with $\psi = 5$ and varying confidence threshold; (c) number of genuine leaders with $\psi = 5$ and varying genuineness threshold; (d) number of leaders with varying σ and π.

as we believe it is a more reliable indicator of influence propagation. At any rate, measuring influence based on first occurrence is certainly representative of the options available.

7.5.2 Mining Leaders

In Figure 7.5 we report the number of leaders found in the Yahoo! Movies dataset for various combinations of the input parameters. All these graphs correspond to a time window of $\pi =$

9 weeks. Figure 7.5(a) measures the number of leaders as a function of σ and ψ. Recall σ controls the minimum number of actions where leadership is required in order for a user to be declared a leader, while ψ controls the minimum number of influenced users. As expected, as these thresholds go up, the number of leaders drops. E.g., when $\sigma = 5$ and $\psi = 5$, the number of leaders is 323 and it drops to 59 when σ becomes 10 and ψ becomes 10. Figure 7.5(b)-(c) depict the variation in confidence leaders and genuine leaders as a function of the various parameters. They can all be seen to exhibit similar behavior. In Figure 7.5(d), we see how the number of leaders varies as a function of π and σ. When π is increased from 3 to 6 weeks at any given σ, the number of leaders nearly doubles. This happens again when π is raised from 6 to 9 weeks. In Figure 7.6(a) the histogram of genuineness is reported: it shows that the concept of genuineness is almost binary, clearly cutting between leaders that are dominated by other leaders and truly genuine leaders.

7.5.3 Mining Tribe Leaders

In Figure 7.6(b) a comparison between number of leaders and number of tribe leaders with varying ψ is reported, while comparison of number of tribe leaders for three different π is reported in Figure 7.6(c). Obviously for larger ψ and smaller π we extract a lower number of leaders. In Figure 7.6(d), we report the run time for extracting the influence vectors needed to mine tribe leaders. Recall that extracting the influence vector only depends on the time threshold π and not on the other two thresholds σ and ψ. The plot only reports the run time needed for the construction of the influence vectors: after this step, one frequent itemsets extraction is needed for each leader to see if it is also a tribe leader. The run time for this second phase is not reported for two reason: firstly this is based on pre-existing results and it is not contribution of this chapter; secondly, we found this execution time was always very small and negligible w.r.t. the time needed to construct the influence vectors. Note that as expected run time is larger for larger time threshold π, and it is linear or sublinear w.r.t. the size of the actions log (reported on the x axis).

7.5.4 Qualitative evaluation

Table 7.1 reports the top 10 tribe leaders w.r.t. size of their largest tribe. Note that this is not by any means an indicator that these are the *best* tribe leaders. We could have shown the top 10 by genuineness, or by confidence times genuineness, but this way we would have missed the possibility of discussing leaders with very low genuineness or confidence. Thus we choose to inspect those ones with the largest tribes and to see their level of confidence and genuineness. We use two very distant values of ψ, and for $\sigma = 5$, i.e., they have to influence the same tribe for at least 5 different actions. The table reports the leader $user\_id$, the size of its largest tribe, the number of actions in which it was a leader (a standard one not a tribe leader), the confidence value, and the genuineness value.

There are many interesting things worth noting in these two tables. The first thing is that large tribe leaders usually exhibit high confidence. Consider that for $\pi = 3\,weeks$ and $\sigma = 5$ the average confidence recorded among the leaders is approximately 0.5. For genuineness the results are different. Among the top 10 tribe leaders w.r.t. size of their largest tribe, we

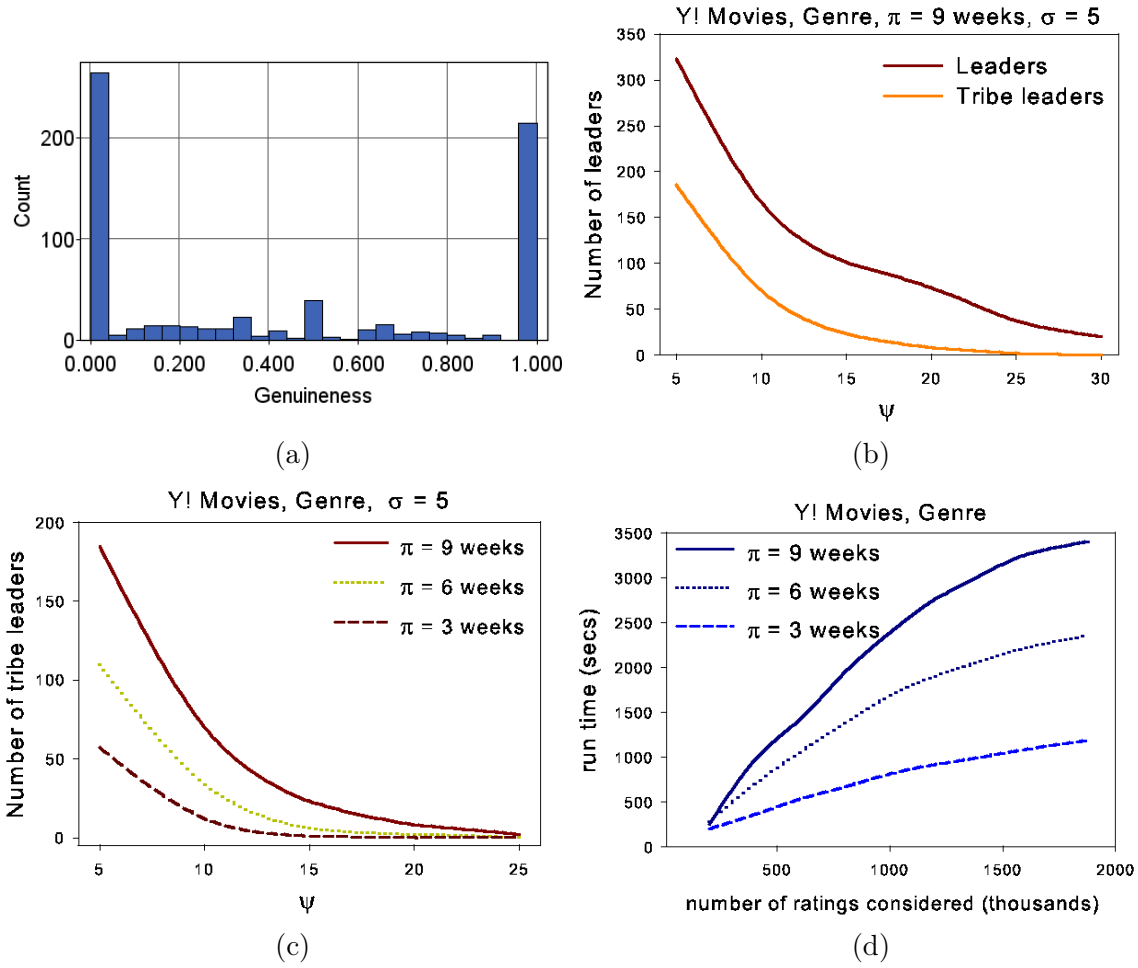


Figure 7.6: (a) Histogram of genuineness of leaders for $\pi = 9$ weeks, (b) comparison between number of leaders and number of tribe leaders with varying ψ, (c) comparison of number of tribe leaders for three different π, (d) and the corresponding run time.

find approximately half showing very high genuineness and half showing very low genuineness indicating that they are dominated by other leaders. In particular users 170467 and 20045 which are present in both tables, user 68241 in Table 7.1(a), and user 144314 in Table 7.1(b). We checked the reason for these low values of genuineness and we found out that these tribe leaders were dominated by other tribe leaders still belonging to the top-10. In particular we found that 206280, 75923 and 198671 dominate 68241, 170467 is dominated by 130514, and 20045 is dominated by 130514 and 22830. Another positive fact worth noting, is that there are three tribe leaders in the intersection of the two top-10 lists.

Finally, let us highlight that being a tribe leader is not simply being a strong leader, but it is an orthogonal concept. We have found many leaders which were acting as leader in many actions and with a large group of followers, but that in the end resulted not to be tribe

$\pi = 3\,weeks$, $\sigma = 5$

rank	*user_id*	*tribesize*	*#actions*	*conf.*	*genu.*
1	198671	15	15	0.58	0.4
2	199726	13	8	1	1
3	130514	12	17	0.89	0.94
4	206280	12	11	0.92	1
5	68241	11	12	0.92	0
6	22830	11	19	0.86	1
7	20045	11	9	0.5	0.22
8	170467	10	7	1	0.29
9	75923	10	9	0.75	1
10	81903	10	11	0.65	0.64

(a)

$\pi = 9\,weeks$, $\sigma = 5$

rank	*user_id*	*tribesize*	*#actions*	*conf.*	*genu.*
1	98711	25	11	0.85	0.73
2	31018	25	9	1	1
3	170467	23	8	1	0
4	20045	22	11	0.61	0.55
5	66331	21	13	0.81	0.92
6	27381	21	16	0.57	0.63
7	85363	20	5	0.63	0.8
8	144314	20	19	0.86	0.1
9	153181	19	22	0.76	0.82
10	206280	19	12	1	0.67

(b)

Table 7.1: Top 10 tribe leaders (those with the largest tribe) for two different values of π.

leaders. On the other hand, we found leaders acting as leaders in a few actions, but that were always followed by the same tribe. Consider for instance user 31018 in Table7.1(b): it acts as a leader in only 9 actions, but in all these 9 actions it is always followed by a tribe of the same 25 users, and it is a very genuine leader (genuineness score = 1). The discussion above confirms the relevance and usefulness of the proposed framework. All the concepts of leaders, confidence, genuineness and tribe leaders make sense per se, and they seems to converge on "strong" leaders. Conjoining these various concepts, we can identify a small set of strong tribe leaders that may be targeted by a viral marketing campaign, in an attempt of maximizing the word-of-mouth effect.

7.6 Conclusions and Discussion

We introduced a novel data mining approach to social networks analysis, based on frequent pattern discovery. While frequent pattern mining has been studied a lot in the last decade and it has found many real-world application scenarios, to the best of our knowledge this is the first proposal of a framework based on frequent pattern mining for discovering leaders in social networks. In particular, we considered social networks where users perform actions such as tagging, buying or rating an item. Given a database consisting of a social graph together with a chronologically ordered log of user actions, we motivated the problem of extracting leaders of various flavors from this database. To that end, we proposed notions of leaders and tribe leaders together with possible additional properties such as confidence and genuineness. We developed efficient algorithms for extracting all types of leaders proposed by us. A notable feature of our algorithms is that all of them make one scan of the action log table. This is accomplished by performing two key operations – *update* and *propagate* – on the fly while sliding a window through the action log table backward in time. Various optimizations were incorporated in the implementation of our algorithms. A key summary data structure employed in our algorithms is called an influence matrix. In the case of tribe leaders, we use an influence cube instead of the matrix.

We conducted extensive experiments on a combination of Yahoo! Instant Messenger dataset forming the social graph with user ids correlated to user ids in Yahoo! Movies dataset. In our experiments, we demonstrated the scalability of our algorithms, and we showed many real interesting patterns of leadership, establishing that the notions of leaders proposed in this chapter are useful and interesting. This work complements work in the area of viral marketing [79, 109].

One potential issue with using pattern discovery for mining leaders is the effect of popular actions. Actions that are very popular may be performed by many users without there really being any influence. This is similar to the famous beer-diaper problem in association rule mining. One way to combat this problem is to discard the most popular actions from the action log and use the rest of the log for mining leaders. Another approach is to associate a "credit" with each action, which is inversely proportional to its popularity and use the credit to determine leadership.

We are currently collecting other data sets where the social network graph comes together with an action log: we plan to test our algorithms on larger and denser data, both for quantitative and qualitative evaluation. In future, it'd be interesting to investigate whether patterns extracted from this approach could be used to determine the parameters required for the viral marketing problem. Furthermore, as discussed in the experiments section, in order to determine interesting patterns, we needed to roll up from the level of individual movies to a more general level of genre, director, or actor. This is a feature that depends on the dataset. Given classification hierarchies on users, actions, and time, it is interesting to ask whether *leadership cube* corresponding to multi-level generalizations over different dimensions can be computed efficiently. Finally, it is worth noting that by mining tribe leaders as a side effect we also mine tribes, that can be considered as small communities. We plan to investigate the relationships between our tribes and the communities structures studied in the literature [45, 69, 87, 130].

Acknowledgments: We acknowledge Aris Anagnostopoulos and Ravi Kumar for the Yahoo!

Instant Messenger graph, Seung-Taek Park and David M. Pennock for the Yahoo! Movie dataset.

Chapter 8

Maximizing Product Adoption in Social Networks

One of the key objectives of viral marketing is to identify a small set of users in a social network, who when convinced to adopt a product will influence others in the network leading to a large number of adoptions in an expected sense. The seminal work of Kempe et al. [78] approaches this as the problem of Influence Maximization. This and other previous papers tacitly assume that a user who is influenced (or, informed) about a product necessarily adopts the product and encourages her friends to adopt it. However, an influenced user may not adopt the product herself, and yet form an opinion based on the experiences of her friends, and share this opinion with others. Furthermore, a user who adopts the product may not like it and hence not encourage her friends to adopt it to the same extent as another user who adopted and liked the product. This is independent of the extent to which those friends are influenced by her. Previous works do not account for these phenomena.

We argue that it is important to distinguish product adoption from influence. We propose a model that factors in a user's experience (or projected experience) with a product. We adapt the classical Linear Threshold (LT) propagation model by defining an objective function that explicitly captures product adoption, as opposed to influence. We show that under our model, adoption maximization is NP-hard and the objective function is monotone and submodular, thus admitting an approximation algorithm. We perform experiments on three real popular social networks and show that our model is able to distinguish between influence and adoption, and predict product adoption much more accurately than the classical LT model.

This chapter is based on our WSDM 2012 paper [17]. This study was performed in collaboration with Smriti Bhagat and Laks V. S. Lakshmanan.

8.1 Introduction

One of the fundamental problems in viral marketing is to identify a small set of individuals in a social network, who when convinced to adopt a product will influence others in the network through a word-of-mouth effect, leading to a large number of adoptions in an expected sense. Previous works such as [78] approached this as the problem of Influence Maximization where, given a social network represented by a directed graph with users as nodes, edges corresponding to social ties, and edge weights capturing influence probabilities, the goal is to find a *seed set* of k users such that by targeting these, the *expected influence spread* (defined as the expected number of influenced users) is maximized. Here, the expected influence spread of a seed set depends on the influence diffusion process which is captured by a model for influence

propagation. While several propagation models exist, two classical models, namely Linear Threshold (LT) and Independent Cascade (IC), have been widely studied in the literature. In this work, we focus on the LT model. In the LT model, a user is either in an inactive or an active state. An active user influences each of its inactive neighbors. The activation of an inactive user depends on the influence probabilities associated with its active neighbors. Each user v picks an activation threshold θ_v uniformly at random from $[0, 1]$. At any time step, if the sum of incoming weights from its active neighbors exceeds the threshold, v becomes active. The process continues until no more activations are possible.

In the bulk of the Influence Maximization literature in data mining, it is assumed that once a user is active (i.e., is influenced), she will automatically and unconditionally adopt the product. In this sense, influence spread is viewed as equivalent to adoption spread (expected number of users who adopt the product). Clearly, product (or technology/innovation) adoption is the main goal in viral marketing, and influence spread is essentially used as a "proxy" for adoption. We argue that it is important to distinguish between influence and adoption, an idea well established in other domains like Sociology and Marketing. Bohlen et al. [19] in 1957 proposed five stages of product adoption in which the adoption stage is considered different than the awareness stage. Kalish [76] characterized the adoption of a new product as consisting of two steps – awareness and adoption – and argued that the awareness information spreads in an epidemic-like manner while the actual adoption depends on other factors such as price and individual's valuation of the product. Given these studies in other traditional domains, the current Influence Maximization work in the data mining community lags behind in modeling such phenomena. Our work is a step towards building a more realistic model of product adoption.

More concretely, previous literature (in data mining) on Influence Maximization makes three assumptions. First, once a user is influenced, she will immediately adopt the product. Second, it is assumed that once a user adopts a product, she encourages her friends to adopt it as well, irrespective of whether or how much she likes the product. In other words, adoption of a product implies that the user likes it regardless of her experience with it. Third, only after a user adopts a product, she shares her opinion on the product and attempts to influence her neighbors. That is, non-adopters either don't have any opinions on the product or such opinions are not visible to others. These assumptions may not hold in general as illustrated by the following examples.

Example 1. *Bob buys an Amazon Kindle and dislikes the product. He posts his experience on his blog and describes the missing features. Kali, a friend of Bob, takes Bob's opinions on technology products seriously. She learns about the Kindle from the post and decides* not *to buy it. Furthermore, she blogs about the product not meeting her expectations and in turn influences the decisions of her friends.* □

Example 2. *Bob watches the movie "The Ring" and likes it. He tells his friend Kali that the movie is great. Kali doesn't get a chance to watch the movie, but tells her friend Charles about the movie. Charles gets influenced and watches it.* □

In Example 1, Bob doesn't like the product and shares his opinion with his friends. Thus, opinions can emerge from a user experience with a product and can propagate from an adopter.

Specifically, if a user does not like the product, their endorsement of the product is unlikely to be strong, a fact that may be noticed by their neighbors. Even when a user is strongly influenced by her neighbor, if the endorsement is weak, the user is less likely to adopt the product. The propagation of information to a user does not always lead to product adoption. There may be many factors that affect a user's adoption decision, including the user's interests, budget and time. Indeed, opinions can propagate from a non-adopter who is active (e.g., Kali in Example 2) and can promote adoption by others. In this example Kali acts as an "information bridge" or a *tattler*, who talks about the product without actually adopting it. How does the presence of such non-adopters who act as information bridges affect the overall adoption? Does the product adoption depend on the opinions, in addition to social influence? These are some of the questions this chapter addresses.

In particular, we argue that the classical propagation models that make the three assumptions above may predict a product adoption quite different from reality, a point we will establish empirically using three real data sets. This motivates re-examining these assumptions and allowing for the following facts: (i) adopters' opinions may strongly affect the degree to which they influence their neighbors' adoption and (ii) non-adopters acting as an information bridge can contribute their own opinions, which too can affect influence propagation as well as adoption. We use user ratings of products as an abstraction of their opinions. Ratings are either provided by the user or can be predicted using collaborative filtering [84]. Therefore, the extent to which a user is influenced by her neighbors (to become active) is a function, not only of the influence probabilities but of the ratings of the neighbors as well. In this chapter, we study an interesting variation of the classical Influence Maximization problem with an aim to *maximize the number of product adoptions* and make the following contributions.

- We study the novel problem of maximizing product adoption, a more natural goal of applications like viral marketing. We present an intuitive model called LT-C, for *LT with Colors*, that captures these intuitions. We show that the problem of product adoption maximization is NP-hard but the objective function, i.e., the expected number of product adoptions, is monotone and submodular; thus the greedy algorithm provides a $(1-1/e-\epsilon)$-approximation to the optimal solution. We also propose methods to learn parameters of our model (Section 8.3).

- We study two types of networks – movies networks (Section 8.4.1) and music networks (Section 8.4.2) using three real world datasets. We found that in almost all instances, the classical LT model significantly over-predicts the spread. Moreover, it predicts the same spread for all the products. We demonstrate that our LT-C model, by incorporating ratings, significantly improves the accuracy of these predictions.

- In all the three datasets, we found that there is a positive correlation between the number of initiators (users who first express opinions among their friend circles) and the final spread, suggesting that the network structure plays a significant role in the spread. We demonstrate that the choice of seed nodes is critical and can make a significant difference to the spread achieved. We also found that largely, the seed sets are different for different products, though some influential users are picked up consistently as seeds irrespective of the product being marketed (Section 8.4).

We review related work in §8.2 and present conclusions in §8.5.

8.2 Related Work

One related problem is that of revenue maximization, studied by Hartline et al. [65]. They offer a family of strategies based on an *influence and exploit* paradigm that work as follows. In the first step, called the *influence* step, the seller gives the item to a chosen set of customers for free. Then in the *exploit* step, the seller visits the remaining buyers in a random sequence and offers each buyer a price that is expected to maximize her revenue. Their model allows the seller to bypass the network structure and visit arbitrary buyers at any time. While revenue maximization, in principle, is closely related to adoption maximization, and arguably should subsume the latter, there are the following key differences with our work. We explicitly model the following phenomena observed in real data sets: first, users may be activated but not necessarily adopt/buy a product and instead may tattle about it; second, opinions shared by tattlers may promote adoption by other users; third, sometimes, tattlers may choose to inhibit adoption by other users by sharing the poor ratings they gave for the product.

Another closely related problem is that Influence Maximization in the presence of negative opinions (see Chen et al. [34]). Their model, called IC-N, builds on the classical IC model by subdividing the active state into two sub-states, positive and negative, while preserving the properties of monotonicity and submodularity. They incorporate the spread of negative opinions with a parameter q that models the quality of the product. The model assumes that the parameter q is the same across all users, an assumption that is not always realistic. Unfortunately, this assumption is essential to their framework, since if the parameter q is allowed to be different for different users (indicating users like the product to differing degrees), their objective function is no longer monotone and submodular, thus, the greedy algorithm fails to provide any approximation guarantee. In our work, we argue that influence propagation depends on the extent to which a user likes the product, in addition to the influence weights among users. Furthermore, we claim that in a network, there exist information bridges, or *tattlers* who propagate the influence without adopting the product themselves. We show that these tattlers are indeed present in real data sets and their presence makes a significant difference to product adoption. A key feature of our model is that not only do we allow the probability of liking a product to be different for different users, we do so while retaining the monotonicity and submodularity of the objective function.

Tang et al. [129] recognize that influence varies with the domain/topic of an item. They present a scalable approach based on MapReduce for learning topic-specific social influence weights from a given social graph and topic distribution. By contrast, in our work, we recognize the reality that user tastes and opinions can be not just topic specific but even item specific and take that into account in our model.

In addition to social influence, *homophily* (or *selection*) has been shown to be a possible cause of users' actions. While some (e.g., [8]) focus on distinguishing the effects of influence and homophily, others (e.g., [39], [88]) showed the feedback effects that both factors have on each other. In particular, Crandall et al. [39] argued that both influence and homophily can be useful in predicting users' future behavior. In our work, we are interested in building

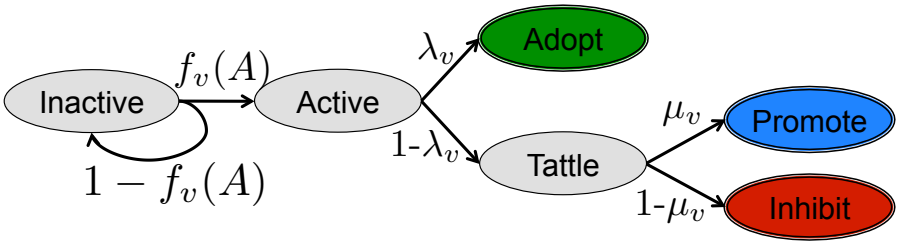


Figure 8.1: LT-C model with colored end states: adopt–green, promote–blue, inhibit–red

a model that predicts the users' actions, in particular, users' adoption of products, without necessarily distinguishing the causes, and use it to develop algorithms that maximize the number of adoptions. Following [76], we argue that while information (or influence) spreads in an epidemic-like manner, actual adoption depends on various other factors. We separate the state of adoption from the state of being influenced (or active) in the LT model. Clearly, for the model to work, it is important that the underlying network structure represent the flow of information, irrespective of whether it corresponds to an explicit social graph (where the links are formed explicitly by users, e.g., friendship links in Facebook), an implicit social graph (where the information flows among users indirectly, e.g., as in recommendation engines like Movielens, Amazon etc.), or a combination of the two.

A recent work by Goyal et al. [57] also focuses on predicting the users' actions, while offering an alternative "data-based" approach that bypasses the propagation models to estimate the spread. The same authors [56] study the problem of learning influence probabilities based on the past propagation traces, paying special attention to the temporal nature of influence probabilities. In other work, Chen et al. [37] have proposed an efficient heuristic for Influence Maximization under the LT model, which is recently improved by [61].

8.3 Proposed Framework

Let $G = (V, E, W)$ be a weighted, directed graph with nodes (users) V and edges (social ties) E, where the influence weights are captured by the function $W : E \rightarrow [0, 1]$. The weight $w_{u,v}$ associated with an edge $(u, v) \in E$ represents the probability of user u influencing user v, such that the sum of incoming edge weights at any node is no greater than 1. That is, $\sum_{u \in N^{in}(v)} w_{u,v} \leq 1$, where $N^{in}(v)$ is the set of in-neighbors of user v. In addition, we assume that a matrix R of Users $\times$ Products is given where each entry $r_{u,i}$ denotes the rating given by user u to the product i. We assume whenever ratings are missing, they are predicted using collaborative filtering. The higher the rating, the more positive the opinion.

Intuitively, a user's decision to adopt a product should not only depend on the influence of neighbors, but also on what they think about the product, as captured by the ratings. Next, we describe our model that we call LT-C model, for *Linear Threshold with Colors*.

8.3.1 LT-C Model

Figure 8.1 gives an overview of our model in the form of a state diagram. An INACTIVE node v has not yet formed an opinion of the new product and is open to being influenced by neighbors. A node is ACTIVE if it is influenced by its neighbors. Let $\mathcal{A}$ be the set of ACTIVE in-neighbors of node v and $f_v(\mathcal{A})$ denote the activation function of v, that is, the total influence on v from nodes in $\mathcal{A}$ s.t. $0 \leq f_v(\mathcal{A}) \leq 1$. Values of $f_v(\mathcal{A})$ close to 1 can be thought of as a collective "recommendation" of ACTIVE neighbors of u strongly favoring the product. On the hand, values of $f_v(\mathcal{A})$ that are close to 0 represent that neighbors of u collectively disapprove the product. As with the LT model, an INACTIVE node v picks an *activation threshold* θ_v uniformly at random from $[0, 1]$ and if $f_v(\mathcal{A}) \geq \theta_v$ then v becomes ACTIVE, else v remains INACTIVE. We instantiate the activation function as follows,

$$f_v(\mathcal{A}) = \frac{\sum_{u \in \mathcal{A}} w_{u,v}(r_{u,i} - r_{\min})}{r_{\max} - r_{\min}} \tag{8.1}$$

where $r_{\max}$ and $r_{\min}$ represent the maximum and minimum ratings in the system, respectively. Intuitively, this definition corresponds to treating, for each edge (u, v) in the graph, the effective influence weight of u on v as $w_{u,v}(r_{u,i} - r_{\min})/(r_{\max} - r_{\min})$, where i is the product being marketed.

Once a node is in the ACTIVE state, it has enough information about the product and has formed an opinion. At this stage, the node can choose to *adopt* the product and rate it, or decide to share its opinion without adopting the product i.e., *tattle*. With some probability λ_v, an activated node v enters the ADOPT state (or adopts) and is colored *green*, and with probability $1 - \lambda_v$, node v enters the TATTLE state (or tattles). Intuitively, the parameter λ_v models the likelihood of a node adopting the product after it has been activated by its neighbors. There are several real world factors that can be modeled into λ_v, such as a user's budget, interests and purchase history. Note that the parameter λ_v is specific to each node v. If the node decides to ADOPT, it provides a rating $r_{v,i}$ for the product i being marketed. In §8.3.4, we propose methods to learn these parameters.

The TATTLE state is an interesting one. It represents a typical gossipmonger who has not experienced the product but is expressing an *acquired* opinion. Such a node is likely to be biased one way or the other. We say a tattler (node in the TATTLE state) can either enter the PROMOTE state (color *blue*) or the INHIBIT state (color *red*) depending on its disposition parameterized by μ_v, specific to v. In other words, v enters the PROMOTE state with probability μ_v and INHIBIT stat with probability $1 - \mu_v$. For instance, consider a user who loves (or hates) the iPhone and owns one already. When the latest model is launched, even though the user is not ready to buy it, the user has some strongly biased opinion of the product which the user shares with his friends. We model this bias in opinion by representing it with a constant rating of $r_{\max}$ for a user in the PROMOTE state and $r_{\min}$ for a user in the INHIBIT state. A node in the PROMOTE or INHIBIT state propagates information but does not directly contribute to the revenue. The node's actions however may stimulate or inhibit adoption by its neighbors, thus proving to be an important information bridge in the network. Finally, we should mention that of the states in Figure 8.1, only the INACTIVE, ADOPT, PROMOTE and INHIBIT states may be observable in a real data set. The intermediate states are used for modeling purposes only.

The dynamics followed by the process of opinion propagation at discrete time steps are as follows. At time $t = 0$, a set S of k nodes is chosen as the *seed set* and are said to be active, and each seed node can choose any of the three states ADOPT, PROMOTE or INHIBIT. At $t > 0$, each node u that activated (ADOPT, PROMOTE or INHIBIT state) at time t, contributes to the activation of its neighbors v where $(u, v) \in E$. Let $\mathcal{A}_t$ denote the neighbors of v that are ACTIVE at time t, then the total influence on node v at time t is $f_v(\mathcal{A}_t)$. If the ACTIVE neighbors push $f_v(\mathcal{A}_t)$ over the threshold θ_v, then v becomes ACTIVE, else it remains INACTIVE. Once in ACTIVE state, the node follows the path to enter one of ADOPT, PROMOTE or INHIBIT states and stays in that state with corresponding probabilities. Also, each ACTIVE node provides a rating of the product that is factored into the activation function. Thus, once a node becomes active, it enters one of these states. A_{t+1} is the set of all nodes that become active as a result of the influence from their neighbors in A_t. The process of node activation stops if at some time step no new nodes can be activated.

8.3.2 Maximizing Product Adoption

We define the *expected spread* (or *spread* for short) of the seed set S as the expected number of ADOPT nodes in the network at the end of the propagation and denote it $\sigma(S)$. We sometimes refer to $\sigma(S)$ as the *coverage* of set S.

Problem 9 (Maximizing Product Adoption)**.** *Given a social graph $G = (V, E, B)$, a parameter k, and a ratings matrix R and parameters $\lambda_v, \mu_v, \forall v \in V$, the problem of maximizing product adoption is to find a seed set S of k nodes such that by activating these nodes, the expected spread under LT-C model is maximized.*

Not surprisingly, the problem is NP-hard. However, as we will show, the spread function is monotone and submodular, so a simple greedy algorithm leads to a $(1-1/e-\epsilon)$-approximation to the optimum, for any $\epsilon > 0$.

Theorem 13. *Problem 9 is NP-hard.*

Proof. Consider the restricted class of instances of the problem where $\forall v \in V, \lambda_v = 1$ and $r_{v,i} = r_{max}$. The problem of maximizing product adoption over this class of instances is equivalent to the classical problem of INFLUENCE MAXIMIZATION under the LT model, which is known to be NP-hard [78]. The theorem follows. □

Theorem 14. *The spread function $\sigma(\cdot)$ under LT-C model is monotone and submodular.*

Proof. It is straightforward to show that $\sigma(\cdot)$ is monotone. To establish submodularity, our proof outline follows that in [78]. The challenge in our case is two-fold: (1) There are four possible observable states of nodes – INACTIVE (gray), ADOPT (green), PROMOTE (blue), and INHIBIT (red) in place of just two; (2) The activation of a node depends on the state of its in-neighbors. The reason is that the influence of a node u on its neighbor v is a combination of the weight $w_{u,v}$ and the rating of u. This rating depends on u's state: it's r_{min} in the red state, r_{max} in the blue state, and a value $r_{u,i}$ obtained from R in the green state. Thus, u's state affects the extent of u's influence on its neighbors.

Borrowing the idea of a live edge model from [78], we define a timed version of the live edge selection process for the purpose of this proof. Start by activating the seed nodes S in G and color them as follows: for any $u \in S$, color u green w.p. λ_u, blue w.p. $(1-\lambda_u)\mu_u$, and red w.p. $(1-\lambda_u)(1-\mu_u)$. At any time $t > 0$, allow each uncolored out-neighbor v of a colored node to choose at most one of its in-neighbors z w.p. $p_{z,v}$ and no in-neighbor w.p. $\sum_{y \in N^{in}(v)} w_{y,v} = 1$. If the chosen in-neighbor is colored, then color v green w.p. λ_v, blue w.p. $(1-\lambda_v)\mu_v$, and red w.p. $(1-\lambda_v)(1-\mu_v)$. Otherwise, don't record the choice. That is, if the chosen in-neighbor is not colored, we don't record the choice of the in-neighbor, but constrain future choices of in-neighbors by v, if any, to be outside the set of nodes colored by time $t-1$. Term each chosen edge i.e., edge to the chosen colored neighbor as a *live edge*. Other edges are blocked/dead. A *live path* is a path made of only live edges. Stop the process when there is no change. This process produces a distribution over possible worlds. Let X be one such possible world. We will show that $\sigma(S) = \sum_X Pr[X]\sigma_X(S)$, where $\sigma_X(S)$ is the number of green nodes reachable from the seed nodes S via live paths in the possible world X. It is easy to see that $\sigma_X(.)$ is monotone and submodular, from which the theorem follows, since a non-negative linear combination of submodular functions is submodular. To show that $\sigma(S) = \sum_X Pr[X]\sigma_X(S)$, we will show that the probability distributions of sets of green/blue/red nodes obtained by running the LT-C model are identical to those obtained from the above process.

We step through the diffusion process according to the LT-C model and determine the probability with which any node v turns green, blue or red. Let each node v pick its threshold θ_v uniformly at random from $[0, 1]$. Let S_t be the set of ACTIVE (colored) nodes at time $t = 0, 1, 2, \ldots$, with $S_0 = S$. The probability with which an uncolored node v will be colored at time $t+1$ is the likelihood that a neighbor u that got colored at time t pushes the total influence on v over the threshold θ_v. We denote this probability of v getting colored at time $t+1$ as ψ_v^{t+1}. Therefore,

$$Pr[v \text{ got colored at time } t+1] = \psi_v^{t+1} = \frac{\sum_{u \in S_t \setminus S_{t-1}} p_{u,v}}{1 - \sum_{u \in S_{t-1}} p_{u,v}}$$

Once a node v is ACTIVE, its color depends solely on the parameters λ_v and μ_v and so we have:

$$\begin{aligned}
&Pr[v \text{ turns green at } t+1] = \lambda_v \cdot \psi_v^{t+1} \\
&Pr[v \text{ turns blue at } t+1] = (1-\lambda_v) \cdot \mu_v \cdot \psi_v^{t+1} \\
&Pr[v \text{ turns red at } t+1] = (1-\lambda_v) \cdot (1-\mu_v) \cdot \psi_v^{t+1}
\end{aligned}$$

Next, we consider the timed version of the live edge selection process defined above and iterate through the corresponding process. Let $S'_0 = S$ be the seed set. Let S'_t, $t \geq 0$, denote the set of nodes that is colored at time t according to this process. Let $v \notin S$ be any node. The probability that it got colored at time $t+1$, denoted by ϕ_v^{t+1}, is the probability with which its chosen in-neighbor u got colored at time t, given that v was not colored before time $t+1$. More precisely,

$$Pr[v \text{ got colored at time } t+1] = \phi_v^{t+1} = \frac{\sum_{u \in S'_t \setminus S'_{t-1}} p_{u,v}}{1 - \sum_{u \in S'_{t-1}} p_{u,v}}$$

Applying induction on time, it is easy to see that the distributions of ACTIVE (i.e., colored) sets of nodes obtained from running the LT-C model, $S_t, t \geq 0$, and the sets of nodes reachable from the seed nodes S by live paths, $S'_t, t \geq 0$, are identical. Given the equivalence between the distributions over S_t and S'_t, it follows that $\phi_v^{t+1} = \psi_v^{t+1}$. Since the color acquired by an ACTIVE node solely depends on the parameters λ_v, μ_v, the distributions of nodes colored green, blue, and red under the LT-C model and under the timed live edge selection process are identical. This was to be shown. □

8.3.3 Choosing Optimal Seed Set

While product adption maximization is NP-hard, as shown by Theorem 14, the spread function $\sigma(S)$ under the LT-C model is monotone and submodular. Thus, we can employ a simple greedy algorithm which repeatedly picks a node with the maximum marginal gain and adds it to the seed set, until the budget k is reached. Furthermore, since the LT model is a special case of the LT-C model (corresponding to $\lambda_v = 1, \forall v$ and $r_{v,i} = r_{\max}, \forall v$), the #P-hardness [37] of computing the spread of a given seed set carries over to our setting. To mitigate this, we employ Monte Carlo simulation for estimating the spread. Finally, we adapt the CELF algorithm of Leskovec et al. [91]. The idea behind CELF is that the marginal gain of a node cannot increase in subsequent iterations (due to submodularity) and thus the spread of seed sets is computed in a lazy forward manner, speeding up the greedy algorithm considerably. Our implementation is based on these ideas. The adaptation of CELF to LT-C model is trivial and we omit the details. It is well known that this approach yields a $(1-1/e-\epsilon)$-approximation to the optimum, for any $\epsilon > 0$.

8.3.4 Learning Model Parameters

Edge Weights. As in [56], we learn influence weights from the past behavior of users. For example, consider a log of ratings in which each tuple is of the form $\langle u, i, t, r \rangle$, saying user u rated an item i at time t with rating r. There are several cases where such information is readily available in the real world. For example, consider a movie recommender system (see §8.4 for the complete case study), then an action can be considered as "user rating a movie". Seeing friends' ratings, one may be influenced to watch the movie and in turn, rate the movie at a later timestamp. Hence, the influence weight on an edge (u, v) is learnt as the fraction of times user v rated an item after u had done so, and normalized over all neighbors x of v such that $\sum_{x \in N^{in}(v)} w_{x,v} = 1$.

Ratings Matrix. We compute the ratings matrix R using collaborative filtering. Several sophisticated collaborative filtering methods have been proposed by the recommender systems community for predicting the rating of a user for a given item or product. Matrix factorization [84] is one such popular method. The input to matrix factorization is a very large sparse matrix of user ratings with a large number of missing values corresponding to users who have not rated a product. The main task is to predict these missing ratings. An assumption made is that some small number of ratings are available for each product. In the context of our problem, we maintain the assumption that the new product which we want to push in the market has been adopted by some small number of early adopters, who have assigned ratings

to the product. This is commonly seen when new products are launched, for instance, technology bloggers get a sneak-peak of new products at tech-media events and they blog about those products. Often, linux operating system and even Google services are first released as beta versions to selected users for product analysis and to get initial feedback before launching to the public. In this chapter, we make use of product ratings by users (either provided by users or predicted by recommender algorithms) as a way of modeling the user opinion of a product.

Node Parameters λ and μ. Recall that an active user v enters the ADOPT state with probability λ_v and it enters the TATTLE state with probability $1 - \lambda_v$. Typically, companies such as Netflix and Amazon have user logs which can be used to determine if a user reviewed the product without adopting it. Next to each review on Amazon, where the user bought the product being reviewed, there is a label that states "Amazon Verified Purchase" representing an adoption. The reviews without this label can be attributed to tattle nodes (states promote or inhibit). Given the user rating log, we learn λ_v as maximum likelihood estimate (MLE) which is the fraction of times a user provided an explicit rating for a product over the times the user provided any opinion including a comment, review and numeric rating.

The parameter μ_v models the inherent bias of user v. As with λ_v, we can rely on the rating log to compute this parameter. As an example, in the movie rating social network Flixster, there are special non-numeric ratings "want to see it" and "not interested" that map closely to the PROMOTE and INHIBIT states in our model respectively. Again, we use maximum likelihood estimate (MLE) to compute μ_v, that is, the fraction of times a user gave the rating "want to see it" over the number of times any such special rating was given by that user. We show the effectiveness of this choice by the means of extensive experiments in §8.4.

8.4 Model Evaluation

We perform empirical analysis to study the adoption of two types of "products" – movies and artists. We analyze influence spread and actual adoption (viewing) of movies on two datasets, one from a social network for movies, Flixster[46], and the other from a movie recommender system, Movielens[47], presented in §8.4.1. Further, we analyze a music social network, Last.fm[48], and study the adoption (listening) of songs and artists, presented in §8.4.2. Table 8.1 presents the basic statistics of all these datasets.

We compare the following models in our evaluation.

1. **Classical LT.** Linear Threshold model proposed in [78].
2. **LT-C.** Our proposed model.
3. **LT Ratings.** Our proposed model without TATTLE nodes. That is, all the nodes who are influenced adopt the product, and $\lambda_v = 1, \forall v \in V$. This is equivalent to modifying the activation function of the classical LT model to include ratings as defined in Eq. (8.1).

[46] www.flixster.com
[47] www.movielens.org
[48] www.last.fm

	Flixster	Movielens	Last.fm
#Nodes	13K	6040	1892
#Edges	192.4K	209K	25.4K
Avg. degree	14.8	34.6	13.4
#Movies or #Artists	25K	3706	17.6K
#Ratings	1.84M	1M	259K
#Edges with non-zero weight	75.7K	154K	157K

Table 8.1: Dataset statistics

4. **LT Tattle.** Our proposed model without any ratings. That is, all the nodes in ADOPT state are assumed to rate the item as $r_{\max}$, as do those in PROMOTE state, while users in INHIBIT state rate $r_{\min}$.

We tested these variations of LT-C in order to understand the relative contribution of the different components to the overall accuracy of predicting the expected adoption spread. In all the experiments, we run 10K Monte Carlo simulations to estimate the coverage for $k = 50$ seeds. The data was divided into test and training sets randomly such that all the ratings of a movie fall in exactly one of training or test sets. All validation experiments are run on the test set.

8.4.1 Adoption of Movies

Datasets. Flixster is a social network for movies which enables users to share their opinion on movies with friends by rating and reviewing movies. This dataset, collected by Jamali et al. [74], in its raw form has 1M users, 14M (undirected) friendship relations among users, and 8.2M ratings that range from half a star (rating 0.5) to five stars (rating 5). Since running Monte Carlo (MC) simulations is very expensive (may take several hours even for 10K nodes), a graph of that size is difficult to handle, given the extensive set of experiments we perform in our study. We use the Graclus[49] software to extract a subgraph which contains 13K users and 192.4K (directed) edges among them. There are 4.14M ratings by these users of which 1.84M are numeric ratings and 2.3M are special ratings.

Our second dataset on movies is from the Movielens recommender system released by Grouplens[50] research group. The dataset consists of 6K users and 1M ratings (on scale of 1-5) on 3.7K movies. The dataset does not have an explicit social graph, so we construct an implicit one representing the flow of influence. In Movielens, influence flows indirectly via the recommendation engine that is based on collaborative filtering [84]. For instance, if a movie is recommended to Bob, it is likely that the "nearest neighbors" of Bob (as seen by the recommendation engine) must have given high ratings to the movie. We deduce these implicit relationships among users by computing similarity among users. Precisely, an edge is created between users u and v if the Jaccard index[51] w.r.t. the movies they rated is greater than a

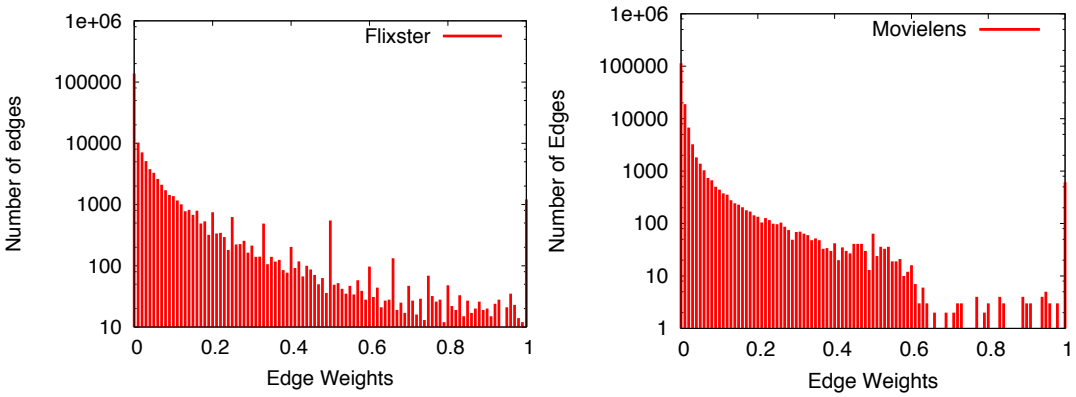
[49] `www.cs.utexas.edu/users/dml/Software/graclus.html`

[50] `www.grouplens.org/node/73`

[51] Jaccard index for sets X, Y is $J(X, Y) = |X \cap Y|/|X \cup Y|$.

threshold (0.25 in experiments). The process resulted in a graph with 6K nodes and 209K edges.

The size of the training set is 19.9K for Flixster, and 2.9K for Movielens; and test set is 5.1K for Flixster, and 741 for Movielens. Our experiments were run on 650 movies randomly picked from the test set for Flixster, and the entire test set of 741 movies for Movielens.

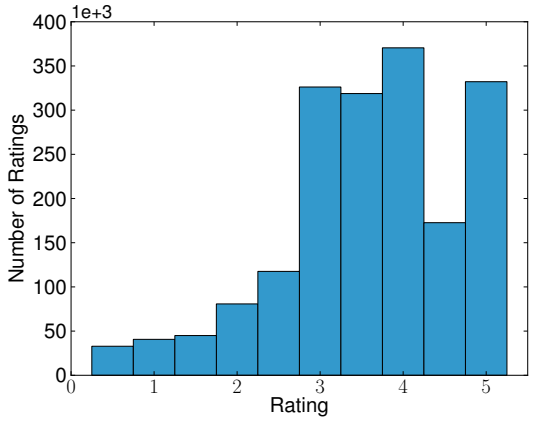


(a) Distribution of edge weights in Flixster (b) Distribution of edge weights in Movielens

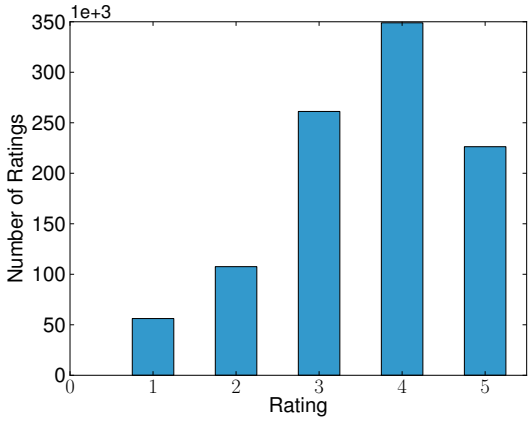
Figure 8.2: Distribution of edge weights

Model Parameters from Data. For both the datasets, edge weights are learned as described in §8.3.4. Figures 8.2(a) and 8.2(b) show the distribution of edge weights for Flixster and Movielens respectively. Flixster allows ratings from 0.5 to 5 in steps of 0.5, while Movielens allows integer ratings from 1 to 5. Figures 8.3(a) and 8.3(b) show the distribution of ratings for Flixster and Movielens respectively. The complete ratings matrix R is computed using the matrix factorization method described in [84] for both the datasets.

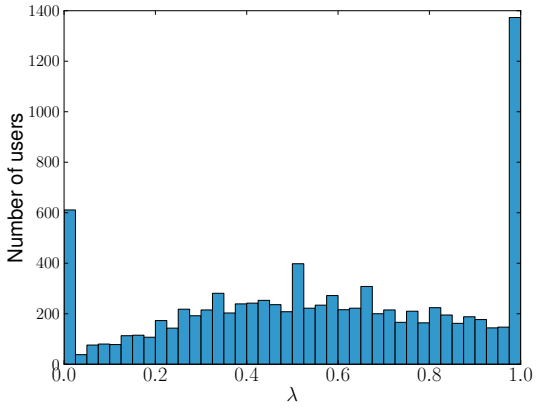
Flixster allows two types of *special* ratings – "want to see it" and "not interested" – in addition to the numeric ratings. There are 2.3M special ratings in our dataset, of which 730K ratings are "want to see it" and 1.6M are "not interested". We map the "want to see it" and "not interested" ratings to the PROMOTE and INHIBIT states in our model and fix their numeric values to 0.5 and 5 respectively. For any user $v \in V$, the adoption probability λ_v is computed as the fraction of times v gave a numeric rating over the number of all (numeric and special) ratings given by v. Similarly, μ_v is computed as a fraction of times v gives a "want to see it" rating over the number of special ratings given by v. Figures 8.3(c) and 8.3(d) show the distribution of the λ and μ parameters learned using the special ratings. In both the distributions, a large majority of the users have λ and μ values close to the ends of the spectrum i.e., 0 and 1. A value of one (zero) for λ for a user indicates that she adopts (tattles about) each product which she learns about. Similarly, a value of one (zero) for μ indicates if the user decides to tattle, she promotes (inhibits) the product. The distributions in Figures 8.3(c) and 8.3(d) show these strong preferences of users. Over 1200 users gave only numeric ratings and have $\lambda = 1$. Considering the special ratings, 1400 users gave "want to watch" ratings only ($\mu = 1$) and 1500 users gave "not interested" ratings only ($\mu = 0$). The fact that over half of all ratings are special and represent opinions shared by users who have not watched the given



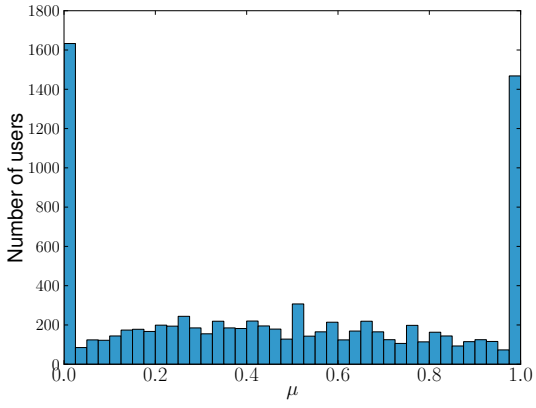
(a) Distribution of ratings in Flixster



(b) Distribution of ratings in Movielens



(c) Distribution of λ in Flixster



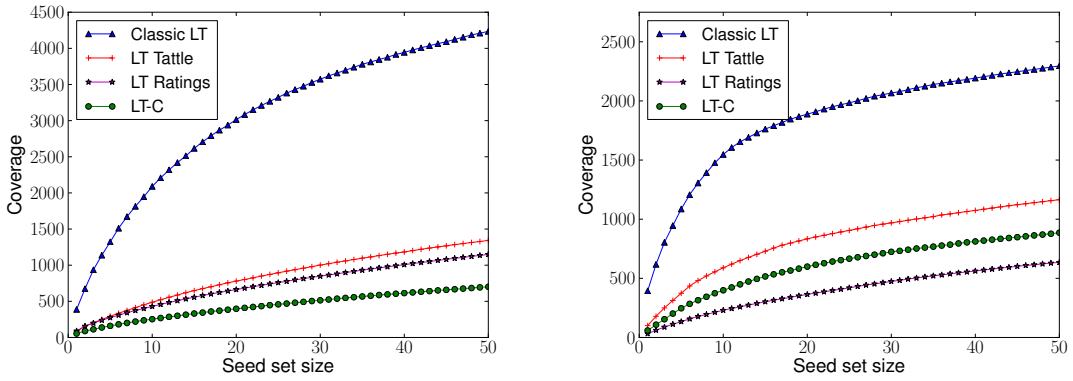
(d) Distribution of μ in Flixster

Figure 8.3: Summary of datasets and distributions of model parameters

movie emphasizes the need to study the effect of TATTLE nodes on adoption.

Movielens also features special ratings, such as, "want to see it" that corresponds to movies in a user's wishlist, and "hide this" that corresponds to "not interested". However, the actual dataset obtained from the Grouplens website did not include these ratings. In order to mitigate this, we use the distribution of λ and μ learned from Flixster for computing the values and for users in the Movielens dataset. Though not ideal, we believe this is a reasonable approximation since both data sets represent users' opinions on movies. The shape of the distributions in Figures 8.3(c) and 8.3(d) suggests a Beta probability distribution as a natural choice for modeling both λ and μ. We learn the parameters for the Beta distribution by fitting a curve to the data from Flixster, and obtain the distributions $\lambda_u \sim \text{Beta}(0.3, 0.1)$ $\mu_u \sim \text{Beta}(0.001, 0.001)$, for any user $u \in V$.

Coverage with different models. We first analyze the effect of ratings on product adoption. Figures 8.4(a) and 8.4(b) show the coverage obtained for a seed set size $k = 50$ using



(a) Coverage obtained with different models on Flixster

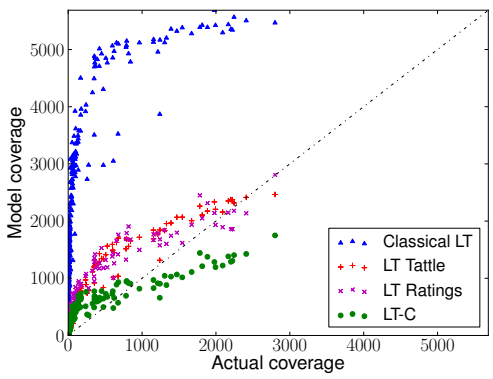
(b) Coverage obtained with different models on Movielens

Figure 8.4: Coverage obtained with different models on Flixster and Movielens datasets

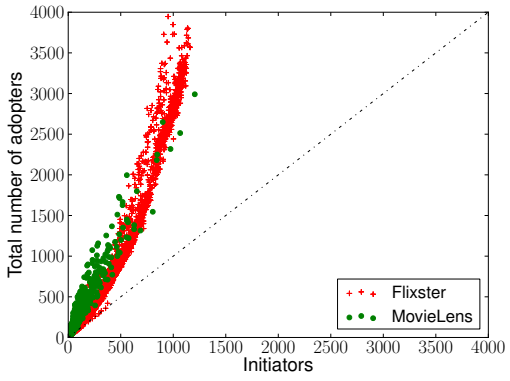
the classical LT model, LT Ratings, LT Tattle and the LT-C model for maximizing product adoption. The figures show this comparison for a particular movie, and the results were similar for a set of 30 and 20 movies picked randomly from Flixster and Movielens, respectively. As can be seen, there is a huge gap between the prediction of classical LT and of LT-C model. For instance, on Flixster, LT model predicts a coverage of 4200 with a budget of 50, and LT-C predicts a coverage of only 701 for that budget. Which model is correct? Is LT-C too pessimistic or is the classical LT model too optimistic? Coverage is an important statistic used in estimating the revenue (and thus profit) and hence, it is important to have an accurate estimate of it. As shown next, gauging models based on the absolute coverage they predict can often lead to models that perform poorly.

Accuracy of estimated coverage. We evaluate the coverage obtained by different models against the actual adoption of a product. We define the *actual adoption coverage* (actual coverage, for short) as the number of users who adopted the product, in this case, gave a numeric rating for a given movie. Since the coverage estimated by a model depends on the size of the seed set, we fix the seed set for each movie and compute the coverage by each model given the seed set. The seed set for each movie is the set of *initiators* which includes all users who watched a given movie before any of their friends did. Figures 8.5(a) and 8.5(c) validate that the LT-C model estimates actual adoption coverage accurately on both datasets. Each point in the figures represents a movie and its coordinates correspond to the actual coverage vs. that estimated by a model, for the set of initiators as the seeds. The set of initiators and hence the size of the seed set may be different for each movie. Figure 8.5(b) shows that the actual coverage is positively correlated with the number of initiators in both the datasets (shown are all movies in the datasets). Moreover, the final coverage is much larger than the number of initiators, suggesting that the network structure plays a significant role in the final spread of adoption.

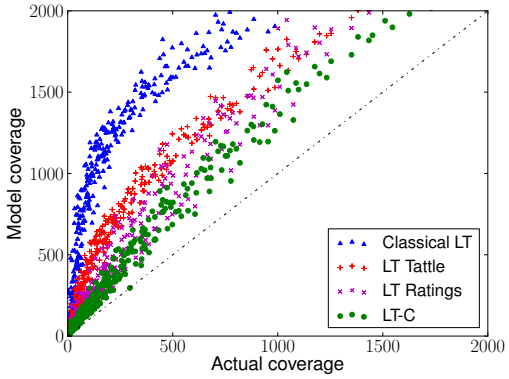
Figure 8.5(a) shows 650 movies and 8.5(c) shows 741 movies of the test set for Flixster and Movielens, respectively. The $x = y$ line is the ideal scenario where the model estimate



(a) Actual vs predicted coverage on Flixster



(b) Adoption Spread



(c) Actual vs predicted coverage on Movielens

Figure 8.5: Accuracy of estimated coverage.

is identical to the actual coverage. As seen from the figures, the classical LT model overestimates the coverage by large amounts. Both including tattle nodes and including ratings are useful in providing better estimates, while estimates of adoption using LT-C model are closest to reality. To quantify this error in estimating the coverage, we compute the average root mean square value (RMSE[52]) over the test set, for the coverage predicted by the different models w.r.t. actual coverage. We observe that the RMSE obtained using the classical LT model is over 10 times that by our model (on Flixster), as shown in Table 8.2. On Movielens, the LT-C and LT Ratings models have similar RMSE. This might be because we do not have data to learn values of λ and μ for various users, instead we draw their values from the Beta distribution whose parameters are taken from Flixster dataset (see above).

Another interesting trend observed in Figure 8.5(b) is that the choice of seeds is critical. For instance, we picked the 109 movies from Flixster where the number of the initiators is 50. The average final coverage on these movies is 58.2, which is quite close to the number of

[52]For vectors x, y of size n, $\text{RMSE}(x, y) = \sqrt{\sum_1^n (x_i - y_i)^2/n}$.

Table 8.2: Average RMSE for different models

Model	Flixster	Movielens
LT	1594.8	826.8
LT Ratings	269.3	301.8
LT Tattle	183	420.7
LT-C	144.6	301.7

initiators. On the other hand, if the initial seed set is picked carefully using the LT-C model, it is possible to achieve the coverage of even more than 500, as seen in Figure 8.4(a).

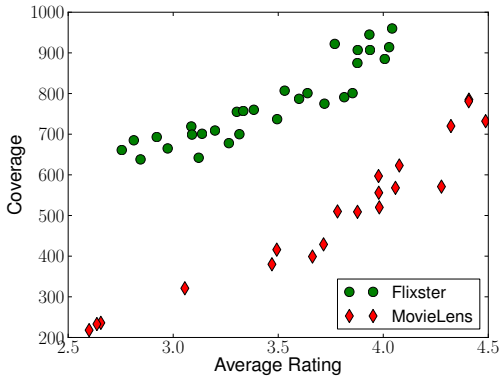
Product and seed set analysis

In Figure 8.6(a) we study the effect of ratings on the overall coverage given a budget of 50 seeds. Figure 8.6(a) shows the coverage for a set of 30 and 20 movies picked at random for the Flixster and Movielens datasets respectively. By taking ratings into account, the LT-C model predicts that good movies (rated high) are likely to have larger viewership compared with bad movies (rated low). By contrast, the coverage estimated using the LT model is the same across all movies, and is approximately 4200 nodes for movies in Flixster and 2300 for the Movielens dataset, significantly higher than the actual coverage in both cases.

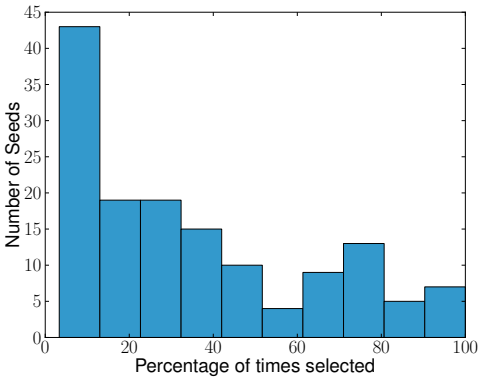
A natural question to ask is whether there are some nodes that are influential across actions (movies) and are *consistently* selected as seeds. Figures 8.6(b) and 8.6(c) show a histogram of the percentage of times a node was selected as a seed for the same set of movies in Figure 8.6(a). For the Flixster dataset, there are 7 seeds among the 144 unique seeds that are consistently selected for 90% of the movies analyzed. The consistency decays exponentially where only a few seeds have high consistency and most seeds have low consistency. A similar trend was observed for the Movielens dataset, where 10 out of 104 unique seeds were consistently selected 90% of the times. Interestingly, these nodes are not the top high degree nodes in the graph. This observation is consistent with a recent study [110] on influence in Twitter.

Evaluating model parameters

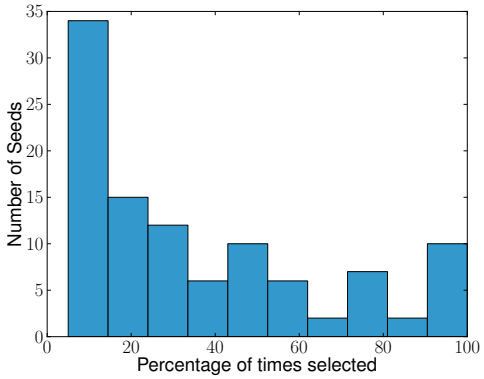
During our study, we realized that sometimes we may not have access to the data required for computing λ and μ for each user in the network, instead, we may be able to infer the distribution of both these parameters. We evaluate the coverage obtained by drawing each model parameter from appropriate distributions and comparing it with that obtained when all parameters are computed from the data. The results are shown in Figure 8.6(d). We analyze the coverage when the ratings are drawn from a normal distribution $\mathcal{N}(\mu_d, \nu_d)$ where the distribution mean $\mu_d = 3.59$ and variance $\nu_d = 1.01$, as computed from the given numeric ratings. The obtained coverage is almost the same as that obtained when ratings are computed using matrix factorization from the available ratings. Next, we analyze the coverage when λ and μ are drawn from Beta distributions as $\lambda_u \sim \text{Beta}(0.3, 0.1)$ $\mu_u \sim \text{Beta}(0.001, 0.001)$, for each user $u \in V$. These distributions are learned by fitting a Beta probability distribution to the λ, μ values computed from the Flixster data. The obtained coverage comes close to that obtained from data. In conclusion, it is encouraging to observe that drawing the model



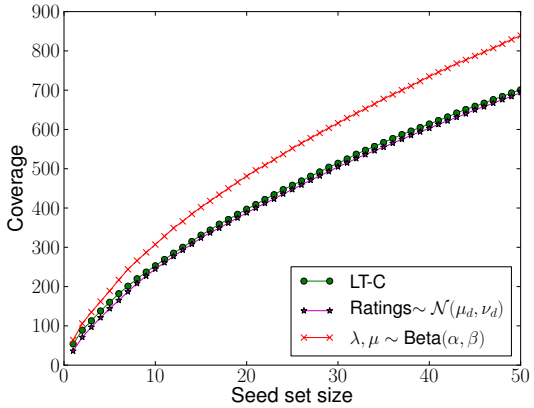
(a) Average rating of movie vs coverage for $k = 50$ on Flixster and Movielens



(b) Consistency of chosen seeds across actions on Flixster



(c) Consistency of chosen seeds across actions on Movielens



(d) Coverage with λ, μ and ratings from data and distributions on Flixster

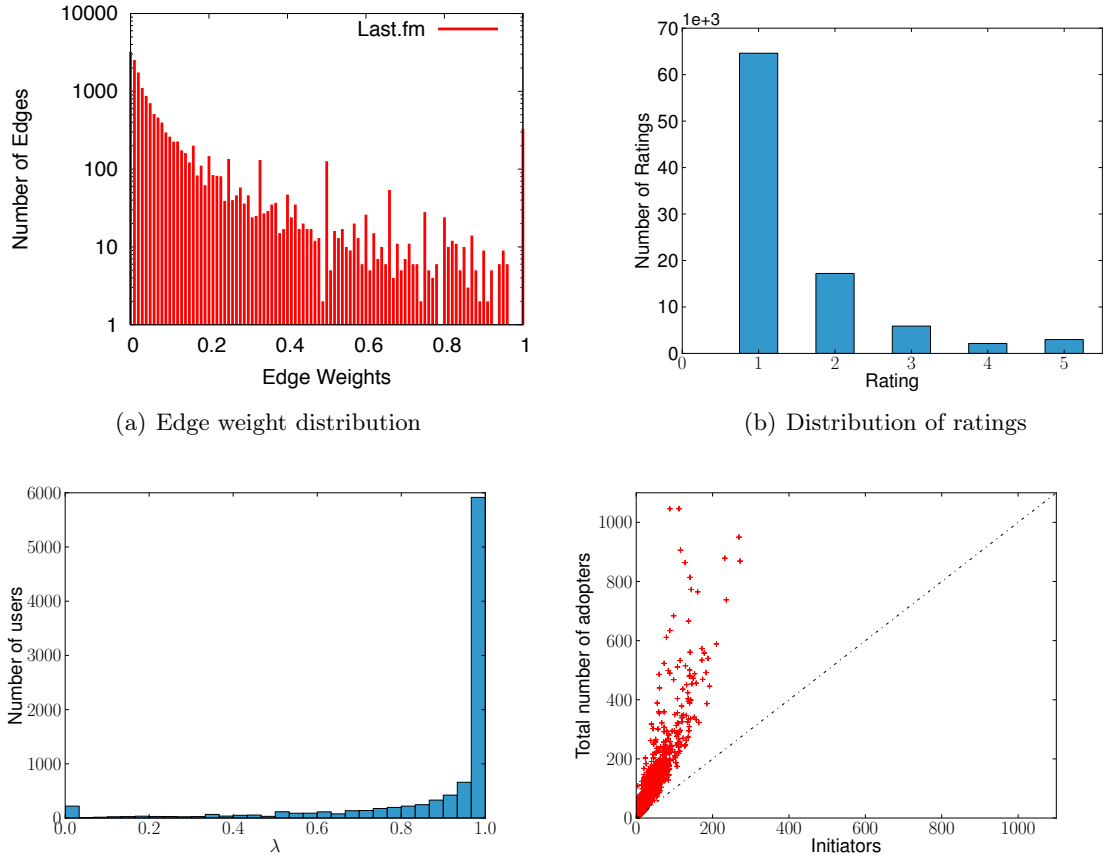
Figure 8.6: Coverage and seed set analysis on Flixster and Movielens datasets

parameters from a distribution that approximates the data may well give a good estimate of the coverage if access to user logs is limited.

8.4.2 Adoption of Music

Datasets. We study the adoption of music on a dataset from the popular music service Last.fm. The dataset we used was released for the Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011)[53]. The details of the dataset are presented in Table 8.1. It has 1892 users with 25.4K friendship relationships among them. The dataset includes the users' listening history in the form of number of times a user listened to songs by an artist, and users' tagging history for tagging artists.

[53] `http://ir.ii.uam.es/hetrec2011`



(a) Edge weight distribution

(b) Distribution of ratings

(c) Distribution of λ

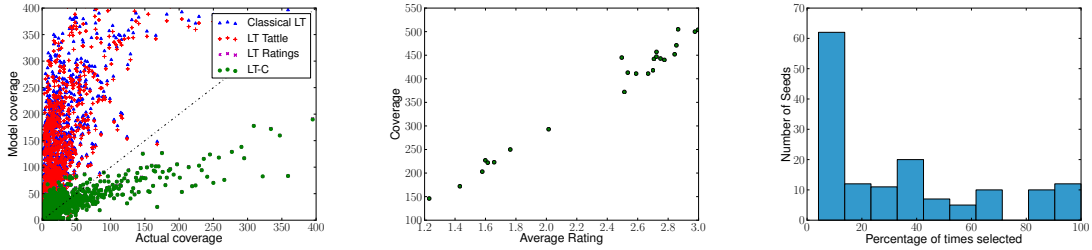
(d) Adoption Spread

Figure 8.7: Summary of dataset and distributions of model parameters for Last.fm

Model Parameters from Data. We use both tagging and listening history as actions for computing edge weights. Since the listening history is only available as playcounts, it does not come with timestamps. We assume these are most recent actions and assign the current timestamp to all listening actions. Now, edge weights can be learned as before. The distribution of edge weights is shown in Figure 8.7(a). The data was divided into test and training sets, with the test set consisting of 3067 randomly picked artists.

A user's opinion of an artist is implicitly present in the form of playcounts, i.e., the number of times the user has listened to songs by that artist. To infer ratings on a scale of 0 to 5 for each user, we partition the artists into 5 buckets, where the top bucket represents artists in the top 20-percentile playcounts, the second bucket in the next 20-percentile and so on. Each artist in the top bucket is assigned a rating of 5, next bucket corresponds to rating 4 and so on. It is well known that listening history follows a power-law distribution where a few artists (or songs) are heard many times and many artists (songs) are heard very few times. This pattern is observed in the obtained ratings as shown in Figure 8.7(b).

Last.fm allows users to indicate their preference on songs by marking them as "loved" or



(a) Actual vs predicted coverage (b) Average rating of movie vs coverage for $k = 50$ (c) Consistency of chosen seeds across actions

Figure 8.8: Coverage and seed set analysis on Last.fm dataset

"banned". For a given song, we regard the users who loved that song as being in the ADOPT state and those who banned that song as being in the INHIBIT state. Given this mapping, we can use the "loved" and "banned" songs for inferring a distribution of λ. However, the dataset for the HetRec workshop did not contain the information on loved and banned songs. We used the API from Last.fm to crawl this information for an arbitrary set of 10K users, and computed λ_u for a crawled user u as the fraction of loved songs over the number of loved and banned songs. Figure 8.7(c) shows the distribution of λ for the crawled users. A trend similar to that in Flixster is observed for values of λ in Last.fm, although with a larger skew towards adopting the product (in this case, an artist). We fit a Beta probability distribution to the values of λ and the best fit was obtained with Beta$(0.5, 0.001)$. Since the consumption of songs from an online music service does not cost much, it is not natural for users to want to listen to a song but not actually listen to it. Therefore, for this domain, $\mu_u = 0$, for every user u.

Accuracy of estimated coverage

Figure 8.8(a) compares the coverage obtained using classical LT, LT Tattle, LT Ratings and LT-C models with the actual coverage obtained by the initiators. Since the distribution of λ as obtained from the data is highly skewed towards 1, including the TATTLE state does not make the coverage estimate much better than the classical LT model. On the other hand, most ratings are low and hence their inclusion into the LT model has a greater impact for this dataset. The estimated coverage of LT Ratings and LT-C are similar, and provide a far better estimate compared with the classical LT model. The RMSE values averaged over 3067 artists for the four models were as follows, classical LT:97.5, LT Tattle:93.6, LT Ratings:31.7 and LT-C:31.7. These errors in coverage estimation show that for the music domain as well (where adoption may not cost money), incorporating users' ratings in the propagation model provides better estimates of adoption, compared with accounting for tattling but not ratings. In Figure 8.7(d), we present the actual coverage against the number of initiators, for all the artists. Again, it can be seen that the two are positively correlated.

Product and seed set analysis

Figure 8.8(b) shows that the coverage varies for artists that have different popularity (or rating). The coverage estimated by the classical LT model is the 1230, regardless of the artist, again way off from reality. The trend in consistency of seeds observed for music domain is similar to that seen for movies. Figure 8.8(c) shows that few (i.e., 12) seeds are picked over 90% of the times and 62 out of 149 unique seeds are picked in less than 10% of the 20 artists analyzed. These two experiments show that using LT-C, it is not only the coverage that is different for different products, even the target seeds are different. In contrast, the classical model disregards these nuances. This suggests taking such nuances into account may lead to a more accurate and effective model for viral marketing.

8.5 Conclusions and Future Work

Classical diffusion models such as IC and LT do not distinguish between influence and product adoption. Thus, they implicitly assume that once influenced, a node necessarily adopts a product and that adopters always influence other users to adopt the product. Our observations on real data show that sometimes influenced users, once they become active, may choose to not adopt but instead tattle about the product; by doing so, they may either promote or inhibit adoption by other users. Furthermore, adopters may endorse a product to varying degrees based on their experience with the product, as often reflected by the ratings they provide in several real data sets.

In this chapter, we proposed a propagation model called LT-C model that accounts for these observations. We formalized the problem of adoption maximization as distinguished from Influence Maximization and showed that it is NP-hard. We also showed the expected adoption spread function under the LT-C model is monotone and submodular and thus, the classic greedy algorithm can be used to get an approximate solution. We conducted extensive experiments on two domains – two data sets from movies and one from the music domain. Our results show that w.r.t. the accuracy of spread prediction, the LT-C model is consistently the best and the classical LT model is consistently the worst.

There is still a long way to go to develop a truly realistic product adoption model. It may be possible that the users are passive and may not express any opinion even after adopting a product [110]. One way to incorporate this is to split the Adopt state into two sub-states, viz., "adopt and rate" and "adopt and not rate". However, it is not clear how we can identify the latter state (for learning the model parameters) in the datasets we can access. Similarly, users may not find tattlers' ratings as trustworthy as adopters' ratings. Marginalizing tattlers' ratings may improve the predictions. Extending the model by incorporating the five stages of product adoption [19] can be useful. Inclusion of negative opinions in the spirit of [34] within the framework of our model is interesting and may lead to an even more expressive model for product adoption. Next, it is important to validate the LT-C model against many more real data sets from diverse domains. Last but not the least, scalable heuristic algorithms need to be developed for the LT-C model in order to handle very large networks and seed sets. Our ongoing work addresses some of these questions.

Chapter 9

RecMax: Targeted Advertising in Recommender Systems

In recent times, collaborative filtering based Recommender Systems (RS) have become extremely popular. While research in recommender systems has mostly focused on improving the accuracy of recommendations, in this chapter, we look at the "flip" side of a RS. That is, instead of improving existing recommender algorithms, we ask whether we can use an existing operational RS to launch a targeted marketing campaign. To this end, we propose a novel problem called RECMAX that aims to select a set of "seed" users for a marketing campaign for a new product, such that if they endorse the product by providing relatively high ratings, the number of other users to whom the product is recommended by the underlying RS algorithm is maximum. We motivate RECMAX with real world applications. We show that seeding can make a substantial difference, if done carefully. We prove that RECMAX is not only NP-hard to solve optimally, it is NP-hard to even approximate within any reasonable factor. Given this hardness, we explore several natural heuristics on 3 real world datasets – Movielens, Yahoo! Music and Jester Joke and report our findings. We show that even though RECMAX is hard to approximate, simple natural heuristics may provide impressive gains, for targeted marketing using RS.

This chapter is based on our KDD 2012 paper [59]. This study was performed in collaboration with Laks V. S. Lakshmanan.

9.1 Introduction

There are many applications like online shopping where the opinions of other users who bought and rated a product are important for a user trying to decide which product to buy. Recommender systems (RS) have emerged as a complementary paradigm for search in order to fulfill users' information needs in such applications, by providing mass personalization [108]. Much of their popularity was spurred by the early success of collaborative filtering techniques [67], which exploit the existing ratings in a system to estimate the expected rating a user might give a product. Today, RS form the backbone of the business of many companies like Amazon and Netflix.

RS research has mostly focused on improving the accuracy with which the algorithm predicts the likely rating a user will provide an item she has not experienced before. The predicted ratings are then used for recommending items to users, e.g., by picking the top-ℓ items with highest predicted rating for the active user. In this chapter, we take a look at the "flip" side of a RS. That is, instead of improving existing recommender algorithms, we ask whether we can

use an *existing* operational RS to launch a targeted marketing campaign. More concretely, you have a product you want to sell in an online market place powered by RS technology. Suppose you give free samples of your product to a select number of users, called the *seed* users. Based on its quality, the seed users rate the product, preferably with high ratings. E.g., Amazon's Vine program employs a similar scheme. The question is, under the condition that the seed users endorse the product with high ratings, can this lead to the product being recommended by the system to a large number of other users? If so, this represents a huge opportunity for the client companies launching the campaign as it helps effectively advertise their product via the RS. It is a business opportunity for the host company as well since it can offer seeding as a service, whereby it selects effective seeds on behalf of the clients. It is beneficial to the seed users as they get free or discounted samples of a new product. Finally, it is helpful to the rest of the users as they receive recommendations of new products earlier than they otherwise would, a well recognized problem in RS [89].

In this chapter, we will show the answer to the above question is "yes" and that the manner in which seeds are selected can make a difference. We focus on the case where the seed users who are targeted like the new product and endorse it with relatively high ratings. As example applications, Warner Brothers can target a select number of Netflix users in order to give free samples of their new movie release. A publisher can target customers of Amazon in a similar way. Motivated by these applications, we study the following problem, called RECMAX: Given a collaborative RS, a number k and a new product ρ, find k users (called *seed set*) to whom ρ should be marketed such that if these users rate the new product with relatively high ratings, the number of users to whom the product is recommended by the system is maximum.

Many ventures like *Increase YouTube Views*[54] and *MileStone Internet Marketing*[55] have emerged that offer services to increase/improve the ratings and reviews of their clients. Amazon's Vine program operates an interesting ecosystem, whereby reviewers (users) ranked highly by other users based on their past reviews are invited to review new items. These reviewers in turn are offered pre-release versions of products for the purpose of their review. We are not aware of any research works that investigate the mechanism used by the above ventures or by Amazon.

RECMAX has an interesting application for the cold-start items problem [7, 89], for the manufacturer of the item: for a newly introduced item, there may be few ratings provided by users. As a RS based on collaborative filtering relies on existing ratings, the new item will not be recommended to users until it gets enough ratings. Using RECMAX, when a new item is introduced, the item manufacturer has the opportunity to market it to the seed users and have the system recommend it to other users. Further comparison with works dealing with the cold-start problem appears in Section 9.2. There has been substantial work on detecting spam in RS and making them robust against spam. Mobasher [100] describes an attack called shilling attack whereby user profiles are injected into the system and reviewers are made to write positive reviews. The goal of RECMAX is *not* to spam a RS. Rather, it is to leverage a normal working RS for purposes of marketing. In particular, the ratings provided by the seed users to the new product are not influenced or interfered with in any way. They are free to

[54] `www.increaseyoutubeviews.com`

[55] `www.milestoneinternet.com/products/social-media/hotel-internet-marketing-reviews.aspx`

rate it as they see fit. We focus on the case where the seed users like the new product and provide a relatively high rating.

The single most important challenge in studying the RECMAX problem is the wide diversity of RS algorithms (see [108], and [1] for comprehensive surveys) that are used in real applications. While User-based [67] and Item-based [118] collaborative filtering methods making use of neighborhood are extensively used, recently model-based approaches such as Matrix Factorization [128] have attracted a lot of attention. All these methods have many variants thus making the problem very difficult to formulate and study. In this work, we focus on User-based and Item-based methods since they are used in many real systems and are representative.

Targeted marketing in RS by selecting seed users has been recognized before [7, 17, 42, 106]. All of them use the notion of influence that a user has might exert in a RS, and based on their definition of influence, they select the seed users. The definition of influence and hence the goal is different in each of these works (as elaborated in Section 9.2). While the motivation and overall strategy are similar, that is, to select a set of seed users for marketing, RECMAX focuses on selecting seed users for maximizing the number of (other) users to whom a new item will be recommended by the RS, directly based on a recommender algorithm, as opposed to using an auxiliary notion of influence and resorting to social influence maximization. One of our main contributions is a thorough theoretical analysis of the complexity of RECMAX. As we shall show, RECMAX is not only a hard problem, it's even NP-hard to approximate within any reasonable factor. Given this, we explore various natural heuristics for solving RECMAX. We undertake a detailed analysis of three diverse real data sets, evaluate several natural heuristics for seed selection on those data sets and report our findings. One of the key contributions is showing that RECMAX is a real practical problem where selecting good seeds can yield a big payoff. To our knowledge, *we are the first to study the problem of identifying seed users to market a product to, in order to trigger a large number of recommendations of the product to other users from an existing RS*. We make the following contributions.

- We propose a novel problem RECMAX. It aims to find a set of seed users to whom a new product should be initially marketed such that if they endorse the product, the number of users to whom the product is recommended is maximum. We offer early empirical evidence that seeding does help in boosting the number of recommendations (Section 9.4).

- We perform a thorough theoretical analysis of RECMAX. We show that RECMAX is not only NP-hard to solve optimally, it is NP-hard to approximate within a factor of $1/(|V|^{1-\epsilon})$ for any fixed $\epsilon > 0$, both in User-based and Item-based methods (Section 9.5), under the settings we study. We also present intuitions behind where the intrinsic hardness of RECMAX comes from.

- Given the hardness, we explore several natural heuristics on 3 real datasets (Section 9.6). Some of the key observations we make are: a) If enough budget is allowed, seeding can make a substantial difference in the number of recommendations a new product can get; b) The gains that can be achieved saturates after some point, in both methods, suggesting that the seeding does not help after a certain budget; c) The overall gains that can be achieved in User-based systems are much higher than can be achieved in

Item-based systems, though saturation is attained much quicker in Item-based systems; d) Choice of the seed set is critical.

Related work is reviewed in Section 9.2 while in Section 9.3, we provide a brief background and define RECMAX formally. Finally, we summarize the chapter and discuss future research in Section 9.7.

9.2 Related Work

Recommender Systems: The majority of works in RS (see [108] and [1] for detailed surveys) focus on improving the quality of the algorithms w.r.t. prediction accuracy. RS in general can be content-based or based on collaborative filtering. The most popular methodology in RS is based on collaborative filtering, which seeks to predict the expected rating that a user may provide to an item which she hasn't experienced before. Collaborative filtering methods are further classified into memory(or neighborhood)-based (e.g., see Chapter 4 of [108]) or model-based (e.g., matrix factorization [108], Chapter 5). This wide variety in RS makes RECMAX a very challenging problem to define and study. In this chapter, we study both User-based and Item-based neighborhood methods, when the predicted ratings are estimated through weighted average.

A by-product of RECMAX is a solution to the cold-start problem [89], for item manufacturers. Various approaches have been proposed to solve the cold start problem including hybrid methods which in addition to existing ratings, utilize the item content data [89]. A recent work by Anand and Griffiths [7] employs an interesting incentive based approach which, given a collection of new items, calls for offering a list of new items for every user in the system. A user gets a fixed payment for every new item she rates. The payment is determined using the influence of a user in the RS. The authors provide general guidelines on what factors the influence should depend on without explicitly defining it. They infer a social graph and apply existing social influence maximizing heuristics like Degree Discount [36]. As such this is a "proxy" to the original problem motivated by them. Moreover, degree discount is proposed for the independent cascade model where attempts of influence on a user by her neighbors are assumed to be independent, an assumption that, strictly speaking, doesn't hold in their framework. As the authors themselves acknowledge, their framework is vulnerable to fake ratings as the ratings provided by the seed users may be compromised by the free payments they get in return. The key differences from this work are as follows: a) Solving the cold start problem is not our main goal, although it is an interesting by-product; b) We look to pick seed users in a way that maximizes the number of users to whom the item is recommended directly using the underlying algorithms the RS use (e.g., User-based or Item-Based), instead of relying on social influence maximization; c) As mentioned earlier, ratings provided by seed users are not dictated or interfered with in anyway, avoiding fake ratings.

Another related line of work is spam in RS and its detection [100]. As mentioned earlier, the goal of RECMAX is not to spam the RS or to encourage fake ratings for the sake of item promotion. Instead, our goal is to leverage RS for ethical marketing. If seed users do not endorse a new item by providing high ratings, that item would simply not be recommended

to other users. In a way, the seed users also play a role of initial customer feedback in our framework.

Influence Maximization: Our study is related in spirit to the problem of Influence Maximization in social networks as well as to the related problems of adoption and revenue maximization [17, 42, 65, 78]. In fact, the notion of influence from a data mining perspective was first studied in the context of RS [42]. Later Kempe et al. [78] formalized the problem as a discrete optimization problem. To the best of our knowledge, till date, there have been 4 papers which measure users' influence for marketing in RS by selecting seed users [7, 17, 42, 106]. Domingos et al. [42] define influence as the expected lift in profit and aim to maximize it. Anand and Griffiths [7] on the other hand, calculate influence empirically by exploiting social influence maximization heuristics. We provided a detailed account of the differences with our work above. Rashid et al. [106] focus on measuring influence scores of users in a RS and define it as a function of user's ability to change the predicted ratings of other users by δ where δ is the difference between consecutive rating valies (e.g., in a standard 1-5 rating scale, $\delta = 1$). Their main goal is to calculate influence scores of users, not to select a seed set. Bhagat et al. [17] on the other hand (presented as Chapter 8 in this dissertation), develop a model for product adoption and use it to select seed users to maximize a new product adoption, from an Influence Maximization perspective.

Several factors set Recmax apart from Influence Maximization. A fundamental distinction is that in Recmax, the goal is to maximize the number of users to whom the system will *recommend* the item. By contrast, in Influence Maximization, the goal is to maximize the number of users who are influenced by the seed users and *adopt* the item. Second, no explicit social network is assumed as input for Recmax, whereas for influence/adoption/revenue maximization, the basic backbone is a social network with influence weights. While in principle, one can induce a social network from a RS as [7] does, the resulting problem formulated on the social network is at best a proxy for the original problem of recommendation maximization. Third, unlike in Influence Maximization, there is no underlying propagation model (such as indepndent cascade or linear threshold – see [78]) in Recmax.

9.3 Background and Problem Studied

Recommender systems serve the top-ℓ items with the highest predicted ratings for a user as recommendations to that user, where ℓ is the desired length of the recommendation list. While User-based [67] and Item-based [118] are among the most popular collaborative filtering methods, recently hybrid approaches have been proposed [134]. Even more recently, Matrix factorization based methods [128] have gained significant attention. For excellent surveys, see Chapters 4 and 5 of [108]. In this work, we study Recmax on User-based and Item-based methods. With $\hat{R}(v,i)$, we denote the expected rating that user v gives an item i. Among User-based methods, perhaps the most popular approach to estimate $\hat{R}(v,i)$ is to use the weighted average:

$$\hat{R}(v,i) = \frac{\sum_{u \in N_i(v)} w(u,v) \cdot R(u,i)}{\sum_{u \in N_i(v)} abs(w(u,v))} \tag{9.1}$$

where $R(u,i)$ is the rating given to item i by user u, $abs(\cdot)$ is the absolute value function and $N_i(v)$ is the set of nearest neighbors of v, based on the similarity scores $w(\cdot,v)$, who have rated i. The similarity scores are calculated using cosine similarity or Pearson correlation or one of their variations (see [108], Ch. 4). In this work, we employ the Pearson correlation as the similarity measure for User-based methods. Let $I(u)$ denote the set of items rated by user u and let $\mathcal{I} = I(u) \cap I(v)$. Then the similarity $w(u,v)$ using Pearson correlation is computed as

$$\frac{\sum_{i\in\mathcal{I}}(R(u,i)-\bar{R}_u)\cdot(R(v,i)-\bar{R}_v)}{\sqrt{\sum_{i\in\mathcal{I}}(R(u,i)-\bar{R}_u)^2}\cdot\sqrt{\sum_{i\in\mathcal{I}}(R(v,i)-\bar{R}_v)^2}} \tag{9.2}$$

where $\bar{R}_u$ is the average of u's ratings over various items.

Item-based methods on the other hand, use the similarities among items to estimate the predicted ratings. For instance, using the weighted average,

$$\hat{R}(v,j) = \frac{\sum_{i\in N_v(j)} w(i,j)\cdot R(v,i)}{\sum_{i\in N_v(j)} abs(w(i,j))} \tag{9.3}$$

where $N_v(j)$ is the set of nearest neighbors of (i.e., the most similar items to) j that are rated by v. Again, the similarities can be computed based on cosine, Pearson, their variants or other methods (see [108], Ch. 4). In this work, we focus on adjusted cosine similarity, which is shown to be the most accurate similarity measure for Item-based methods [118]. Let $V(i)$ denote that set of users who rate item i, and let $U = V(i) \cap V(j)$. Then the adjusted cosine similarity between items i and j is

$$\frac{\sum_{u\in U}(R(u,i)-\bar{R}_u)\cdot(R(u,j)-\bar{R}_u)}{\sqrt{\sum_{u\in U}(R(u,i)-\bar{R}_u)^2}\cdot\sqrt{\sum_{u\in U}(R(u,j)-\bar{R}_u)^2}} \tag{9.4}$$

In both User-based and Item-based methods, the number of nearest neighbors considered is often restricted to a limited number d, usually between 20 and 50. This has been found to improve not only scalability, but even accuracy [67, 108].

Problem Studied. In addition to the log of past ratings, we assume we know the algorithm used by the RS. In this work, we focus on both User-based and Item-based methods. For User-based, we employ Pearson correlation similarity measure, while for Item-based, we utilize adjusted cosine similarity measure, both defined above.

With ρ, we denote the new item to be marketed. The number of items that can be recommended to a user v is limited and as a result various items are in competition to enter the recommendation lists of users. Clearly, for ρ to appear in the recommendation list of user v, the expected rating $\hat{R}(v,\rho)$ must be more than the *rating threshold* of user v, which is the predicted rating of v for the last item in its recommendation list. We use θ_v to refer to the rating threshold of user v. We do not necessarily assume that the new item ρ has no prior ratings.

Let V be the set of users in the system. The problem we study in this chapter is to select a set $S \subseteq V$ of k users (called *seed set*) such that by convincing them to provide relatively high ratings to the new item ρ, the number of users to whom it is recommended is maximized.

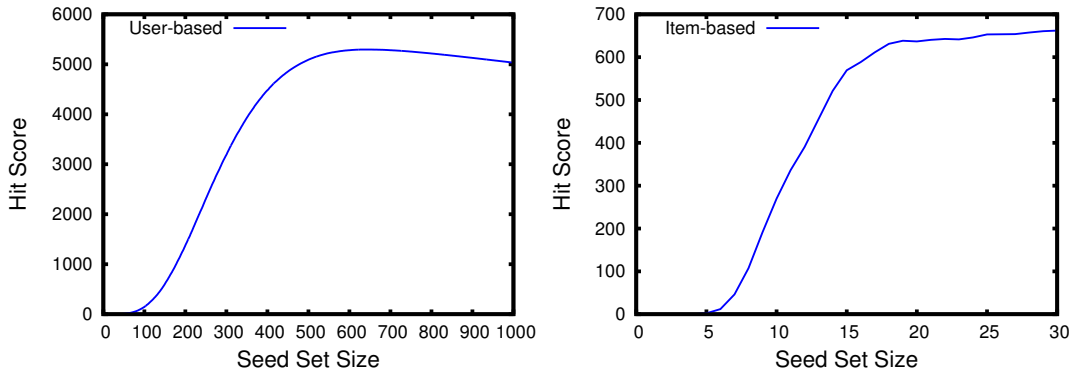


Figure 9.1: Hit Rate achieved by random seed set on Movielens dataset on User-based (left) and Item-based (right). The plots show that even when the seed sets are selected randomly, seeding does help and exhibits impressive gains in hit score.

We leave the definition of "relatively high rating" open for now and argue that if the new item is not a good item causing the seed users to provide relatively low ratings, then in any case, it should *not* receive many recommendations. The goal of this work is not to promote bad items. In our experiments, we derive a rating $R(u, \rho)$ for a seed user u by taking an average of the top-20% highest ratings provided by u.

We use $H(S)$ to denote the set of users to whom the item is recommended by virtue of the seed set S. Clearly, $H(S) \subseteq V \setminus S$ as the users in S have already adopted the item and there is no value to recommending the item to them. Intuitively, users in the seed set S should have high "influence" where the influence flows indirectly via the recommendations. We define $f(S) := |H(S)|$, the *hit score* of seed set S as:

$$f(S) = \sum_{v \in V - S} I(\hat{R}(v, \rho) > \theta_v) \tag{9.5}$$

where $I(\cdot)$ is an indicator function which is 1 if $\hat{R}(v, \rho) > \theta_v$ and 0 otherwise. Formally, the problem is defined as follows.

Problem 10 (RECMAX)**.** Given a recommender system and rating thresholds θ_v for each user v in the system, and a number k, find a set S of k users s.t. $f(S)$ is maximized.

9.4 Does Seeding help?

Before we study the RECMAX problem of how to select an optimal seed set, we address several natural questions: Does seeding help? What are the gains that may be achieved? How does hit score vary with respect to the seed set size?

To answer these questions, in this section, we show the results from preliminary experiments, using the popular Movielens data set. More detailed experiments are discussed in Section 9.6. Predicted ratings are computed according to weighted average, both for User-based and Item-based methods (see Eq. 9.1 and 9.3). The similarity measure we use is Pearson

correlation for User-based (Eq., 9.2) and adjusted cosine for Item-based (Eq. 9.4). See Section 9.6 for explicit experiment settings.

Next, we run a RANDOM heuristic to select the seed set and calculate the hit score achieved, both in User-based and Item-based methods. We assume that a seed user $u \in S$ provides a relatively high rating to the new item. In our experimental evaluation, we set this rating as the average of the top-20% highest ratings provided by user u. Given a budget k, a seed set of size k is randomly selected and the hit score is computed. We repeat this process 100 times and take the average. Note that it is important to repeat the process several times as in some cases, the seed set picked by RANDOM may be really good with a large hit score, while in other cases, very poor with a low hit score. Fig. 9.1 shows the results. As can be seen, the hit score achieved is remarkable, under both methods, with a much higher hit score under User-based. For example, a budget of 500 can get a hit score of 5091 on User-based and a budget of 20 can achieve a hit score of 636 on Item-based.

This immediately suggests that *seeding does help and establishes the case for* RECMAX. Another interesting observation is that the hit score curve resembles the classic sigmoid curve, implying that if sufficient budget is allowed, then the gains achieved can be substantial. Moreover, if we continue to increase the budget, the hit score plateaus, and sometimes, it may even start decreasing (unlike sigmoid). These interesting observations inspire us to perform a deeper analysis of RECMAX – both theoretical and empirical – and we do so in the next sections.

9.5 Complexity of RecMax

In this section, we study the complexity of RECMAX. Unfortunately, it turns out to be intractable: in fact, it is hard to approximate within any reasonable factor, unless P=NP. Due to the wide variations in RS methods, it is tedious to study the complexity for everyone of them. Instead we present formal proofs for cases where the underlying methods are either User-based or Item-based, with Pearson and adjusted cosine as similarity functions respectively. We believe these are representative cases and our results give an indication that RECMAX may indeed be a very hard problem for known RS algorithms in general.

We also provide important insights as to why RECMAX is so hard. First, it is easy to show that the function $f(S)$ defining the hit score of a seed set is in general non-monotone and non-submodular. Intuitively, $f(\cdot)$ is based on predicted ratings computed using weighted average, which is neither monotone nor submodular. Indeed, the plot in Fig. 9.1 (left) shows an example where $f(\cdot)$ is not monotone and both (left) and (right) plots show it is not submodular in general. To help us prove the hardness results, we introduce a helper problem that we call *Maximum Encirclement Problem* (MEP). We show that MEP is as hard to approximate as the classical *Maximum Independent Set Problem* (MIS). Recall that in an undirected graph $G = (V, E)$, a set $S \subseteq V$ is an independent set iff no two nodes in S are adjacent. MIS cannot be approximated within a factor of $1/(|V|^{1-\epsilon})$ for any fixed $\epsilon > 0$, unless P=NP [66]. We show that RECMAX is as hard to approximate as MEP and our result follows.

Given an undirected graph $G = (V, E)$ and a set $S \subseteq V$, we say a node $v \in V \setminus S$ is *encircled* by S, provided v has at least one neighbor and all its neighbors are in S.

Problem 11 (MEP)**.** Given an undirected graph $G = (V, E)$ and an integer k, find a set $S \subseteq V$ such that $|S| \leq k$ and S encircles the maximum number of nodes.

For the sake of simplicity, we assume there are no isolated (i.e., zero degree) nodes in the graph. As an insight, notice that MEP is very close to the minimum vertex cover problem (MVC) which asks if there is a set $S \subseteq V$ of size $\leq k$, given $k \leq |V|$, such that every edge of the graph is incident on at least one node in S. Notice, S is a vertex cover iff for any node $v \in V \setminus S$, all its neighbors are in S, i.e., iff v is encircled by S (assuming there are no isolated nodes). Thus, MVC is equivalent to asking if there is a set of nodes $S \subseteq V$ of size k such that it encircles rest of the nodes in the graph. From this, it follows that MEP is NP-hard, which is not very surprising. What is interesting is that while MVC enjoys 2-approximation, MEP is not approximable, as we show next.

Lemma 8. *The Maximum Encirclement Problem (*MEP*) cannot be approximated within a factor of* $\frac{1}{|V|^{1-\epsilon}}$ *for any fixed* $\epsilon > 0$*, unless P=NP.*

Proof. Consider an instance of MIS, consisting of a graph $G = (V, E)$. Create an instance of MEP by keeping the graph same. Clearly, a set $S \subseteq V$ is an independent set of G if and only if the set $V \setminus S$ encircles each node in S. In particular, S is an optimal solution to MIS iff $V \setminus S$ is an optimal solution to MEP where the budget is set to $|V| - |S|$. Assume there is a β-approximation algorithm for MEP. For $k = 1, ..., |V|$, run the approximation algorithm on the MEP instance with the budget set to k. Let $\hat{S}$ be a set of nodes output by this algorithm for some value of $\hat{k}$: $|\hat{S}| \leq \hat{k}$ and the number of encircled nodes is the maximum among all values of k. Clearly, the number of nodes encircled by $\hat{S}$ is $\geq \beta \cdot (|V| - \hat{k})$, since for some budget, there must exist a set that encircles every node outside the set. Let T be the set of nodes encircled by $\hat{S}$. Clearly, T is an independent set of size $\geq \beta \cdot (|V| - \hat{k})$ which thus is a β-approximate solution to MIS. Therefore, MEP is as hard to approximate as MIS. □

Now, we are ready to prove hardness results for RECMAX.

Theorem 15. *Under User-based Collaborative Filtering that uses Eq. 9.1 and 9.2 to compute predicted ratings,* RECMAX *is NP-hard. Moreover, it cannot be approximated within a factor of* $\frac{1}{|V|^{1-\epsilon}}$ *for any* $\epsilon > 0$*, unless P=NP.*

Proof. We prove the theorem by reducing MEP to RECMAX. From an instance $\mathcal{I}$ of MEP consisting of graph $G = (V, E)$, create an instance $\mathcal{J}$ of RECMAX as follows. For each node $u \in V$, create a user u in instance $\mathcal{J}$. For each edge $(u, v) \in E$, create an item i_{uv} such that users u and v rate the item i_{uv} with rating 2. Add $|V|$ dummy items to the instance $\mathcal{J}$ such that each user rates exactly one of these dummy items, with rating 1. Let d be the maximum number of nearest neighbors that are being considered by the RS, then also add d dummy users (call this set of dummy users X) in instance $\mathcal{J}$ such that each of these dummy users $x \in X$ rates all $|V|$ dummy items with rating 2. Hence, in our construction, we have $|V| + d$ users, $|E| + |V|$ items and $2 \cdot |E| + (d + 1) \cdot |V|$ ratings. Let every seed user provide the same rating to the new item, say 2. Next, let each dummy user provide rating 1 to the new item ρ. We will decide the rating threshold θ_v later.

Let $S \subseteq V$ be any seed set. Then, we claim that a node $v \in V \setminus S$ is encircled by S (in instance $\mathcal{I}$) iff the new item is recommended to v, that is, $v \in H(S)$ (in instance $\mathcal{J}$). We prove this claim below. From this claim, it is easy to see that a set S of size k encircles η nodes in instance $\mathcal{I}$ if and only if $|H(S)| = \eta$ in instance $\mathcal{J}$. Therefore, RecMax is as hard to approximate as Mep, following the same logic used in the proof of Lemma 8, and this proves the theorem.

Consider a user $v \notin S$. According to Eq. 9.2, the similarity between v and a user $u \in S$ is 1. Likewise, the similarity among v and a dummy user $x \in X$ is -1 With $deg(v)$, we denote the degree of user v in instance $\mathcal{I}$. If $l \leq deg(v)$ (non-dummy) neighbors of v rate the new item ρ, then, according to Eq. 9.1,

$$\begin{aligned}\hat{R}(v,\rho) &= \frac{\sum_{u \in N_\rho(v)} w(u,v) \cdot R(u,i)}{\sum_{u \in N_\rho(v)} abs(w(u,v))} \\ &= \frac{l \cdot 1 \cdot 2 + (d-l) \cdot (-1) \cdot 1}{abs(l \cdot 1) + abs((d-l) \cdot (-1))} = \frac{3 \cdot l - d}{d}\end{aligned}$$

Set the rating threshold $\theta_v = (3 \cdot deg(v) - d)/d - \delta$ where δ is a small number to ensure that $\hat{R}(v,\rho)$ is $(3 \cdot deg(v) - d)/d$ iff the new item is recommended to v. This rating threshold can be achieved iff all neighbors of v rate the new item. As a result, an item is recommended to a user v iff all of its neighbors are in S. that is, iff v is encircled by S. This was to be shown. □

Theorem 16. *Under Item-based Collaborative Filtering that uses Eq. 9.3 and 9.4 to compute predicted ratings,* RecMax *is NP-hard. Moreover, it cannot be approximated within a factor of* $\frac{1}{|V|^{1-\epsilon}}$ *for any fixed* $\epsilon > 0$*, unless P=NP.*

Proof. As above, we prove the claim by reducing Mep to RecMax under the Item-based method that uses weighted average along with adjusted cosine similarity (see Eq. 9.3 and 9.4). Consider an instance $\mathcal{I}$ of Mep consisting of an undirected graph $G = (V, E)$. Create an instance $\mathcal{J}$ of RecMax as follows. Each node $v \in V$ corresponds to a user. Create $|E|$ items in instance $\mathcal{J}$, one for each edge in instance $\mathcal{I}$. An item i_{uv} corresponding to edge $(u, v) \in E$ is rated by users u and v with rating 2. Add a special item j such that each user rates it with rating 1. Thus, in instance $\mathcal{J}$, we have $|V|$ users, $|E| + 1$ items and $2 \cdot |E| + |V|$ ratings. We will decide the values of rating thresholds later.

Let $S \subseteq V$ be any seed set. Like above, we claim that a node $v \in V \setminus S$ will be in $H(S)$ (in instance $\mathcal{J}$) iff v is encircled by S (in instance $\mathcal{I}$). The proof then follows from the same logic used in the proof of Theorem 15.

We now show the above claim. Say, each seed user provides a rating 2 to the new item. Then, for a user $u \in S$, $\bar{R}_u = (2 \cdot deg(u) + 1 + 2)/(deg(u) + 2)$, as it provides rating 2 to all the items that correspond to the edges in $\mathcal{I}$ and 1 to the special item j, in addition to rating 2 to the new item ρ. Considering the adjusted cosine similarity function among items (see Eq. 9.4), for any item i that corresponds to an edge in instance $\mathcal{I}$, the deviation term is $2 - (2 \cdot deg(u) + 3)/(deg(u) + 2) = 1/(deg(u) + 2)$. Let us call this x_u. The similarity of i with

ρ would then be

$$w(i,\rho) = \frac{\sum_{u\in U} x_u \cdot x_u}{\sqrt{\sum_{u\in U} x_u^2} \cdot \sqrt{\sum_{u\in U} x_u^2}} = 1$$

where $U = V(i) \cap V(\rho) = V(i) \cap S$. Similarly, for the special item j, the deviation term is $y_u = 1 - (2 \cdot deg(u) + 3)/(deg(u) + 2) < 0$. It implies that the similarity between j and ρ is strictly less than 0. Let it be z. Thus, from Eq. 9.3, the expected rating of a user v on item ρ is

$$\hat{R}(v,\rho) = \frac{\sum_{i\in I(v)\cap I(S)} w(i,\rho) \cdot R(v,i)}{\sum_{i\in I(v)\cap I(S)} abs(w(i,\rho))} \tag{9.6}$$

where $I(S) = \bigcup_{u\in S} I(u)$ is the set of items rated by any seed user in S. Note that if an item i is not rated by any user in S, then it cannot be among the neighbors of new item ρ because ρ is rated only by users in S. That's why we iterate over all items $i \in I(v) \cap I(S)$ in Eq. 9.6. Moreover, $|I(v)| = 1 + deg(v)$. Clearly, the special item j is in $I(S)$, as all users rate j. Next, any item $i \neq j$, if added to $I(S)$, cannot decrease $R(v,\rho)$ as all such items are rated with rating 2. Hence, the maximum possible predicted rating that v could give the new item ρ is realized when all the items rated by v are in $I(S)$, i.e., $I(v) \subseteq I(S)$. Following Eq. 9.6, this maximum rating is

$$R_{max}(v) = \frac{2 \cdot deg(v) - z}{deg(v) + z}$$

Thus, we can set θ_v as $R_{max}(v) - \delta$ where δ is an extremely small number which ensures that the item ρ is recommended to v if and only if the predicted rating $\hat{R}(v,\rho) = R_{max}$. It means that the user $v \in H(S)$ if and only if $I(v) \subseteq I(S)$. However, an item $i_{uv} \in I(v) \setminus \{j\}$ can be in $I(S)$ if and only if the neighbor u of v is in S (because only u and v have rated the item i_{uv}). Thus, $I(v) \subseteq I(S)$ iff all neighbors of v are in S, i.e., iff S encircles v. This was to be shown. □

It should be noted that a slight modification in the reduction shown above shows that the RECMAX under Item-based method (that uses weighted average) is as hard as MEP even when the similarity function is cosine or Pearson. We omit the details for brevity.

Discussion. By now, it is clear that the RECMAX problem cannot be approximated within any reasonable factor, for the specific settings we focus on in this chapter. Many real RS are based on the User-based and Item-based methods considered in our settings of RECMAX and thus our results apply to RECMAX over those systems. What can we say about RS that are based on the hybrid of content-based and collaborative filtering, on matrix factorization etc.? Given the complexities of systems based on such methods, it is very likely that RECMAX continues to be hard (to approximate) on these systems. The main reason behind this hardness of RECMAX is its similarity with MEP: in order to push an item to the recommendation list of a user, several other users may need to rate the item. This has a flavor similar to the user being "encircled" by other users. Since the hit score function is neither monotone nor submodular, as noted in the beginning of the section, it is not clear a priori which heuristics

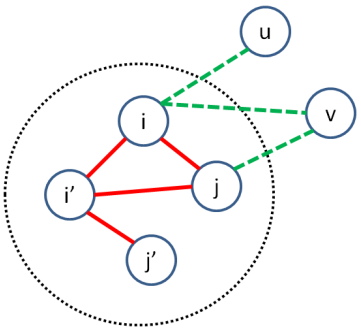


Figure 9.2: Example.

will work well for RECMAX. So, the question arises: What can we do with RECMAX? Does it still have a case? As we show in Section 9.4, the answer is yes. Even if the problem is too hard to approximate, it still makes a good sense for targeted marketing, as the gains (in terms of hit score) achieved are impressive, even with the RANDOM heuristic. Clearly, we cannot develop an algorithm with theoretical guarantees, but we can try several heuristics and see what works best and under what conditions – which is what we do in the next section.

9.6 Experiments

Algorithms. Given that RECMAX cannot be approximated within any reasonable factor, in this section we explore several natural heuristics, and then gauge their performance on 3 real datasets – Movielens, Yahoo! Music and Jester– and analyze the results. Let k be the budget on seed set size. We explore the following heuristics in our evaluation.

RANDOM: Seed set is selected randomly. The process is repeated 100 times and the average hit score of different choices is taken. We take this as a baseline.

MOST-ACTIVE: Choose the top-k users with most number of ratings. The intuition here is that the users who are most active "control" most of the ratings in the system, and thus can be seen as a good target for marketing.

MOST-POSITIVE: Choose the top-k users with most positive average ratings as seeds. It is another natural way for targeted marketing, as many manufacturers may want to avoid bad ratings/opinions about their product. Thus, targeting positive users is a "safe" marketing choice.

MOST-CRITICAL: Choose the top-k users with most critical average ratings. This is another extreme, where it may be argued that if the most critical users endorse a product, the chances of success (i.e., large hit score) are high.

MOST-CENTRAL: For each user, we compute its similarity with every other user. Then, we compute its aggregate similarity score $agg(u) = \sum_{v \in V} sim(u, v)$. Next, we select top-$k$ users with highest aggregate similarity scores as the seed set. In case of User-based, we compute the similarity according to the Pearson correlation, which is what is used in estimating predicted ratings.

For Item-based, the recommendations are generated using similarities among items, and

	Movielens	Yahoo	Jester
#Users	6040	10K	25K
#Items	3706	5069	100
#Ratings	1M	1M	1.8M
Mean	3.58	51.9	10.88
Std. Dev.	1.12	39.9	5.23

Table 9.1: Dataset statistics

not users. Hence, it is not clear how to compute similarities among users in case of Item-based. For instance, consider the example shown in Fig. 9.2. It has 4 items i, j, i', j', and we know the weights on the red edges: they are computed according to Eq. 9.4. No two users are directly connected in Fig. 9.2. Hence, to compute the similarity between users u and v, we take the weighted average of ratings provided by the users u and v on every pair of items they rate, as follows.

$$sim(u, v) = \frac{\sum_{i \in I(u), j \in I(v)} w(i, j) \cdot sim(R(u, i), R(v, j))}{\sum_{i \in I(u), j \in I(v)} w(i, j)}$$

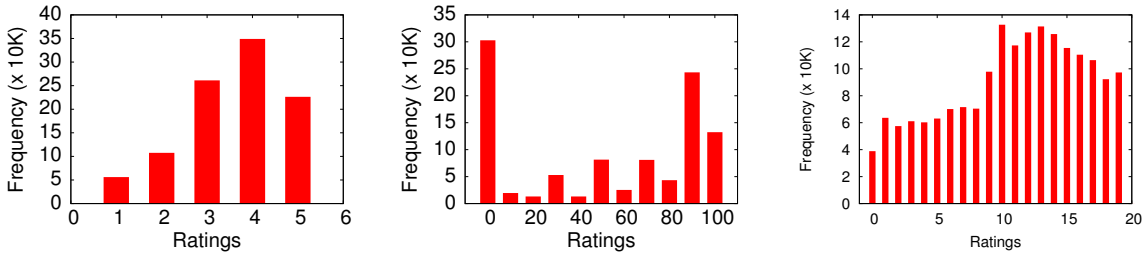
where $sim(R(u, i), R(v, j))$ is the similarity between the ratings provided by user u on item i and user v on item j. If users u and v rate a common item i, i.e., $i = j$, then, $w(i, j) = w(i, i) = 1$. Intuitively, if the ratings are nearly equal, $w(R(u, i), R(v, j))$ should be close to 1 and if the ratings are very different, the similarity score should be close to 0. One way to define it is

$$sim(R(u, i), R(v, j)) = 1 - \frac{abs(R(u, i) - R(v, j))}{(R_{max} - R_{min})}$$

where R_{max} and R_{min} are the maximum and minimum ratings in the system and $abs(\cdot)$ is the absolute value function.

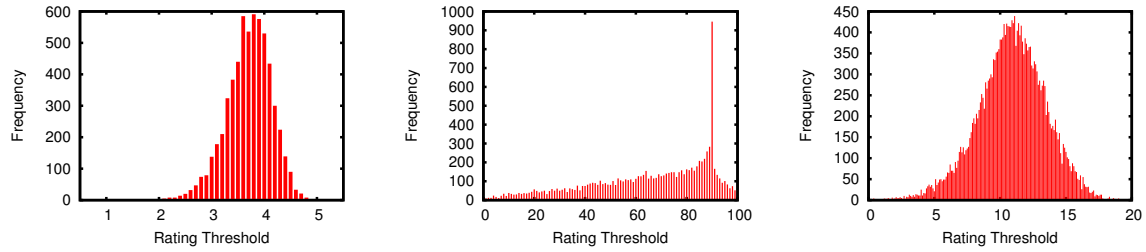
Datasets. We run experiments on three real world datasets, whose statistics are given in Fig. 9.1. The first dataset is Movielens. It consists of 1M ratings given by 6040 users distributed over 3706 movies. The ratings are on a scale of 1-5. Fig. 9.3(a) shows the distribution of ratings which follows the normal distribution with mean 3.58 and standard deviation 1.12. Next, we use Yahoo! Music[56] data set, provided as part of the Yahoo! Research Alliance Webscope program. It represents a sample of Yahoo! users' ratings of musical artists, gathered over a thirty-day period sometime prior to March 2004. The raw data contains 11.5M ratings over 98K artists given by 1.9M users. The ratings are integers ranging from 0 to 100, except 255 (a special rating meaning "never play again"). From this data set, we randomly sampled 10K users who assigned 20 or more ratings. From the sample, we discarded all the artists who received less than 20 ratings. This pre-processing resulted in 1M ratings over 5069 artists. We ignored the special rating 255 in the pre-processing and kept only explicit ratings. The rating distribution of the resulting data set is presented in Fig. 9.3(b). Notice that unlike in Movielens, there is a large number of ratings with value 0 in Yahoo!. Finally, we use the

[56] `www.music.yahoo.com`



(a) Movielens Ratings Distribution (b) Yahoo! Music Ratings Distribution (c) Jester Ratings Distribution

Figure 9.3: (a) Summary of datasets; (b)-(d) Distributions of ratings.



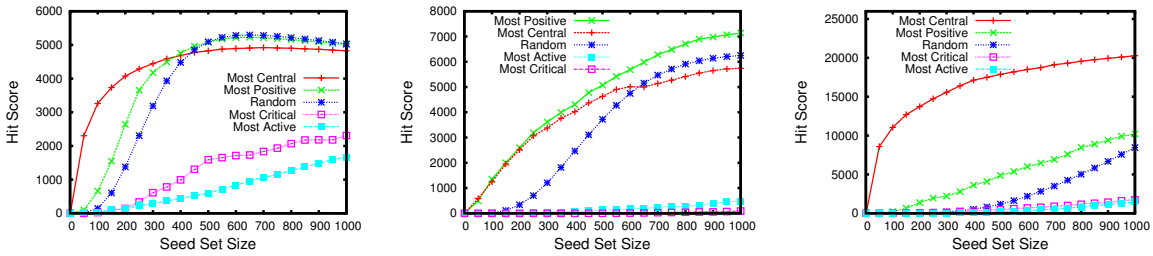
(a) Movielens users' ratings thresholds (b) Yahoo! users' ratings thresholds (c) Jester users' ratings thresholds

Figure 9.4: Distributions of users's rating thresholds.

Jester Joke collaborative filtering data set released by Ken Goldberg from UC Berkeley[57][50]. It has 100 jokes rated by 25K users. There are a total of 1.8M ratings, on a continuous scale of -10 to 10. Since we use the weighted average to compute expected ratings (according to Eq. 9.1 and 9.3) which is meant for positive ratings, we shift the ratings from the $[-10, 10]$ scale to $[0, 20]$, by adding 10 to each rating. The ratings distribution is presented in Fig. 9.3(c). Again, unlike the movie ratings in Movielens, there is a significant number of negative ratings in Jester. In our experiments, we derive a rating $R(u, \rho)$ for a seed user u by taking an average of the top-20% of the highest ratings provided by u. Finally, while computing $\hat{R}(v, \rho)$, we ignore the small changes in similarity values among users (in case of User-based) and existing items (in case of Item-based) that happen because of ratings provided by seed users to the new item ρ. We evaluate our heuristics for RecMax on both User-based and Item-based collaborative filtering methods.

User-Based collaborative filtering. We first report our findings in User-based systems. Predicted ratings are computed using weighted average along with Pearson correlation (see Eq. 9.1 and 9.2). For $\hat{R}(v, i)$ to be non-zero, we require that at least 5 of v's nearest neighbors must rate the item i, i.e., $|N_i(v)| \geq 5$. Similarly, for the similarity between users u and v to be non-zero, we require them to rate at least 5 items in common, i.e., $|I(u) \cap I(v)| \geq 5$. Finally, we consider at most 25 nearest neighbors of v while computing $\hat{R}(v, i)$. These settings are

[57] `eigentaste.berkeley.edu/user`

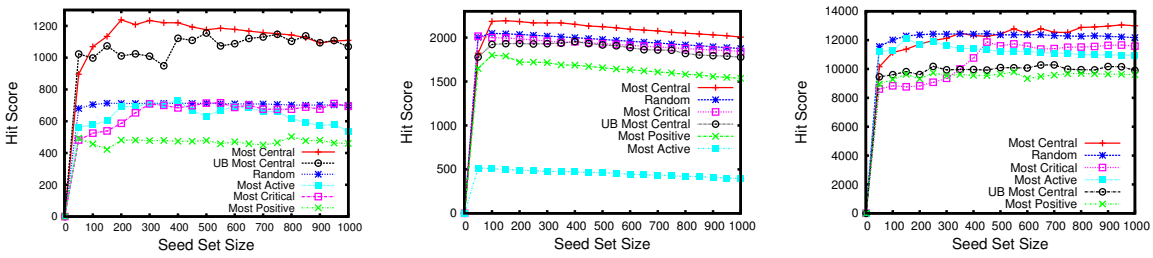


(a) Movielens (b) Yahoo! (c) Jester

Figure 9.5: Hit Score achieved with various algorithms on different datasets on User-based RS.

used and recommended by previous works (E.g., see [67, 108]). On Movielens and Yahoo!, we set the length of the recommendation list ℓ as 15, as popular recommendation music/movies systems like Yahoo! Music, Movielens, Youtube usually show 15-20 recommendations on a single page. On Jester, ℓ is set to 1 because only one joke is recommended at a time on the Jester website. Based on these settings, we compute the rating thresholds θ_v of every user v. The distributions of the rating thresholds for the three data sets are shown in Fig. 9.4.

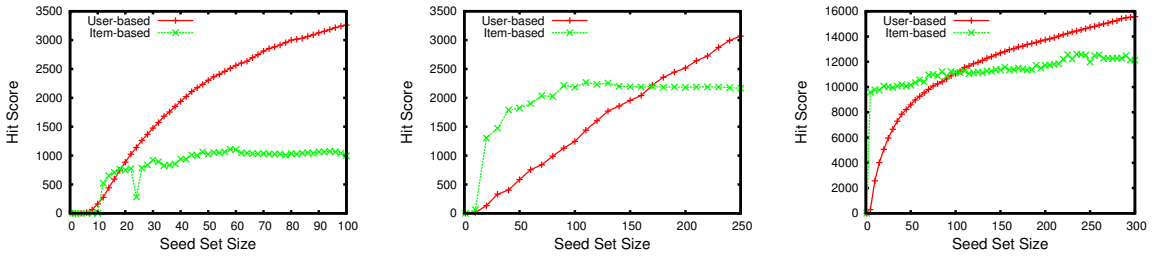
Fig. 9.5 compares the various heuristics, described above, on all 3 datasets. The plots reveal many interesting points. (i) On all 3 datasets, we see that Most-Active and Most-Critical perform poorly, and Most-Positive and Most-Central perform relatively better. Thus, simply choosing users who provide lots of ratings or users who are very critical does not seem to help. Surprisingly, even Random exhibits a relatively good performance. For instance, on Yahoo! (Fig. 9.5(b)), with a budget of 500, Random is able to achieve a hit score of 3719. With the same budget, Most-Central and Most-Positive achieve 4628 and 5073, that is, a 24.4% and a 36.4% improvement over Random, respectively. By contrast, Most-Active and Most-Critical achieve poor hit scores of only 132 and 2 respectively. This clearly establishes that *the choice of seed sets is critical and can make a substantial difference.* Except on Yahoo! where Most-Positive wins, Most-Central is found to be the best. Most-Central picks seeds who have the greatest affinity to other users in the aggregate and in the context of User-based collaborative filtering, this does make sense. (ii) In general, the hit score achieved is impressive. For instance, with a budget of only 300, Most-Central attains a hit score of 4444, 3377 and 15.6K on Movielens, Yahoo! and Jester respectively. It shows even the simple heuristic strategies for seed selection make Recmax relevant and useful for targeted marketing, despite the problem being inapproximable within any reasonable factor. (iii) Depending on the data set and the seed set selected, we may encounter a "tipping point", a point at which a slight increase in budget can make a significant difference in the hit score achieved. For example, on Movielens, using Most-Positive, the hit score with a budget of 50 is only 288, i.e., the average hit score per seed is 5.76. On the other hand, the overall hit score shoots up to 3656 with a the budget of 250, i.e., the average hit score for the last 200 seeds is 16.84. The rate of growth in hit score slows down as the budget is increased beyond a point. E.g., when the budget is extended futher by 200 to 450, the hit score increases to 4962, that is, the average hit score for the last 200 seeds drops to 6.53. When we continue to expand



(a) Movielens (b) Yahoo! (c) Jester

Figure 9.6: Hit Score achieved with various algorithms on different datasets on Item-based RS.

the budget, the hit score plateaus and in some cases it may even decrease. For instance, on Movielens with MOST-POSITIVE, if we increase the budget from 700 to 1000, the hit score falls from 5215 to 5008.



(a) Movielens (b) Yahoo! (c) Jester

Figure 9.7: Comparison of User-based and Item-based RS with the algorithm MOST-CENTRAL.

Item-Based collaborative filtering. We now report our findings in Item-based collaborative filtering. Again, to compute the expected rating $\hat{R}(v,i)$, we use the weighted average with similarities among items computed using adjusted cosine (see Eq. 9.3 and 9.4). As earlier, we require for $\hat{R}(v,j) > 0$, that there be at least 5 similar items to j that are rated by v, that is, $|N_v(j)| \geq 5$. Similarly, the similarity among items i and j can be more than 0 only if they are rated by at least 5 common users, i.e., $|V(i) \cap V(j)| \geq 5$. Finally, we consider at most 25 most similar items to j to compute $\hat{R}(v,j)$. We keep the length of the recommendation list ℓ the same as before, that is, 15, 15 and 1 for Movielens, Yahoo! and Jester respectively. Based on these settings, the rating thresholds of all users are computed. We found their distributions to be very similar to those for User-based (see Fig. 9.4) and so suppress their plots for brevity.

In Fig. 9.6, we show the hit score achieved by the various heuristics. Note that the MOST-CENTRAL heuristic is fundamentally different in Item-based. Out of curiosity, we also include in our comparison the MOST-CENTRAL heuristic of User-based method. For clarity, we refer to the latter as UB-MOST-CENTRAL in Fig. 9.6. Again, many interesting observations are made. (i) On all 3 datasets, the overall hit score that is achieved in Item-based is much lower than in User-based. Moreover, on all the datasets, the hit score plateaus much faster in Item-

based than in User-based. For example, on Movielens with Most-Central heuristics, the hit score saturates at 1238 at the budget of 200 in case of Item-based. In User-based, on the other hand, the hit score saturates at 4825 at the budget of 500 (see Fig. 9.5(a)). As a result, much less seeding is required to achieve the maximum possible hit score in Item-based. (ii) Except in Jester, Most-Central beats every other algorithm. For instance, on Movielens, with a budget of 200, while Most-Central gets the hit score of 1238, Random achieves just 712. Most-Active, Most-Critical and Most-Positive attain the scores 692, 587 and 481 respectively. (iii) Another interesting observation is that while Most-Positive performs quite well in User-based, it performs poorly in Item-based. This shows strategies that work well in User-based need not work well in Item-based. However, interestingly enough, the intuition underlying Most-Central seems to be borne out well by the results for both User-based and Item-based.

Next, we undertake a comparison of User-based with Item-based. Fig. 9.7 presents a zoomed-in comparison of hit scores achieved in User-based and Item-based methods, with the heuristic Most-Central. Except in Movielens, the initial rise of hit score is much steeper in Item-based than in User-based. And as the budget increases, hit score in User-based continues to increase well beyond the peak value of Item-based and it plateaus at a much higher budget than Item-based. Finally, we look at the intersection of seed sets obtained from Most-Central heuristic in both cases. Out of 1000 seeds, the number of common seeds are 103, 219 and 62 on Movielens, Yahoo! and Jester, respectively, suggesting that the seed sets are different in both methods.

In summary, the hit scores behave very differently in User-based and Item-based methods. While the overall hit scores that are achieved in User-based are much higher, the saturation in hit scores is achieved much faster in case of Item-based. In both, respective versions of Most-Central are found to be most effective, although other heuristics behave differently. For instance, Most-Positive performs quite well in User-based, while its performance in Item-based is poor. These observations suggest that different approaches are required to select good seed sets on different systems, and no single rule of thumb works well for all systems.

9.7 Conclusions and Future Work

The main goal of this work is to propose and study a novel problem that we call Recmax. It aims to develop a technology to select seed users in a collborative filtering based RS such that if they endorse a new product with relatively high ratings, the new product is recommended to a large number of users, solely because of the underlying mechanics of the RS. We motivate the problem with real world applications. We focus on the two widely used methods – User-based and Item-based. We perform a thorough theoretical analysis and show that Recmax is not only NP-hard to solve optimally, it is NP-hard to approximate within any reasonable factor. Given that, we explore various natural heuristics and show that, even though the problem is inapproximable, simple heuristics like Most-Central can fetch impressive gains. This makes Recmax an interesting problem for targeted marketing in RS.

This work opens up a wide array of challenges. First, developing more effective heuristics is important, interesting and challenging, given that the hit score function is neither monotone

nor submodular. We also need to calibrate the proposed and new heuristics on a wide array of real data sets. Second, as we saw in our empirical evaluation, the hit score function behaves distinctly on User-based and Item-based systems. It would be interesting to see how it behaves when mixed approaches are employed. Third, studying RecMax on more recent RS methodologies, for example, Matrix factorization [128] – both theoretically and empirically, would be exciting. Finally, the vision behind RecMax is marketing based on an operational recommender *system*, which is not just a simple RS algorithm but has a much greater complexity. Formulating and solving RecMax and launching it on an operational RS is a fascinating challenge with great potential.

Acknowledgments. We thank Suresh Venkatasubramanian and Nick Harvey for helpful discussions. We acknowledge Yun Lou's involvement in the initial phase of the project. We appreciate anonymous reviewer's feedback that helped us in improving the presentation of the chapter.

Chapter 10

Conclusions and Future Work

While the study of social influence has long attracted the attention of psychologists and marketing scientists, it became popular among computer scientists in the last decade only. The study has been facilitated by massive amounts of data that are available today as a result of the emergence of Web 2.0 and social media platforms like Facebook and Twitter. Many applications like viral marketing (or targeted advertising in general), recommender systems, analysis of information diffusion in Social Media like Twitter and Facebook, events detection, experts finding, link prediction and ranking of feeds can benefit from social influence analytics. The vision of this dissertation is to investigate social influence and its applications from an algorithmic perspective. One of the fundamental problems in the study of social influence is Influence Maximization, motivated by the application of viral marketing. The idea is to identify a small set of users in a social network, who when convinced to adopt a product will influence others in the network leading to a large number of adoptions in an expected sense. In this dissertation, we identify several limitations and open problems in previous studies and make an attempt to address them.

Existing solutions to Influence Maximization are not scalable – even selecting a seed set of size 50 can take up to months in a relatively small network consisting of just 20,000 users. We propose two algorithms – Celf++ and Simpath that improve the efficiency of existing solutions. Celf++ is an optimization and has no trade-off with the quality of seed sets obtained. It can be applied to any propagation model in which the objective influence spread function is monotone and submodular (like independent cascade and linear threshold models). We show that Celf++ improves the efficiency of existing Celf optimization by 17-61%. Simpath on the other hand, is a heuristic and designed specifically for linear threshold model. Using a parameter δ, we can strike a balance between running time and desired quality of the seed set. We show that Simpath outperforms the state of the art algorithm Ldag on all three fronts – running time, memory consumption and the quality of the seed set chosen (Chapter 3).

The framework employed in previous works requires two kinds of data – a directed graph and an assignment of probabilities (or weights) to the edges of the graph, capturing degrees of influence. In real life, while the graph representing social network is often explicitly available, edge probabilities are not. To mitigate this, previous works have resorted to simply making arbitrary assumptions about these probabilities and the question of learning influence probabilities from real world data has largely been ignored. We attack the problem of learning influence probabilities. One of our key findings is that the probabilities are not static and change over time, and that as a result, existing influence models are not suitable (Chapter 4).

Most of the previous works revolve around two classical propagation models – independent cascade and linear threshold. Both the models, although very popular, have never been

tested against real world datasets. We perform a large scale experimental study to test various propagation models against real world datasets. We found both independent cascade and linear threshold models grossly overpredict the expected spread of influence in most cases. On account of this, we develop a data driven framework for the problem of Influence Maximization. In particular, we introduce a new model, which we call credit distribution, that directly leverages available propagation traces to learn how influence flows in the network and uses this to estimate expected influence spread. Our approach also learns the different levels of influenceability of users, and it is time-aware in the sense that it takes the temporal nature of influence into account. We show that credit diffusion model is significantly more accurate than independent cascade or linear threshold model in modeling influence propagation (Chapter 5).

The essence of Influence Maximization is to identify high quality seed users and recommend them to the marketers (decision makers) who would target them for the marketing of the given new product (e.g., by providing free samples or discounts). These marketers may wish to optimize other objectives, instead of maximizing influence. To this end, we study alternative problem formulations – Mintss and Mintime and show interesting theoretical results. These problem formulations capture the problem of deploying viral campaigns on budget and time constraints. We show that the simple greedy algorithm continues to provide optimal seed sets that can be achieved in polynomial time (Chapter 6).

Likewise, before spending money, the marketer may wish to examine various solution frameworks instead of just the "optimization" framework. We study the problem of identifying community leaders using a pattern mining framework. We define a *leader* as the user whose sufficient number of actions (captured by parameter σ) are being followed by sufficiently many users (captured by parameter ψ) within a sufficiently small time window (captured by parameter π). We found that over many actions, for a leader, the set of followers overlaps, and in a sense forms "tribe". We call such leaders as "tribe leaders" and the set of followers its tribe. Our framework is intuitive and allows the marketers to dig deeper into the leadership qualities of the leaders. For instance, consider a leader. A marketer may want to look at the following questions: on what users the leader is influential? On how many actions (or different types of actions) she is influential? Is there a "tribe" that follows her actions? Our framework facilitates answering these questions (Chapter 7).

Finally, we examine the applications of social influence. First, we look at the application of viral marketing. Using real world datasets, we show that product adoption is not exactly influence, and argue that while influence spreads in an epidemic-like manner, the actual adoption depends on other factors such as price and individual's valuation of the product. Given this, we develop a product adoption model and study the problem of maximizing product adoption. While models like linear threshold (or independent cascade) predict the same spread for a product if the seed set and influence weights remain same, our model takes into account the experience of the users. As we show, this significantly improves the accuracy of spread prediction. We also report several interesting findings. We found that there is a positive correlation between the number of initiators (users who first express opinions among their friend circles) and the final spread, suggesting that the network structure plays a significant role in the spread. We demonstrate that the choice of seed nodes is critical and can make a significant difference to the spread achieved. We also found that largely, the seed sets obtained from our algorithm are different for different products (which we wish to market), while some influential

users are picked up consistently as seeds irrespective of the product being marketed (Chapter 8).

While influence (or information) flows primarily through explicit social/friendship links in social networks, with the ever-increasing incorporation of recommender systems in social networks, the influence also flows through recommender systems models. Thus, it is important to take into account these information channels as well. To this end, we take the first step in this dissertation and propose a novel problem that we call RECMAX (short for recommendation maximization): Given a collaborative recommender system, a budget of k and a new product ρ, find k users (called seed set) to whom the new product ρ should be marketed such that if they endorse the product by providing relatively high ratings, the number of other users to whom the product is recommended by the underlying recommender system algorithm is maximum. Note that the objective of RECMAX is different from INFLUENCE MAXIMIZATION and hence it is a different problem. It captures the notion of viral campaign over recommender systems. We motivate RECMAX with real world applications. We show that seeding can make a substantial difference to the number of users to whom a product is recommended, if done carefully. We prove that RECMAX is not only NP-hard to solve optimally, it is NP-hard to even approximate within any reasonable factor. Given this hardness, we explore several natural heuristics on 3 real world datasets and report our findings. We show that even though RECMAX is hard to approximate, simple natural heuristics may provide impressive gains, for targeted marketing using recommender systems (Chapter 9).

10.1 Future Work

This dissertation has made substantial progress in the study of social influence and its applications. Still, several challenges remain. First, it remains unclear, under what conditions the classical models like independent cascade and linear threshold models are applicable. We test these models against two real world datasets – Flickr and Flixster and find that they grossly over-predict the spread. However, we should be careful before discarding these models, as the prediction heavily depends on the influence probabilities. A systematic testing of these models against a large number of data sets is required before we can draw any final conclusions about their performance. We observe that the real world datasets are very sparse and hence, the learning methods are vulnerable to over-fitting, as we report in Chapter 5. Therefore, more sophisticated techniques are needed for learning influence probabilities. Similarly, it is important to test these techniques and models on many more datasets.

The client to INFLUENCE MAXIMIZATION solutions is a marketer who would target the seed users for the marketing of the given new product (e.g., by providing free samples or discounts). It is natural that the marketer would like to analyze the influence spread of these seed users before actually committing resources required for targeting. She would be interested in the answers of the following questions: Where exactly the influence of a certain seed user lies? On what actions (products) the user is influential? What are the demographics of its followers? However, the current INFLUENCE MAXIMIZATION approaches largely work as a black-box and just output the seed set, with an estimate on expected influence spread. Our works in Chapter 5 and 7 facilitate a deeper exploration of these questions. Still, a more principled approach

can be developed.

Next, it is interesting to ask what makes a product viral: does it solely depend on marketing or does it also depend the product itself? A recent paper by Aral et al. [9] investigated how incorporating viral features in the product itself helps in making a product viral. They show that viral product design features can result in identifiable peer influence and social contagion effects. Hence, it is important to take into account the product (or content of action being propagated) while predicting the spread. Our work in Chapter 8 accounts for this "indirectly" by incorporating user opinions as ratings. But, the assumption we make there is that there must exist some ratings on the product that we want to market. Lifting this assumption would be interesting. It should be noted that prediction the spread is inherently a difficult problem as it depends on many factors – marketing, design of products, external factors, individual factors etc, particularly so for new products on which few ratings exist.

Recall that in Chapter 9, we study a novel problem RECMAX and explore several natural heuristics showing that even these heuristics provide substantial gains. Developing more effective heuristics is interesting and challenging. Moreover, studying RECMAX on more recent approaches to recommender systems, for example, Matrix factorization – both theoretically and empirically, would be exciting. Finally, incorporating both social networks and recommender systems in an integrated study of information/influence flow via multiple channels is an exciting future work.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17, 2005.

[2] Nitin Agarwal, Huan Liu, Lei Tang, and Philip Yu. Modeling blogger influence in a community. *Social Network Analysis and Mining*, pages 1–24, 2011. 10.1007/s13278-011-0039-3.

[3] 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*, WSDM '08, pages 207–218, New York, NY, USA, 2008. ACM.

[4] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[5] Chuck Allison. Bit handling in c++, part 1. *C Users Journal*, 11(12):71–90, 1993.

[6] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 7–15, New York, NY, USA, 2008. ACM.

[7] Sarabjot Singh Anand and Nathan Griffiths. A market-based approach to address the new item problem. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 205–212, New York, NY, USA, 2011. ACM.

[8] Sinan Aral, Lev Muchnik, and Arun Sundararajan. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences*, 2009.

[9] Sinan Aral and Dylan Walker. Creating social contagion through viral product design: A randomized trial of peer influence in networks. *Manage. Sci.*, 57(9):1623–1639, September 2011.

[10] David Arthur, Rajeev Motwani, Aneesh Sharma, and Ying Xu. Pricing strategies for viral marketing on social networks. In *Proceedings of the 5th International Workshop on Internet and Network Economics*, WINE '09, pages 101–112, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 44–54, New York, NY, USA, 2006. ACM.

[12] Lars Backstrom, Ravi Kumar, Cameron Marlow, Jasmine Novak, and Andrew Tomkins. Preferential behavior in online groups. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 117–128, New York, NY, USA, 2008. ACM.

[13] Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 65–74, New York, NY, USA, 2011. ACM.

[14] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. Generalized submodular cover problems and applications. *Theor. Comput. Sci.*, 250(1-2):179–200, January 2001.

[15] F. Bass. A new product growth model for consumer durables. *Management Science*, 15:215–227, 1969.

[16] Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. An exact almost optimal algorithm for target set selection in social networks. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pages 355–362, New York, NY, USA, 2009. ACM.

[17] Smriti Bhagat, Amit Goyal, and Laks V. S. Lakshmanan. Maximizing product adoption in social networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 603–612, New York, NY, USA, 2012. ACM.

[18] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In Xiaotie Deng and Fan Graham, editors, *Internet and Network Economics*, volume 4858 of *Lecture Notes in Computer Science*, pages 306–311. Springer Berlin / Heidelberg, 2007.

[19] Joe M. Bohlen and George M. Beal. The diffusion process. *Spl. Report No. 18, Agri. Extn. Serv., Iowa State College*, 1957.

[20] Francesco Bonchi. Influence propagation in social networks: A data mining perspective. *IEEE Intelligent Informatics Bulletin*, 12(1):8–16, 2011.

[21] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 11–, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Allan Borodin, Yuval Filmus, and Joel Oren. Threshold models for competitive influence in social networks. In *Proceedings of the 6th international conference on Internet and network economics*, WINE'10, pages 539–550, Berlin, Heidelberg, 2010. Springer-Verlag.

[23] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[24] Justus Bross, Keven Richly, Matthias Kohnen, and Christoph Meinel. Identifying the top-dogs of the blogosphere. *Social Network Analysis and Mining*, pages 1–15, 2011. 10.1007/s13278-011-0027-7.

[25] Cristian Bucila, Johannes Gehrke, Daniel Kifer, and Walker White. DualMiner: A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7:241–272, 2003.

[26] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 665–674, New York, NY, USA, 2011. ACM.

[27] Tianyu Cao, Xindong Wu, Tony Hu, and Song Wang. Active learning of model parameters for influence maximization. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6911 of *Lecture Notes in Computer Science*, pages 280–295. Springer Berlin / Heidelberg, 2011.

[28] Tim Carnes, Chandrashekhar Nagarajan, Stefan M. Wild, and Anke van Zuylen. Maximizing influence in a competitive social network: a follower's perspective. In *Proceedings of the ninth international conference on Electronic commerce*, ICEC '07, pages 351–360, New York, NY, USA, 2007. ACM.

[29] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, MDMKDD '10, pages 4:1–4:10, New York, NY, USA, 2010. ACM.

[30] Meeyoung Cha, Hamed Haddadi, Fabrício Benevenuto, and P. Krishna Gummadi. Measuring user influence in twitter: The million follower fallacy. In *ICWSM*, 2010.

[31] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 721–730, New York, NY, USA, 2009. ACM.

[32] Meeyoung Cha, Juan Pérez, and Hamed Haddadi. The spread of media content through blogs. *Social Network Analysis and Mining*, pages 1–16, 2011. 10.1007/s13278-011-0040-x.

[33] Ning Chen. On the approximability of influence in social networks. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 1029–1037, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[34] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *SDM*, 2011.

[35] 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*, KDD '10, pages 1029–1038, New York, NY, USA, 2010. ACM.

[36] 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*, KDD '09, pages 199–208, New York, NY, USA, 2009. ACM.

[37] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 88–97, Washington, DC, USA, 2010. IEEE Computer Society.

[38] J. Coleman, H. Menzel, and E. Katz. *Medical Innovations: A Diffusion Study*. Bobbs Merrill, 1966.

[39] David Crandall, Dan Cosley, Daniel Huttenlocher, Jon Kleinberg, and Siddharth Suri. Feedback effects between similarity and social influence in online communities. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 160–168, New York, NY, USA, 2008. ACM.

[40] Peng Cui, Fei Wang, Shaowei Liu, Mingdong Ou, Shiqiang Yang, and Lifeng Sun. Who should share what?: item-level social influence prediction for users and posts ranking. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 185–194, New York, NY, USA, 2011. ACM.

[41] Morton Deutsch and Harold B. Gerard. A study of normative and informational social influences upon individual judgement. *Journal of Abnormal & Social Psychology*, 51(3):629–36, 1955.

[42] 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*, KDD '01, pages 57–66, New York, NY, USA, 2001. ACM.

[43] Wen Dong and Alex Pentland. Modeling influence between experts. In *Proceedings of the ICMI 2006 and IJCAI 2007 international conference on Artifical intelligence for human computing*, ICMI'06/IJCAI'07, pages 170–189, Berlin, Heidelberg, 2007. Springer-Verlag.

[44] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[45] Lisa Friedland and David Jensen. Finding tribes: identifying close-knit individuals from employment patterns. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 290–299. ACM, 2007.

[46] Toshihiro Fujito. On approximation of the submodular set cover problem. *Operations Research Letters*, 25(4):169 – 174, 1999.

[47] Toshihiro Fujito. Approximation algorithms for submodular set cover with applications. *IEICE Trans. Inf. Syst*, 83, 2000.

[48] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[49] Jennifer Golbeck and James Hendler. Inferring binary trust relationships in web-based social networks. *ACM Trans. Internet Technol.*, 6(4):497–529, 2006.

[50] K. Y. Goldberg et al. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2), 2001.

[51] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001.

[52] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML'11*, pages 561–568, 2011.

[53] 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*, KDD '10, pages 1019–1028, New York, NY, USA, 2010. ACM.

[54] Neil Zhenqiang Gong, Ameet Talwalkar, Lester W. Mackey, Ling Huang, Eui Chul Richard Shin, Emil Stefanov, Elaine Shi, and Dawn Song. Predicting links and inferring attributes using a social-attribute network (san). *CoRR*, abs/1112.3265, 2011.

[55] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. Discovering leaders from community actions. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 499–508, New York, NY, USA, 2008. ACM.

[56] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 241–250, New York, NY, USA, 2010. ACM.

[57] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. A data-based approach to social influence maximization. *Proc. VLDB Endow.*, 5(1):73–84, September 2011.

[58] Amit Goyal, Francesco Bonchi, Laks V. S. Lakshmanan, and Suresh Venkatasubramanian. On minimizing budget and time in influence propagation over social networks. *Social Network Analysis and Mining*, pages 1–14, 2012.

[59] Amit Goyal and Laks V. S. Lakshmanan. Recmax: exploiting recommender systems for fun and profit. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 1294–1302, New York, NY, USA, 2012. ACM.

[60] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 47–48, New York, NY, USA, 2011. ACM.

[61] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. SimPath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 211–220, Washington, DC, USA, 2011. IEEE Computer Society.

[62] Amit Goyal, Byung-Won On, Francesco Bonchi, and Laks V. S. Lakshmanan. Gurumine: A pattern mining system for discovering leaders and tribes. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1471–1474, Washington, DC, USA, 2009. IEEE Computer Society.

[63] Daniel Gruhl, R. Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 491–501, New York, NY, USA, 2004. ACM.

[64] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 403–412, New York, NY, USA, 2004. ACM.

[65] Jason Hartline, Vahab Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 189–198, New York, NY, USA, 2008. ACM.

[66] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *FOCS*, 1996.

[67] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 230–237, New York, NY, USA, 1999. ACM.

[68] Shawndra Hill, Foster Provost, and Chris Volinsky. Network-based marketing: Identifying likely adopters via consumer networks. *Statistical Science*, 21(2):256Ű–276, 2006.

[69] John Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 541–546, New York, NY, USA, 2003. ACM.

[70] Jian Huang, Ziming Zhuang, Jia Li, and C. Lee Giles. Collaboration over time: characterizing and modeling network evolution. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 107–116, New York, NY, USA, 2008. ACM.

[71] Junming Huang, Xue-Qi Cheng, Hua-Wei Shen, Tao Zhou, and Xiaolong Jin. Exploring social influence via posterior effect of word-of-mouth recommendations. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 573–582, New York, NY, USA, 2012. ACM.

[72] Dino Ienco, Francesco Bonchi, and Carlos Castillo. The meme ranking problem: Maximizing microblogging virality. In *Proceedings of the SIASP 2010 workshop at ICDM 2010*.

[73] Raghuram Iyengar, Christophe Van den Bulte, and Thomas W. Valente. Opinion leadership and social contagion in new product diffusion. *Marketing Science*, 30(2):195–212, March 2011.

[74] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 135–142, New York, NY, USA, 2010. ACM.

[75] D. B. Johnson. Find all the elementary circuits of a directed graph. *J. SIAM, 4:77-84*, 1975.

[76] Shlomo Kalish. A new product adoption model with price, advertising, and uncertainty. *Management Science*, 31(12), 1985.

[77] George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4), 2009.

[78] 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*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[79] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proceedings of the 32nd international conference on Automata, Languages and Programming*, ICALP'05, pages 1127–1138, Berlin, Heidelberg, 2005. Springer-Verlag.

[80] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[81] Masahiro Kimura and Kazumi Saito. Approximate solutions for the influence maximization problem in a social network. In Bogdan Gabrys, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4252 of *Lecture Notes in Computer Science*, pages 937–944. Springer Berlin / Heidelberg, 2006.

[82] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *Proceedings of PKDD 2006, Lecture Notes in Computer Science, Volume 4213.*, 2006.

[83] Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Efficient estimation of influence functions for sis model on social networks. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 2046–2051, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[84] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*. 2011.

[85] Jan Kostka, Yvonne Anne Oswald, and Roger Wattenhofer. Word of mouth: Rumor dissemination in social networks. In *Proceedings of the 15th international colloquium on Structural Information and Communication Complexity*, SIROCCO '08, pages 185–196, Berlin, Heidelberg, 2008. Springer-Verlag.

[86] D. Kroft. All paths through a maze. *Proc. IEEE 55*, 1967.

[87] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 611–617, New York, NY, USA, 2006. ACM.

[88] Timothy La Fond and Jennifer Neville. Randomization tests for distinguishing social influence and homophily effects. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 601–610, New York, NY, USA, 2010. ACM.

[89] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, ICUIMC '08, pages 208–211, New York, NY, USA, 2008. ACM.

[90] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Trans. Web*, 1(1), May 2007.

[91] 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*, KDD '07, pages 420–429, New York, NY, USA, 2007. ACM.

[92] Inge Li Gørtz and Anthony Wirth. Asymmetry in k-center variants. *Theor. Comput. Sci.*, 361(2):188–199, 2006.

[93] L. Licamele and L. Getoor. Social capital in friendship-event networks. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 959 –964, dec. 2006.

[94] Lu Liu, Jie Tang, Jiawei Han, Meng Jiang, and Shiqiang Yang. Mining topic-level influence in heterogeneous networks. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 199–208, New York, NY, USA, 2010. ACM.

[95] V. Mahajan, E. Muller, and F. Bass. New product diffusion models in marketing: A review and directions for research. *Journal of Marketing*, 54(1):1–26, 1990.

[96] Puneet Manchanda, Ying Xie, and Nara Youn. The role of targeted communication and contagion in product adoption. *Marketing Science*, pages 961–976, 2008.

[97] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 529–537, New York, NY, USA, 2011. ACM.

[98] Yasuko Matsubara, Yasushi Sakurai, B. Aditya Prakash, Lei Li, and Christos Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 6–14, New York, NY, USA, 2012. ACM.

[99] Miller McPherson, Lynn S. Lovin, and James M. Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[100] B. Mobasher et al. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Techn.*, 7(4), 2007.

[101] Alan L. Montgomery. Applying quantitative marketing techniques to the internet. *Interfaces*, 31(2):90–108, March 2001.

[102] R. Narayanam and Y. Narahari. A shapley value-based approach to discover influential nodes in social networks. *Automation Science and Engineering, IEEE Transactions on*, PP(99):1 –18, 2010.

[103] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsŮi. *Mathematical Programming*, 14(1):265–294, 1978.

[104] Rina Panigrahy and Sundar Vishwanathan. An $O(\log^* n)$ approximation algorithm for the asymmetric p-center problem. *J. Algorithms*, 27(2):259–268, 1998.

[105] Foster J. Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 445–453, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[106] Al Rashid, George Karypis, and John Riedl. Influence in ratings-based recommender systems: An algorithm-independent approach. In *SDM*, 2005.

[107] Lisa Rashotte. Social influence. In *The Blackwell encyclopedia of sociology*, volume IX, pages 4426–4429, 2006.

[108] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[109] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 61–70, New York, NY, USA, 2002. ACM.

[110] Daniel M. Romero, Wojciech Galuba, Sitaram Asur, and Bernardo A. Huberman. Influence and passivity in social media. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 113–114, New York, NY, USA, 2011. ACM.

[111] Eldar Sadikov, Montserrat Medina, Jure Leskovec, and Hector Garcia-Molina. Correcting for missing data in information cascades. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 55–64, New York, NY, USA, 2011. ACM.

[112] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Learning continuous-time information diffusion model for social behavioral data analysis. In Zhi-Hua Zhou and Takashi Washio, editors, *Advances in Machine Learning*, volume 5828 of *Lecture Notes in Computer Science*, pages 322–337. Springer Berlin / Heidelberg, 2009.

[113] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Selecting information diffusion models over social networks for behavioral analysis. In *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part III*, ECML PKDD'10, pages 180–195, Berlin, Heidelberg, 2010. Springer-Verlag.

[114] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *KES '08: Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part III*, pages 67–75, Berlin, Heidelberg, 2008. Springer-Verlag.

[115] Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. Learning diffusion probability based on node attributes in social networks. In *Proceedings of the 19th international conference on Foundations of intelligent systems*, ISMIS'11, pages 153–162, Berlin, Heidelberg, 2011. Springer-Verlag.

[116] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 851–860, New York, NY, USA, 2010. ACM.

[117] Juan J. Samper, Pedro A. Castillo, Lourdes Araujo, and Juan Julián Merelo Guervós. Nectarss, an rss feed ranking system that implicitly learns user preferences. *CoRR*, abs/cs/0610019, 2006.

[118] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[119] Rossano Schifanella, Alain Barrat, Ciro Cattuto, Benjamin Markines, and Filippo Menczer. Folks in folksonomies: social link prediction from shared metadata. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 271–280, New York, NY, USA, 2010. ACM.

[120] Shang Shang, Pan Hui, Sanjeev R. Kulkarni, and Paul W. Cuff. Wisdom of the crowd: Incorporating social influence in recommendation models. In *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ICPADS '11, pages 835–840, Washington, DC, USA, 2011. IEEE Computer Society.

[121] 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*, WSDM '12, pages 733–742, New York, NY, USA, 2012. ACM.

[122] Parag Singla and Matthew Richardson. Yes, there is a correlation: - from social networks to personal behavior on the web. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 655–664, New York, NY, USA, 2008. ACM.

[123] Petr Slavík. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters*, 64(5):251–254, 1997.

[124] Xiaodan Song, Yun Chi, Koji Hino, and Belle L. Tseng. Information flow modeling based on diffusion rate for prediction and ranking. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 191–200, New York, NY, USA, 2007. ACM.

[125] Xiaodan Song, Belle L. Tseng, Ching-Yung Lin, and Ming-Ting Sun. Personalized recommendation driven by information flow. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 509–516, New York, NY, USA, 2006. ACM.

[126] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41 – 43, 2004.

[127] Mohsen Taherian, Morteza Amini, and Rasool Jalili. Trust inference in web-based social networks using resistive networks. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, ICIW '08, pages 233–238, Washington, DC, USA, 2008. IEEE Computer Society.

[128] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 267–274, New York, NY, USA, 2008. ACM.

[129] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 807–816, New York, NY, USA, 2009. ACM.

[130] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 717–726, New York, NY, USA, 2007. ACM.

[131] Oren Tsur and Ari Rappoport. What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 643–652, New York, NY, USA, 2012. ACM.

[132] T. Valente. *Network Models of the Diffusion of Innovations*. Hampton Press, 1955.

[133] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing, vol. 8, no. 3, pp. 410-421*, 1979.

[134] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 501–508, New York, NY, USA, 2006. ACM.

[135] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 1039–1048, New York, NY, USA, 2010. ACM.

[136] D.J. Watts. Challenging the influentials hypothesis. *WOMMA Measuring Word of Mouth, Volume 3*, pages 201–211, 2007.

[137] D.J. Watts. Viral marketing for the real world. *Harvard Business Review*, pages 22–23, May 2007.

[138] D.J. Watts and P.S. Dodds. Influential, networks, and public opinion formation. *Journal of Consumer Research*, 34(4):441–458, 2007.

[139] 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*, WSDM '10, pages 261–270, New York, NY, USA, 2010. ACM.

[140] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering proble. *Combinatorica*, 2(4):385–393, 1982.

[141] Jiyoung Woo, Jaebong Son, and Hsinchun Chen. An SIR model for violent topic diffusion in social media. In *Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on*, pages 15 –19, july 2011.

[142] Jaewon Yang and Jure Leskovec. Modeling information diffusion in implicit networks. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 599–608, Washington, DC, USA, 2010. IEEE Computer Society.

[143] Cai-Nicolas Ziegler and Georg Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.