Efficient and Effective Identification of Influential Vertices in Social Networks (ソーシャルネットワーク上の高影響力頂点集合を特定する効率的かつ効果的なアルゴリズム)

by

Naoto Ohsaka
大坂 直人

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on December 8, 2017
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology
in Computer Science

Thesis Supervisor: Hiroshi Imai　今井 浩
Professor of Computer Science

ABSTRACT

Social influence and information sharing occur in daily life, and social networks have been a place where such social interactions diffuse. The recent advancement of social networking services has significantly boosted the scale and speed of influence and information diffusion and enabled us to exploit influence diffusion for business use such as viral marketing. Moreover, we have been able to access a vast amount of trace of user actions at an individual level, which has encouraged a deep understanding of the mechanism of social influence at scale. *Computational social influence* is one of the research fields utilizing such data, aiming at analyzing, understanding, and optimizing social influence through modeling of the diffusion process, learning of model parameters, and optimization of the obtained networks. One of the most fundamental problems involving social influence optimization is *influence maximization*, which was formulated by Kempe, Kleinberg, and Tardos in 2003. Influence maximization is a graph optimization problem of finding a set of vertices that maximize the expected number of influenced vertices, i.e., the size of influence diffusion. Due to approximation algorithms with a theoretical guarantee and the potential application to marketing strategies and information dissemination, it has been actively studied in graph mining and graph database community for the last ten-odd years.

However, from an algorithmic point of view, the following challenges have remained unresolved. Firstly, influence maximization is still difficult to solve on real-world social networks even though there have been developed nearly-linear time approximation algorithms. This is due to the massive scale and dynamic nature of networks of the day and *insufficient evaluation of algorithmic efficiency*. Notably, the benchmarking study on existing influence maximization algorithms published by Arora, Galhotra, and Ranu in May 2017 has demonstrated that the setting of model parameters assigned to each edge has a significant impact on algorithmic efficiency, and there is no single state-of-the-art with the best trade-off between computation time and solution quality. Hence, boosting algorithmic efficiency even in an experimental sense is an urgent task. Secondly, influence maximization may result in ineffective strategies for influence diffusion. Since network diffusion is a probabilistic process, influence maximization has adopted expectation as a statistic to be optimized due to its simplicity and tractability. However, influence diffusion may end with a much smaller number of influenced vertices than the expectation. Expectation itself is not able to capture such a *risk*. Thus, it is unclear whether expectation maximization is able to produce low-risk strategies.

In this thesis, we address the above two challenges. In the first aspect, we explore efficient computation of influence maximization in practice. Our common tool for this purpose is the *empirical observations of the diffusion process*. There are two factors that may affect influence diffusion, i.e., the network structure and the setting of edge parameters. We conduct comprehensive experimental analysis using eighteen real-world networks and seven settings of edge parameters. We then discover the configurations of network and edge parameter setting for which existing algorithms become inefficient. We also find that existing algorithms incur redundant computation for such configurations. Based on the empirical analysis, we devise efficient algorithms under three situations below. First, we propose a *fast algorithm for influence maximization*. Our empirical observation tells us that for real-world networks, the difficult subproblem of influence maximization can be solved more quickly by using a simple linear time preprocessing technique. We experimentally compare the proposed algorithm with a number of existing algorithms. We show that heuristic algorithms often provide 10% less influential solutions while running faster than the proposed algorithm, and existing algorithms that have a theoretical guarantee of the solution quality demonstrate high-quality solutions; however, they cannot handle ten-million-edge networks for a certain setting of edge parameters. For such parameter settings, the proposed algorithm works and it provides comparable solutions to the existing algorithms. In particular, the proposed algorithm runs within two hours for a large network with hundreds of millions of edges. Further, we confirm the computation time reduction due to the proposed techniques by several orders of magnitude. Next, we develop a *dynamic indexing algorithm for real-time influence maximization* on evolving networks. We design a dynamic index structure, query algorithms for influence maximization, index update algorithms for graph changes. Then, we

propose techniques for improving the algorithmic efficiency of naive update algorithms based on our empirical observation. We experimentally verify that our algorithm can update an index within one second on networks with tens of millions of edges for almost all configurations, which is several orders of magnitude smaller than that required to reconstruct an index from scratch. Then, we present a *reduction algorithm for massive networks*. In order to process billion-edge-scale networks, we consider reducing the size of an input at the expensive of solution quality. We propose a strategy for effectively identifying subgraphs that cause redundant computation and algorithms that produce a smaller graph that approximates an input graph. Throughout experimental evaluations using real-world networks with up to billions of edges, we confirm that an input graph is reduced to up to 4% and running influence maximization on the obtained graph achieves a few times speed-up without significant loss of solution quality.

In the second aspect, we address the risk of having a few influenced individuals. To this end, we employ *portfolio optimization* approach, which is a standard approach for risk management. Conceptually, in our context of influence diffusion, we virtually invest in the possible sets of vertices. Then, we adopt conditional value at risk as a statistic to be optimized instead of expectation. Conditional value at risk is one of the most popular risk measures in financial economics and actuarial science. Since we cannot use a standard approach for portfolio optimization because of exponentially many variables, we develop a new polynomial-time approximation algorithm. Our algorithm constructs a portfolio that approximates the maximum conditional value at risk within a constant additive error. Using relatively small network dataset, we experimentally demonstrate that the portfolios that our algorithm constructs achieve two times larger conditional value at risk than standard influence maximization, and the distribution of the number of influenced vertices is well concentrated on the expectation, which is desirable in terms of risk aversion.

論文要旨

日常生活で見られる社会的影響や情報共有はソーシャルネットワークを通じて拡散し、普及していく。ソーシャル・ネットワーキング・サービスの台頭により、その拡散の速度と規模は急上昇し、マーケティング戦略や情報流布といった応用の可能性が広がった。さらに、これらのサービスを通じて個人ユーザの膨大な行動履歴が入手可能となったことで、計算機科学の手法を利用して社会的影響や情報拡散の機構を解析・理解する需要が高まっている。このような解析を行う一般的な枠組みは、ソーシャルネットワークをグラフにより表現し、拡散過程のモデル化、モデルパラメータの学習、そして、得られたグラフの最適化という三段階を踏む。特に、社会的影響の最適化に関する最も基本的な問題は、**影響最大化**と呼ばれる Kempe、Kleinberg、Tardos により 2003 年に提案されたグラフ上の最適化問題である。影響最大化の目標は、影響を伝える頂点数の期待値が最大となるような頂点集合を選択することである。この問題に対する近似アルゴリズムの存在やバイラルマーケティングへの潜在的な応用可能性により、十数年に渡りグラフマイニングやグラフデータベースの領域で活発に研究が行われてきた。

しかしながら、この問題にはアルゴリズム的観点において挑戦的課題がある。まず、ほぼ線形時間の近似アルゴリズムが提案されているにもかかわらず、現実のソーシャルネットワークで影響最大化を効率的に解くことは依然として難しい点である。これは、ネットワークの規模やその成長速度だけではなく、これまでの研究における**不十分な性能評価**に起因する。2017 年 5 月には、Arora、Galhotra、Ranu が影響最大化の既存アルゴリズムを様々なネットワークデータと辺に割り当てられたモデルパラメータで実験するベンチマーキング論文を発表した。この研究により、モデルパラメータはアルゴリズムの効率に多大な影響をもたらすこと、及び、計算時間と解の質の最良のトレードオフを達成する単一の手法は無い、という示唆が得られた。したがって、計算効率を実験的な点で改善及び実証することが重要な課題となる。次に、影響最大化そのものが非効果的な戦略を生み出す可能性がある点が挙げられる。確率的な現象である拡散は実行のたびに異なる結果をもたらすため、何らかの指標を導入し最適化する必要がある。影響最大化は期待値をその扱いやすさから採用している。しかしながら、実際には期待値に比べて非常に小さい拡散が発生してしまう可能性がある。期待値はそのような**リスク**を捉えることはできないため、期待値最大化がリスク回避に効果的な戦略に結びつくとは限らない。

本論文では、これらの課題の解決に取り組む。一つ目の観点では、影響最大化の効率的計算を目指す。共通する道具は**拡散過程の実験的観察**である。拡散過程にはネットワークの構造と辺のパラメータからなる二つの因子が大きな変化を生じうる。そこで、十八種の現実のネットワークデータ及び七種の辺確率設定を用いた実験的解析を行い、既存アルゴリズムの性能劣化が起こってしまう設定を重点的に調べる。さらに、そのような設定においては既存アルゴリズムが冗長な計算をしうる点を説明する。この観察に基づき、以下の三種の状況設定における効率的アルゴリズムの開発及びその実験的実証を行った。まず、**影響最大化の効率的アルゴリズム**を提案する。提案手法は影響最大化アルゴリズムの一つのカテゴリに属するが、そこでボトルネックとなる部分問題に対する線形時間前処理による高速化技法を提案する。実ソーシャルネットワークを用いた実験評価によって、ヒューリスティックに基づく手法は提案手法よりも速い一方、しばしば影響力の低い解を出力する点、理論的保証のある手法の多くは拡散の大きい設定において性能劣化を起こす点、ま

た、提案手法は数億辺を有するグラフを処理し、ほとんど多くの設定において一貫して影響力の高い解を出力することを示す。特に、既存手法が性能劣化を起こす設定において提案手法は解の質と計算時間の最良トレードオフを達成する。次に、**成長するグラフにおける実時間解析のための動的索引手法**を提案する。まず、索引構造、影響最大化のクエリ処理手法、グラフ変化に対する索引更新手法の設計を行う。ここで、拡散過程の実験的解析に基づいた索引更新手法の高速化技法を提案する。数千万辺を有するグラフを用いた計算機実験により、多数の設定で索引を一秒以内に更新できることを確認する。これは索引の再構築に要する時間の数千分の一である。そして、**巨大なネットワークの縮小アルゴリズム**を提案する。数十億辺規模のネットワークを処理するために、影響最大化の質の劣化をゆるし、事前にグラフを縮小することを考える。実験的観察に基づき、冗長な計算の原因となる部分を特定する戦略及びそれに基づき入力グラフを縮小するアルゴリズムを与える。数十億辺を有する現実のネットワークを用いた実験によって、提案手法がグラフの大きさを最大で 4%に縮小する点、及び、影響最大化を縮小されたグラフで行うことで、入力グラフと遜色ない質の解を数分の一の時間で得られることを確認する。

第二の観点では、影響が広範囲に拡散せずに終了してしまうリスクの回避を取り扱う。まず、リスクマネジメントにおいて標準的なアプローチである**ポートフォリオ最適化**を導入し、複数の頂点集合に仮想的な投資を行うことを考える。そして、期待値の代わりに *Conditional Value at Risk* という金融経済や保険数理の分野のリスク指標を導入する。定式化した問題は指数個の変数を含み、ポートフォリオ最適化の標準的手法は適用できないため、新たな多項式時間近似アルゴリズムを提案する。比較的小さなグラフデータを用いた実験により、提案手法は影響最大化やヒューリスティックと比較して最大で二倍の Conditional Value at Risk を持つポートフォリオを構築した。さらに、影響の大きさの分布は期待値により集中しておりリスク回避の点で望ましい点を示す。

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisor, Professor Hiroshi Imai for his precious advice and comments. His guidance helped me in all the time of conducting research, presenting research talks, and writing of this thesis. Without his guidance and persistent help, I would not have been able to accomplish my five-year research in the graduate school.

I am also thankful to my thesis committee chair, Prof. Tetsuo Shibuya, and the committee members, Prof. Naoki Kobayashi, Prof. Mary Inaba, Prof. Rui Yamaguchi, and Prof. Junya Honda, for the time and effort taken to review this thesis and for the valuable feedback and suggestions.

I am deeply grateful to all my research collaborators, Takuya Akiba, Sumio Fujita, Takuro Fukunaga, Kazuhiro Inaba, Yoichi Iwata, Naonori Kakimura, Ken-ichi Kawarabayashi, Takanori Maehara, Tomoaki Ogasawara, Tomohiro Sonobe, Yutaro Yamaguchi, and Yuichi Yoshida. I have been extremely fortunate to have worked with these talented researchers. I want to offer my special thanks to Takuya Akiba for guiding me how to proceed experimental evaluations effectively and giving a lot of knowledge, and Yuichi Yoshida for countless discussions with him.

I also would like to thank all members in Imai Laboratory. I received insightful comments and feedback from these members through internal seminars. I would like to list all members, Akitoshi Kawamura, François Le Gall, Vorapong Suppakitpaisarn, Masato Edahiro, Kenta Takahashi, Takahiko Satoh, Jean-Francois Baffier, Toshihiro Tanuma, Yoichi Iwata, Bingkai Lin, Akiba Takuya, Hidefumi Hiraishi, Alonso Gragera, Keigo Oka, Akira Motoyama, Yuto Hirakuri, Prompong Pakawanwong, Yosuke Yano, Yuki Kawata, Chihiro Komaki, Takeo Asai, Takuto Ikuta, Takanori Hayashi, Fumiya Satoh, Makoto Soejima, Shuichi Hirahara, Kentaro Yamamoto, Tomohiro Katayama, Shogo Nakajima, Kohei Uezato, Satoru Yasuda, Seiya Takahara, Ly Nguyen, Anthony D'Amato, Kensuke Imanishi, Wataru Inariba, Tomoaki Ogasawara, Takashi Shimada, Shinya Shiroshita, Hyonsoku Chang, Sakuya Hashimoto, Shogo Murai, Shuma Okamura, Kanto Teranishi, Kohji Liu, Naosuke Shindo, Takuto Shigemura, Tsuyoki Kumazaki, Emanuel Gedin, Marius Kaufmann, Ruben Wohlgenannt, Clara Tersen, Az-elrabe Bitane, Amaury Josse, Rohit Kumar Singh, Vu Phan Thanh, Arasu Arun, Shaswat Chaubey, Kittiphon Phalakarn, Kittiphop Phalakarn, Tanguy Pomas, Tatiana Neuer, Simon Klein, Florian Steinberg, Jeremy Cohen, and Holger Thies.

In addition to the laboratory, I have been a member of complex network and map graph group at JST ERATO Kawarabayashi Large Graph Project. Throughout weekly group seminar and annual events, I have broadened my knowledge and experience. I would like to list all the group members, Ken-ichi Kawarabayashi, Yuichi Yoshida, Masato S. Abe, Takuya Sekiguchi, Wataru Inariba, Ryosuke Nishi, Yutaka Horita, Taro Takaguchi, Takahiro Ezaki, Junichi Teruyama, Naoki Masuda, Takehisa Hasegawa, Kazuhiro Inaba, Mitsuru Kusumoto, Takuya Ak-

iba, Yoichi Iwata, Tatsuro Kawamoto, Yuki Kawata, Yukie Sano, Leo Speidel, Yosuke Yano, Takanori Hayashi, Mao Fukadai, Ryohei Hisano, Keigo Oka, Kodai Saito, Daiki Takeuchi, Yoshitaka Murai, and Takuto Ikuta. Additionally, I have had a lot of opportunities to collaborate with other group members, specifically, Takuro Fukunaga, Naonori Kakimura, Takanori Maehara, Tomohiro Sonobe, and Yutaro Yamaguchi.

I also have been involved in Yahoo! JAPAN Research for four years. Especially, I would like to thank Tajima Akira and Sumio Fujita for permission to access a large volume of valuable data and their technical assistance.

Lastly, I would like to express my profound gratitude to my family for their emotional and financial support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis deals with graph-algorithmic problems related to computational social influence. In this chapter, we will describe the role of social networks in influence and information diffusion, review the research on computational social influence, and provide an overview of the contribution of the thesis.

1.1 Influence and Information Diffusion

When people make a decision, they are frequently influenced by others. For example, when a new product is just released, you may consult with your relatives, friends, or colleagues in order to decide whether to purchase it or not. The change in people's decisions, opinions, emotions, or behaviors caused by others is called *social influence*. Social influence happens in daily life and takes place in a *social network*, which is defined as a graph made up of vertices corresponding to persons and edges describing relationships between them. We are connected to a surprising number of persons (indirectly) on social networks [88, 185]. Hence, social influence *diffuses* over social networks and would form an explosive cascade. Moreover, social networks also play a substantial role in information sharing and acquisition.

1.1.1 Social Networking Services and Microbloggings

While social influence and information sharing have suffered from temporal or spatial restrictions for a long period, the Internet has broken them. We can communicate with persons throughout the world instantaneously via various online services. Moreover, social networking services and microbloggings have been rapidly popularized since the 2000s. A *social networking service (SNS)* is an online site that promotes closed person-to-person communication activities among its users who share similar interests, careers, or residences. A *microblogging* is a website where its users can post a short text concerning their thoughts or notes and exchange it with other users. These services include social media Facebook, Google+, and Tumblr, photo-sharing websites Flickr, Instagram, and Pinterest, a business-oriented SNS LinkedIn, and a microblogging Twitter. They provide a "platform" allowing people to produce, share, and distribute contents and communicate with each other, and form massive-scale social networks: the Twitter social network contains 288 million users and 60 billions of following links [6] and the Facebook social network contains 1.4 billion users and 400 billion edges [48]. Further, a countless number of productions are continually published on these services: 300 hours of video are uploaded to YouTube every minute [4], 6,000 tweets are tweeted on Twitter every second [5], 52 million photos are uploaded to Instagram each day and 1.65 billion favorite markings are generated [2]. Content sharing or communication among the users causes social influence. Its cascade may grow *at scale* beyond the publisher's intention and become a social phenomenon.

1.1.2 Viral Marketing

Influence diffusion on the Internet has also a potential impact for business use. *Viral marketing* is a marketing strategy that intentionally triggers a rapid diffusion of message sharing, with the aim of product promoting, customer acquisition, or brand awareness increasing. The term "viral" is an analogy to the spread of virus over a population. The message to be diffused takes the form of texts, images, videos, or web pages, and it is called "going viral" when being shared with a number of individuals.

Hotmail, a web-based e-mail service, is one of the most successful examples of viral marketing. For every e-mail sent by its users, the tagline *"Get your free email at Hotmail"* is inserted at the bottom. By clicking this tagline, a receiver is guided to Hotmail's web page. This approach was really effective; Hotmail has acquired twenty millions of new sign-ups in a mere eighteen months [175].

1.2 Computational Social Influence

While a theory to explain the speed and scale of diffusion of a specific entity over a population has been initiated by sociologists [53, 166] by the name of "Diffusion of Innovations" [165] in the middle of the 20th century, the recent advancement of online social networking services has enabled us to collect a vast amount of traces of user actions at an individual level. Moreover, the service administrators have the whole platform under control, which encourages a deep understanding of the mechanism of social influence at scale [10, 17, 35] and an enhancement of influence diffusion, which has a potential in business use such as viral marketing and information dissemination.

Computer social influence [39] is an emerging field that aims at analyzing, understanding, and optimizing social influence with computational methods. To this end, we use graphs to describe social networks. However, unlike usual social network analysis, it is essential to capture the probabilistic nature of social influence; we consider graphs where parameters are assigned to every edge. These edge parameters represent the strength of influence between a pair of vertices. We refer those graphs as *influence graphs* throughout this thesis.

Figure 1.1 illustrates a flow of computational social influence. It is entailed three steps: (1) modeling of the diffusion process, (2) learning of the model parameters, and (3) optimization of the obtained influence graph. We finally make use of the analysis result in real-world applications. We describe each step in the following subsections.

1.2.1 Modeling

First of all, we need a model for network diffusion. Diffusion models state the (random) process by which social influence diffuses over a network. Classical models such as the susceptible-infected-recovered (SIR) model [101] in epidemiology and the Bass model [22] in marketing research have a few parameters that are able to control the global behavior only. In order to capture social interactions at the individual level, researchers have proposed a plethora of diffusion models.

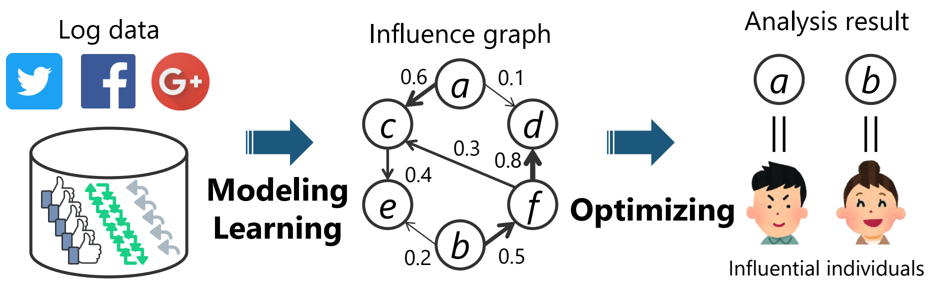


Figure 1.1: Flow of computational social influence.

The most fundamental and extended models are the independent cascade model and the linear threshold model.

Independent cascade model.

The *independent cascade (IC)* model was proposed by Goldenberg, Libai, and Muller [70, 71] in marketing research and has been popularized in computational social influence. In the IC model, each vertex takes the state of either *active* or *inactive*, and each edge has a parameter called *influence probability*. The diffusion process begins with a set of initially activated vertices called *seeds*. Then, an active vertex is given a single chance to activate each of its inactive neighbors. It succeeds with a probability associated with the corresponding edge. In this sense, the IC model is “sender-centric”.

Linear threshold model.

The *linear threshold (LT)* model was proposed by Kempe, Kleinberg, and Tardos [98] as a variation of the models of Granovetter [83] and Schelling [171]. The LT model has vertex-specific thresholds, each of which defines how hard it is to activate the vertex and is uniformly distributed between zero and one. Each edge has a weight representing the strength of influence. The diffusion process begins with a set of seeds. Then, an inactive vertex becomes active whenever the total weight of already-activated neighbors goes beyond its threshold. In this sense, the LT model is “receiver-centric”.

Extensions.

To reproduce the actual cascades observed in the real world more accurately, complex factors have been incorporated into the models. Those include a factor of topic [21], time-delay [76, 168] time-dependent process [85], vertex attributes [169], and locations [54].

1.2.2 Learning

Having defined diffusion models, we are required to learn those parameters. We usually assume that the underlying graph is known and the log data of user actions is available since these can be extracted from the platform. Existing learning methods use ad-hoc techniques [79, 113] or machine learning techniques [56, 68, 167]. It may be also the case that diffusion takes implicit networks and we have to infer the unobserved and unknown network structure [74, 75]. Besides, it is strongly demanded to determine how many training-samples are sufficient to learn model parameters. In learning theory, this question has been studied with the notion of sample complexity and learnability [7, 60, 60, 77, 143, 145].

1.2.3 Optimizing

Now, using influence graphs, we optimize social influence. We can come up with diverse objectives and situations. For example, one wants to make the diffusion of influence as large as possible by seeding influential individuals [98] for an effective viral marketing. Conversely, it might be demanded to confine already-propagated undesirable "misinformation" by broadcasting the correct information from appropriately selected individuals [33, 107]. Another one may manipulate the network topology to control the global behaviors [36, 102, 179], or monitor several vertices so as to detect the information diffusion as quickly as possible [118].

All the problems described above can be formulated as optimization problems on influence graphs. It should be noted that the probabilistic nature of network diffusion yields exponentially many possible observations. Hence, we choose a statistic to summarize them and then optimize it. *Expectation* is typically adopted due to its simplicity and tractability.

1.3 Influence Maximization Problem

Our interest is in graph-algorithmic optimization problems, the third step in computational social influence. In this thesis, we intensively study one of the most fundamental problems involving influence optimization called *influence maximization*. This section provides an overview and previous studies of influence maximization.

1.3.1 Informal Definition and Computational Properties

The motivation of influence maximization is the question in viral marketing that "what are the most effective individuals in the network to use for product promotion?" The seminal work of Domingos and Richardson [58, 162] in the early 2000s modeled a viral marketing using a Markov random field and considered to find a subset of customers that leads to the maximum increase in profits. In 2003, Kempe, Kleinberg, and Tardos [98] formulated influence maximization as a discrete optimization problem on influence graphs inspired by the study of Domingos and Richardson. Informally speaking, the influence maximization problem is defined as follows.

> Given an influence graph, a budget k, and a diffusion model, find k seeds such that activating them results in the maximum expected number of active vertices under the diffusion model.

In other words, the objective function of influence maximization so-called the *influence spread* is the expected number of vertices influenced by a set of seeds.

The computational complexity concerning influence maximization has been thoroughly investigated [37, 38, 98, 99, 103, 142]. Firstly, it is NP-hard to exactly solve influence maximization even in the special case of IC and LT [98]. Indeed, exactly computing the influence spread is #P-hard [41, 42]. Meanwhile, we are able to obtain approximate solutions as described below. Kempe, Kleinberg, and Tardos [98] have proven that the influence spread exhibits monotonicity and submodularity. *Monotonicity* is a property of set functions that captures an intuition that adding new elements into the input set does not decrease the function value. *Submodularity* is a property that roughly speaks that the change in the function value that the addition of a single element makes decreases as the size of an input set increases. For monotone submodular function maximization, the gold-standard *greedy algorithm* returns a $(1 - \mathrm{e}^{-1})$-approximate (k-sized) solution against the optimal solution, which was proven by Nemhauser, Wolsey, and Fisher [144] in 1978. More precisely, the greedy algorithm selects a new vertex that makes the largest increase in the influence spread and adds it into the seed set until k vertices have been inserted. Hence, the greedy algorithm requires a value oracle for the influence spread and calls it at most nk times, where n is the size of the ground set. Despite the #P-hardness of influence spread computation, fortunately, it is possible to obtain an estimate of the influence with an arbitrary accuracy by running Monte-Carlo simulations. Consequently, we are able to compute $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solutions in *polynomial time*.

The problem formulation of influence maximization using influence graphs has motivated a diversified range of subsequent studies as reviewed below.

1.3.2 Previous Studies

Development of efficient algorithms.

Because of the computational issue of evaluating the objective function, there has been remained a large room for efficient algorithm development. We describe three categories of existing influence maximization algorithms, which will be further reviewed Chapter 3.

The first category is **simulation-based** methods, which estimate the influence spread by repeatedly running Monte-Carlo simulations. They can be further classified into two types. *Naive estimation algorithms* [80, 118, 150, 151, 191, 192] conduct a number of Monte-Carlo simulations naively when required to estimate the influence spread during the greedy seed selection. This is computationally prohibitive since we need influence estimates for nk seed sets.

Snapshot-based algorithms [40, 46, 52, 106, 128, 131] is an application of sample average approximation. To be precise, snapshot-based algorithms sample a kind of random graphs from an input influence graph, each of which corresponds to an outcome of the random activation process and reuse them over greedy seed selection. Note that the empirical influence spread defined over the random graphs is the average number of reachable vertices on the random graphs. Therefore, we aim at selecting a vertex set that maximizes the total number of reachable vertices.

While snapshot-based algorithms yield well-influential seed sets and run faster than naive estimation algorithms, running them on large graphs is still costly. Even in the case of $k = 1$, we need to "compute the number of vertices that are reachable from each vertex" for all random graphs. This problem is known as *descendant counting* [27, 49]. We are able to solve descendant counting by running a graph traversal algorithm, e.g., a breadth first search (BFS) in linear time, for each vertex, which consumes quadratic time in the graph size. Unfortunately, this simple algorithm is almost best possible; the descendant counting problem is not solvable in less than quadratic time under a computational complexity assumption [28].

The second category is algorithms based **reverse influence sampling**. Reverse influence sampling (RIS) is the first nearly-linear time randomized algorithm (for constant k) pioneered by Borgs, Brautbar, Chayes, and Lucier [29] in 2014. This was a theoretical and experimental breakthrough in influence maximization algorithms. RIS-based algorithms generate *reverse reachable (RR) sets*, which is a set of vertices that would have influenced a uniformly-chosen target vertex. Then, for a vertex set, the fraction of RR sets intersecting it (multiplied by the graph size) is an unbiased estimator of its influence spread.

The central problem concerning RIS-based algorithms is to decide the time at which we stop RR set generation. Since the stopping condition of Borgs, Brautbar, Chayes, and Lucier [29] demands a hidden constant being quite large, there have been established follow-up work [30, 57, 92, 147, 148, 149, 152, 176, 177]. Until recently, RIS-based algorithms have been regarded as state-of-the-art [148, 177].

The last category is **heuristic** algorithms [40, 41, 42, 45, 47, 67, 81, 94, 96, 104, 105, 122, 138, 153, 161, 182, 184, 189, 190]. Heuristic algorithms perform neither direct Monte-Carlo simulations nor RR set generations. Basically, they assume ad-hoc conditions so as to make it easy to compute the influence spread. For example, one assumes influence flowing along with shortest paths [105] and another assumes influence diffusing only local neighbors [40]. As expected, the resulted seed sets are less influential.

More sophisticated formulations.

As the original problem formulation was simple, it is natural to think about making it more suitable for practical situations. One may consider maximizing the probability that a particular vertex is influenced [86], minimizing the time required to influence a certain fraction of the entire network [82], or selecting a set of seeds, each of which incurs a different cost [154].

Further, it is also easy to imagine to incorporate factors that affect the diffusion process. Those factors cover time-delay [43, 59, 73, 127] geographic location [121, 183, 194], novelty decay [64], product value [129] and product design [20], intermediate levels of influence [55], or new diffusion processes, e.g., voter model [61, 124, 193] and heat diffusion [133]. Note that the ultimate goal of these studies is the use for viral marketing.

The situational limitation of influence maximization is that we must complete the seed set selection *before* implementing viral campaigns. We never know the result unless we try, and this hinders us from reflecting the feedback of the activation process to improve the current strategy. In machine learning community, there have been emerged new frameworks that reflect the feedback for more effective viral marketing. One adopts an *online setting* [115], where we execute viral campaigns and refine model parameters alternately. In each step, we pick up a small seed set, which may not be influential, and run a viral campaign starting from the set. Then, we observe the feedback of this small campaign to improve the knowledge of model parameters. The other adopts an *adaptive setting* [72], where we adaptively select the next seed every time observing the result of the activation process made so far. Importantly, those frameworks require to repeatedly select an influential seed set. Thus, influence maximization is used as a "subroutine" inside them rather than as a standalone.

Field experiments and other possible applications.

Lastly, we describe the applicability of influence maximization. The original motivation for this problem comes from the viral marketing application. However, to the best of our knowledge, there have been no concrete cases of such applications. This is because of ideal assumptions of influence maximization described so far; there has been a discrepancy regarding the intricacy of diffusion models between the predicting side and the algorithmic side.

Nonetheless, due to its simplicity and generality, influence maximization can be used for (slightly) different purposes. For example, a project at the University of Southern California [3] aims at increasing awareness about a dangerous disease among a homeless population. To this end, it has posed to exploit network diffusion effect and has developed a decision support system for finding the most influential individuals [187, 188]. In addition, we can run influence maximization on other types of network. In computational biology, a genetic regulatory network is described as a graph where vertices represent genes, proteins, or messenger ribonucleic acids, and edges represent interactions between them. On such networks, identifying a group of vertices that plays a role in information dissemination is a fundamental task. Recent studies have made attempts to apply influence maximization for such purpose [69, 95].

1.4 Challenges

Having reviewed the studies of influence maximization, we would like to point out two challenges from an algorithmic aspect. In particular, we will explore how to "efficiently" and "effectively" identify influential seed vertices from a social network.

1.4.1 Efficient Computation

Despite a considerable effort for the development of efficient algorithms for the last ten-odd years, influence maximization is still difficult to solve on real-world networks. This is due to the massive scale and dynamic nature of networks of the day and *insufficient evaluation of algorithmic efficiency*. While the algorithmic study has received much attention from researchers, the influence graph dataset has been almost publicly unavailable. Consequently, we have no choice but to assign influence probabilities for each edge *artificially*. This caused a *bias* in parameter settings for performance evaluation. Notably, Arora, Galhotra, and Ranu [12] have revealed this bias by their paper presented at the 43rd ACM SIGMOD International Conference on Management of Data in May 2017. This paper has conducted an exhaustive benchmarking study of existing algorithms with a number of graph data and several model parameter settings, e.g., an IC with constant probabilities, an IC with degree-weighted probabilities, and an LT with degree-weighted weights. The result of [12] implicated that (1) the setting of influence probabilities has a significant impact on algorithmic efficiency, and (2) there is no single state-of-the-art that achieves the best trade-off between computation time and solution quality. Hence, boosting algorithmic efficiency even in an "experimental sense" is an urgent task.

1.4.2 Effective Strategies

What is an effective strategy for information dissemination? Since network diffusion is a probabilistic process, influence maximization has adopted expectation as an objective function due to its simplicity and tractability. However, this choice is rather ad-hoc; influence diffusion may end with a much smaller size than the expectation.

Suppose that we have two seed sets, where one influences either 10 or 90 individuals with equal probability and the other influences either 40 or 60 individuals with equal probability. Both influence 50 individuals on average. However, in some realistic scenario, a campaign planner may want to avoid having fewer influenced individuals than the average frequently, and the latter solution is thus preferable. Expectation itself cannot capture such a *risk*, and hence, expectation maximization is not necessarily able to produce low-risk strategies.

1.5 Contributions

In this thesis, we address the challenges involving efficient computation (Chapters 4–7) and effective strategies (Chapter 8) especially for against influence maximization under the independent cascade model. In what follows, we describe an overview of our contributions and the key ideas behind them.

1.5.1 Analysis of the Trends of Diffusive Behaviors (Chapter 4)

In the first aspect, we explore how to boost influence maximization in practice. Our common tool for this purpose is the empirical analysis of the trends if diffusive behaviors. There are two factors that may affect influence diffusion. One factor is the structure of networks. Real-world networks made up of relationships concerning social, communication, web, biology, and computers share essential structural features and are referred to as *complex networks*. They look like neither trees nor truly random; they have heavy-tailed degree distributions [19, 62] and include an amazing number of triangles [185], which helps us interpret the obtained outcomes. The other is the strategy of influence probability assignment. Intuitively, the higher influence probability is, the more frequently activation trials succeed.

The objective is to answer the difference in diffusive behaviors and algorithm efficiency among configurations of the two factors. For this purpose, we carry out comprehensive experimental investigation of two concepts related to influence maximization, using eighteen real-world networks and seven strategies for influence probability assignment. The strategies can be roughly categorized into two types below.

- **Unweighted settings.** Each influence probability is independently drawn from a certain distribution, e.g., uniform or exponential. Vertices with many neighbors have many chances to influence neighbors. Hence, what to expect is that influence triggered by higher-degree vertices diffuses more widely over the network.
- **Degree-weighted settings.** Each influence probability is determined based on vertex degrees, which equalizes vertices' impact on the neighbors. Hence, one can expect that any influence does not widely diffuse.

Concept 1: Reachable sets.

A reachable set of a vertex is a set of vertices that the vertex can reach. Computing the reachable sets for each vertex in random graphs is a key step in snapshot-based algorithms. Our main observation is that there are two extreme types of the size distribution of reachable sets, i.e., *bimodal* distributions and *decreasing* distributions. We have bimodal distributions only when we use unweighted settings and the *giant component (GC)* is present in random graphs. In such a case, the total size is quite large, and running snapshot-based algorithms naively is computationally prohibitive. However, we also find that it comprises redundant graph traversal, which is easily avoidable. When we have decreasing distributions if either the GC is not present or we use degree-weighted settings. In such a case, the total size is tiny.

Concept 2: Reverse reachable sets.

A reverse reachable (RR) set is a set of vertices that would have influenced a randomly-chosen vertex. Generating RR sets is a key step in RIS-based algorithms. As an RR set is identical to a reachable set (in a transposed influence graph), we observe the similar trends to reachable sets. If the size distribution of RR sets is bimodal, RIS-based algorithms may consume more computational effort. Meanwhile, RR sets under degree-weighted settings are consistently small. Moreover, we find that vertices in larger RR sets are connected to more edges than the whole network. To put it differently, large RR sets seldom change when the underlying graph has been updated.

In summary, our observations acquired so far prompt "simple but powerful" speeding-up techniques for managing static networks (Chapter 5) and dynamic networks (Chapter 6) and a "redundancy reduction" strategy (Chapter 7).

1.5.2 Fast Algorithm for Influence Maximization (Chapter 5)

First, we present an efficient algorithm for influence maximization. We propose a new snapshot-based algorithm *pruned Monte-Carlo (PMC)*. PMC comprises of two boosting techniques which do not affect influence estimation.

Technique 1: Pruned BFS.

The first technique is *pruned breadth first search (pruned BFS)* for fast descendant counting. Based on our empirical observation, we aim at cutting down the redundant graph traversal. We perform a linear-time preprocessing for each random graph to find the GC and compute related information. Then, before beginning a BFS from a vertex, we check whether it can reach the GC or not, If this is the case, then "knowing" that we will visit the GC, we prune the graph traversal from the GC. This simple pruning achieves incredible speeding-up in real-world social networks. For unweighted settings, pruned BFS reduces the number of vertices explored during solving descendant counting by several orders of magnitude.

Technique 2: BFS avoidance.

The second is *breadth first search avoidance (BFS avoidance).* We aim at correctly "guessing" the size of reachable sets of each vertex without any BFS. To this end, we propose to reuse the reachability information obtained so far. We perform a linear-time postprocessing at the end of each iteration to detect which vertices' reachability information have to be updated. Henceforth, we are able to suppress an increase in running time with increasing k. The reduction of the number of BFSes by this technique is up to 99.8%.

Evaluations.

We conduct intensive experimental evaluations using social networks with up to 200 million edges and seven strategies of influence probability assignment. We confirmed that PMC consistently produced high-quality solutions except for a part of degree-weighted settings. PMC finished in two hours for any configurations and demonstrated robust efficiency against influence probability settings and increase in budget k. Comparing with a collection of existing algorithms, we confirm the following remarks. Heuristic algorithms ran often faster than PMC, but their solutions were occasionally 10% less influential than those of PMC. Existing simulation-based algorithms did not finish or ran out-of-memory errors for networks with tens of millions of edges although providing comparable solutions to PMC. RIS-based algorithms demonstrated the best performance for degree-weighted settings in terms of both running time and solution quality. However, they extremely slowed down under unweighted settings even for medium-sized networks. Furthermore, when the GC is present in random graphs, some of the RIS-based algorithms resulted in out-of-memory errors. To sum up, for unweighted settings, PMC can be the best choice in terms of the trade-off between efficiency and quality.

Publication. This result was achieved in joint work with Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. An extended abstract was also published in the Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014; technical track paper) [156].

1.5.3 Dynamic Indexing Algorithm for Real-time Influence Analysis (Chapter 6)

Next, we consider influence maximization in dynamic networks. Real-world social networks exhibit a highly dynamic nature and evolve rapidly over time [117], and every information can quickly become outdated. Unfortunately, running static algorithms such as PMC from scratch every time the graph updates is computationally prohibitive because it requires at least linear time in the graph size. A hopeful approach is to design a *dynamic index* that is able to efficiently support both graph modifications and analysis queries.

In this chapter, we propose a dynamic indexing algorithm for real-time influence analysis in dynamic social networks. Our index is *fully dynamic*, i.e., it can instantly incorporate any kind of influence graph modification including additions and deletions of vertices and edges, and updates of influence probabilities. It enables us to quickly obtain solutions of influence maximization on the latest graph snapshot and keep track of the influence transition of vertices. The development of our dynamic algorithms involves complicated tasks as explained below.

Task 1: Memory-efficient index design.

Our index structure is an extension of RR sets of Borgs, Brautbar, Chayes, and Lucier [29]. Conceptually, once we can update RR sets according to the graph change as if reconstructed from scratch, we maintain and reuse them. However, the structure of the original RR set does not contain sufficient information to implement correct update algorithms. Hence, we examine the necessary information that should be stored and carefully design *sketches* that we store in an index. We refer our empirical observation to reduce space consumption.

Task 2: Fast update algorithms.

We then design index update algorithms for each kind of graph modification. Naive implementations for them traverse each sketch in an index and decide whether it will change or not. Whereas we are able to guarantee the non-degeneracy of the quality of solutions after an arbitrary number of updates, those implementations are not efficient enough to track highly-dynamic networks. This is serious in particular when adopting unweighted settings under which RR sets can become extremely large.

To overcome this, we consider speeding-up those naive implementations. Our empirical observation tells us that under unweighted settings, relatively large RR sets infrequently change after a small graph modification. This is because of many "detours" inside such RR sets. We leverage this fact to propose a *reachability-tree-based pruning technique*. We make our index having reachability trees in addition to sketches. By using them, we test whether each sketch will *not* change after the deletion of edge or vertex with small computational effort. Further, for sketches that passed the test, we perform a traversal on a limited area of the sketch, which is determined based on the reachability tree, instead of the whole area. It should be noted that our technique correctly updates the sketches.

Evaluations.

We conduct experiments using real-world social networks with tens of millions of edges where each edge has the timestamp at which it was created, under both unweighted settings and degree-weighted settings. As expected, naive update algorithms show poor efficiency under unweighted settings. Our speeding-up techniques drastically reduce the time required to update an index for a single deletion of vertex and edge by the order of thousands. Consequently, our algorithm can update an index within one second for almost all configurations, which is several orders of magnitude smaller than that required to reconstruct an index from scratch. Furthermore, using our index, we can obtain accurate estimates of the influence spread within a millisecond, and we select highly influential vertices approximately ten times faster than static algorithms are able to.

Publication. This result was achieved in joint work with Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. An extended abstract was also published in the Proceedings of the VLDB Endowment, Vol. 9, No. 12 (PVLDB 2016; research track full paper) [157].

1.5.4 Reduction Algorithms of Massive Influence Graphs (Chapter 7)

Lastly, we examine the reduction of massive-scale influence graphs. Since it is still expensive to analyze billions-edge-scale influence graphs directly, we aim at reducing the size of an input influence graph so that existing influence maximization algorithms can be used.

Coarsening procedure.

Let us start with a strategy for influence graph reduction. Our empirical observation tells us again that the existence of the GC (in random graphs) causes performance degradation of influence maximization algorithms. Conversely, shrinking down a partial influence graph which intersects random GCs frequently, we may be able to reduce the computational cost. This motivates us to devise a procedure called *coarsening*, which merges a vertex set into a single weighted vertex. Intuitively speaking, the coarsening procedure aims to collapse redundant pieces in an influence graph. In order to investigate what type of vertex set is desired to be coarsened, we theoretically analyze the impact of coarsening on the diffusion properties and the graph size. Based on our results, we introduce a novel notion of vertex set which is desired to be coarsened called r-robust strongly connected components. As r-robust strongly connected components are able to hold the GC, coarsening them effectively reduces the number of edges with a little deterioration of diffusive properties in practice.

Algorithm and frameworks.

We propose a new scalable reduction algorithm designed for influence graphs. Given an input influence graph, the proposed algorithm produces a compact vertex-weighted influence graph by coarsening it. We provide two implementations; a speed-oriented implementation which runs in linear time with linear space and a scalability-oriented implementation which runs in practically linear time with sublinear space. We further explain how to implement our algorithm on parallel and distributed systems. In addition, we present general frameworks that accelerate existing algorithms for influence estimation and influence maximization problems. Using these frameworks, we can quickly obtain solutions that have accuracy guarantees under a reasonable assumption.

Evaluations.

Our intensive experiments on real-world networks with up to billions of edges demonstrate the effectiveness, efficiency, and scalability of the proposed algorithm and frameworks. The remarkable reduction of the graph size is accomplished for unweighted probability settings. Specifically, we confirm that the proposed algorithms processes billion-edge graphs within hours and reduces the number of edges to up to 4%, the proposed influence estimation framework cuts down the computation time of a simulation-based method [98] to 3.5% and the proposed influence maximization framework achieves up to four times speed-up of an RIS-based algorithm D-SSA [148] without significant loss of solution quality.

Publication. This result was achieved in joint work with Tomohiro Sonobe, Sumio Fujita, and Ken-ichi, Kawarabayashi. An extended abstract was also published in the Proceedings of the 43rd ACM SIGMOD International Conference on Management of Data (SIGMOD 2017; research track full paper) [158].

1.5.5 Portfolio Optimization for Acquiring Low-risk Strategies (Chapter 8)

Finally, in the second aspect, we deal with a risk of having a few influenced individuals that expectation is not able to capture. This chapter comprises a new problem formulation and the corresponding algorithm.

Problem formulation.

We adopt *portfolio optimization* [136], which is a standard approach for risk management in the field of financial economics and mathematical finance. A *portfolio* is a collection of investments on financial assets such as stocks and bonds. In general, properly constructing portfolios tunes the trade-off between return (e.g., expectation) and risk (e.g., standard deviation). In our context of influence diffusion, we construct a portfolio by *virtually* investing on the possible sets of k vertices. In other words, virtual investment corresponds to a real number and assets correspond to k-vertex sets. Intuitively speaking, a portfolio is effective in risk aversion if its seed sets show weak (or negative if possible) correlations of the number of influenced vertices.

In the field of financial economics and actuarial science, various risk measures have been proposed to quantify risk. We adopt the *conditional value at risk (CVaR)* [163, 164], which is one of the most popular risk measures because of its good mathematical properties [8]. In our context, roughly speaking, CVaR measures the expected number of influenced vertices in the worst α-fraction of cases. α is typically chosen to be 0.01 or 0.05.

Proposed algorithm.

Now, we develop an algorithm for our CVaR maximization problem. Whereas our problem is a standard formulation of portfolio optimization, existing algorithms for portfolio optimization cannot be applied because it has exponentially many variables. To overcome this difficulty, we propose to use the multiplicative weights update algorithm [15]. An essential ingredient in the multiplicative weights algorithm is an oracle for a constraint that is a convex combination of the constraints in the original problem. Although we cannot precisely check whether the constraint is satisfied, we can approximately check it by running the greedy algorithm for submodular function maximization. Our proposed algorithm returns a portfolio on vertex sets that approximates the maximum CVaR within a constant additive error.

Evaluations.

We conduct experiments using small real-world social networks with up to 70 thousand edges. In short, the portfolios that our algorithm constructs achieve two times higher CVaRs with $\alpha = 0.01$ than the seed set obtained by a standard influence maximization algorithm in Chapter 5. Further, the distribution of the number of influenced vertices is well concentrated on the expectation, which is desirable in terms of risk aversion.

Publication. This result was achieved in joint work with Yuichi Yoshida. An extended abstract was also published in the Proceedings of the 26th International Conference on World Wide Web (WWW 2017; research track full paper) [155].

1.6 Organization of This Thesis

The structure of this thesis is depicted in Figure 1.2. In Chapter 2, we introduce basic notions used throughout this thesis and define diffusion models and the influence maximization problem. Chapter 3 is devoted to an extensive review of existing algorithms of influence maximization. In Chapter 4, we conduct an experimental analysis of the diffusion process of the independent cascade model. In the following three chapters, we explore efficient influence maximization by exploiting the empirical observations. In Chapter 5, we propose our fast algorithm for influence maximization and experimentally compare with a collection of existing algorithms. We next develop a dynamic indexing algorithm for real-time influence analysis in evolving networks in Chapter 6. Then, we next present a reduction algorithm for massive-scale influence graphs with billions of edges in Chapter 7. Further, in Chapter 8, we consider a portfolio optimization problem aiming at maximization of the conditional value at risk of the size of influence

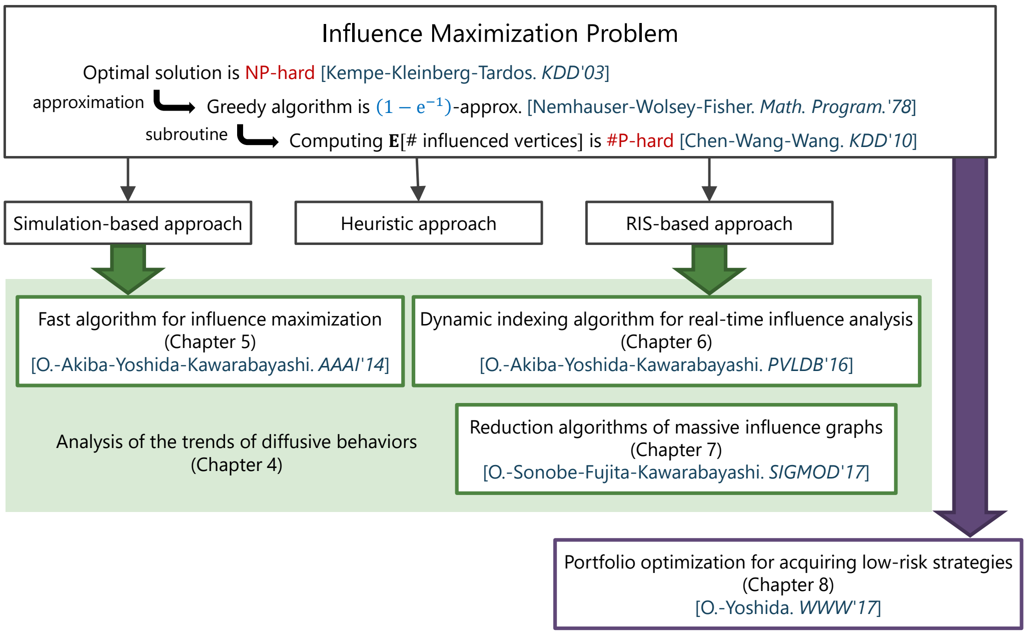

Influence Maximization Problem
Optimal solution is NP-hard [Kempe-Kleinberg-Tardos. KDD'03]
approximation
Greedy algorithm is $(1 - e^{-1})$-approx. [Nemhauser-Wolsey-Fisher. Math. Program.'78]
subroutine
Computing $\mathbf{E}$[# influenced vertices] is #P-hard [Chen-Wang-Wang. KDD'10]
Simulation-based approach
Heuristic approach
RIS-based approach
Fast algorithm for influence maximization (Chapter 5) [O.-Akiba-Yoshida-Kawarabayashi. AAAI'14]
Dynamic indexing algorithm for real-time influence analysis (Chapter 6) [O.-Akiba-Yoshida-Kawarabayashi. PVLDB'16]
Analysis of the trends of diffusive behaviors (Chapter 4)
Reduction algorithms of massive influence graphs (Chapter 7) [O.-Sonobe-Fujita-Kawarabayashi. SIGMOD'17]
Portfolio optimization for acquiring low-risk strategies (Chapter 8) [O.-Yoshida. WWW'17]

Figure 1.2: Illustration of this thesis.

diffusion, and propose a polynomial-time approximation algorithm for it. Finally, we conclude the thesis in Chapter 9.

Chapter 2

Preliminaries

In this chapter, we will introduce several notations, definitions, and tools used throughout this thesis. Section 2.1 and 2.2 provide basic definitions and algorithms related to graphs. Section 2.3 explains the notion of submodularity. Section 2.4 and 2.5 give the definition of diffusion models and influence maximization and its computational complexity. Sections 2.6 and 2.7 explain concepts and tools related to Chapter 8. Table 2.1 summarizes the notation used in this thesis.

Table 2.1: Notations frequently used throughout this thesis.

notation	**description**
$G = (V, E)$	graph
$V(G), E(G)$	vertex set and edge set of graph G
$\mathcal{G} = (V, E, p)$	influence graph
(u, v)	edge connecting u to v
$p(u, v)$	influence probability of edge (u, v)
$\mathsf{w}(v)$	weight of vertex v
$\mathcal{N}^-_G(v), \mathcal{N}^+_G(v)$	set of in- and out-neighbors of vertex v in graph G
$\mathsf{R}_G(v)$	reachable set of vertex v in graph G
$\mathsf{r}_G(v)$	size of reachable set of vertex v in graph G
$\mathsf{Inf}_{\mathcal{G}}(S)$	influence spread of vertex set S in influence graph G

2.1 Definitions and Notations

2.1.1 Set and Partitions

For a positive integer k, let $[k]$ denote the set $\{1, 2, \ldots, k\}$. For a set V and an integer $k \leq |V|$, we define $\binom{V}{k}$ as a collection of all k-element subsets of V. A collection $\mathcal{P}$ of sets is called a *partition* of a set S if $\mathcal{P}$ does not contain the empty set $\emptyset$, the union of the sets in $\mathcal{P}$ equals S, and any two distinct sets in $\mathcal{P}$ are disjoint. For two partitions $\mathcal{P}$ and $\mathcal{Q}$ of the same set, if every element in $\mathcal{P}$ is a subset of some element in $\mathcal{Q}$, we say that $\mathcal{P}$ is a *refinement* of $\mathcal{Q}$, or $\mathcal{P}$ ($\mathcal{Q}$) is *finer* (*coarser*) than $\mathcal{Q}$ ($\mathcal{P}$). The *meet* of $\mathcal{P}$ and $\mathcal{Q}$, denoted $\mathcal{P} \wedge \mathcal{Q}$, is defined as the coarsest partition that is finer than both $\mathcal{P}$ and $\mathcal{Q}$.

2.1.2 Graphs

In this thesis, we consider problems on graphs. We first define directed and undirected graphs.

Definition 2.1 (Directed graph)**.** *A* directed graph *is defined as a pair $G = (V, E)$, where V is a finite set and $E \subseteq V \times V$.*

Definition 2.2 (Undirected graph)**.** *An* undirected graph *is defined as a pair $G = (V, E)$, where V is a finite set and $E \subseteq \binom{V}{2}$.*

An element in V is called a *vertex* and an element in E is called an *edge*. We denote the set of vertices and edges of a graph G by $V(G)$ and $E(G)$, respectively.

In an undirected graph, an edge is an ordered pair $\{u, v\}$ with $u, v \in V$ and $u \neq v$. However, in order to use the same notation, we will denote $\{u, v\}$ by (u, v) in accordance with convention, and hence, (u, v) and (v, u) in an undirected graph represent the identical edge. Directed graphs and undirected graphs are often both simply referred to as graphs. In general, we consider directed graphs since information flow is not mutual but directed.

Definition 2.3 (Connect, leave, and enter)**.** *For a graph $G = (V, E)$ and an edge $(u, v) \in E$ consisting of two vertices $u, v \in V$, we say that (u, v)* connects *u to v,* leaves *u, or* enters *u.*

For an undirected graph, we sometimes consider its *directed version*.

Definition 2.4 (Directed version)**.** *For an undirected graph $G = (V, E)$, the* directed version *of G is defined as a directed graph $G' = (V, E')$, where E' contains a directed edge (u, v) if and only if $(u, v) \in E$.*

In other words, for an undirected graph $G = (V, E)$, the edge set of its directed version is obtained from E by replacing each undirected edge $(u, v) \in E$ with a pair of two directed edges (u, v) and (v, u).

Definition 2.5 (Subgraph)**.** *For two graphs $G = (V, E)$ and $G' = (V', E')$, G' is called a* subgraph *of G if $V' \subseteq V$ and $E' \subseteq E$.*

Definition 2.6 (Induced subgraph)**.** *For a graph $G = (V, E)$ and a vertex set $V' \subseteq V$, the subgraph of G* induced *by V' is defined as a subgraph (V', E') of G, where E' consists of all edges in E spanned by V', and we denote it by $G[V']$. Moreover, we use $G - V'$ to denote the subgraph of G induced by $V \setminus V'$, i.e., $G - V' := G[V \setminus V']$.*

We sometimes deal with graphs where *weights* are associated to each vertex.

Definition 2.7 (Vertex weights)**.** *For a graph $G = (V, E)$, vertex weights are a mapping w from each vertex in V to a positive number. We denote the weight of a vertex $v \in V$ by $\mathsf{w}(v)$.*

2.1.3 Local Properties

We here define the notion of *neighbors* and *degrees*.

Definition 2.8 (Neighbor)**.** *For a graph $G = (V, E)$ and two vertices $u, v \in V$, u is called an* in-neighbor *of v if $(u, v) \in E$ and u is called an* out-neighbor *of v if $(v, u) \in E$. The set of in-neighbors and out-neighbors of a vertex v is denoted by $\mathcal{N}^-_G(v)$ and $\mathcal{N}^+_G(v)$, i.e., $\mathcal{N}^-_G(v) = \{u \mid (u, v) \in E\}$ and $\mathcal{N}^+_G(v) = \{u \mid (v, u) \in E\}$, respectively.*

Definition 2.9 (Degree)**.** *For a graph $G = (V, E)$ and a vertex $v \in V$, the* in-degree *and* out-degree *of v are defined as the number of out-neighbors and in-neighbors, i.e., $|\mathcal{N}^+_G(v)|$ and $|\mathcal{N}^-_G(v)|$, respectively.*

Note that if G is undirected, then $\mathcal{N}^-_G(v)$ and $\mathcal{N}^+_G(v)$ are identical, and hence in-neighbor and out-neighbor are simply referred to as *neighbor*, and in-degree and out-degree are simply referred to as *degree*. We often omit the subscripts when the graph is clear from the context.

2.1.4 Paths, Reachability, and Connectivity

We then define the notion of *paths*, *reachability*, and *connectivity*.

Definition 2.10 (Path)**.** *For a graph $G = (V, E)$ and two vertices $s, t \in V$, a* path *from s to t is a finite sequence $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ of edges in E, where $v_0 = s$ and $v_k = t$, and the v_i's are all distinct except for the first and last.*

Definition 2.11 (Length)**.** *The* length *of a path is defined as its number of edges.*

Definition 2.12 (Shortest path)**.** *For a graph $G = (V, E)$, and two vertices $s, t \in V$, a path from s to t is called a* shortest path *if its length is minimum among all paths from s to t.*

Next, we define the notion of *reachability*, which plays an essential role in the process of diffusion.

Definition 2.13 (Reachable)**.** *For a graph $G = (V, E)$ and two vertices $u, v \in V$, if there is a path from u to v in a graph G, we say that v is* reachable *from u, or u* can reach *v.*

Definition 2.14 (Reachable set)**.** *For a graph $G = (V, E)$ and a vertex $v \in V$, the reachable set of v, denoted $\mathsf{R}_G(v)$, is defined as the set of the vertices reachable from v.*

For a graph G and a vertex $v \in V$, we denote the size of the reachable set of v by $\mathsf{r}_G(v)$, i.e., $\mathsf{r}_G(v) = |\mathsf{R}_G(v)|$. If vertex weights w are associated with G, we denote the total weight of vertices reachable from v by $\mathsf{r}_{G,\mathsf{w}}(v)$, i.e., $\mathsf{r}_{G,\mathsf{w}}(v) = \sum_{w \in \mathsf{R}_G(v)} \mathsf{w}(w)$. We abuse the notation to let act R_G, r_G, $\mathsf{r}_{G,\mathsf{w}}$ on a vertex set $S \subseteq V(G)$ by writing $\mathsf{R}_G(S) = \bigcup_{v \in S} \mathsf{R}_G(v)$, $\mathsf{r}_G(S) = |\mathsf{R}_G(S)|$, and $\mathsf{r}_G(S) = \sum_{w \in \mathsf{R}_G(S)} \mathsf{w}(w)$, respectively.

We then go into the notion of *connectivity.*

Definition 2.15 (Strongly connected)**.** *For a graph $G = (V, E)$ and two vertices $u, v \in V$, we say that u and v are* strongly connected *if they can reach each other. We say that G is* strongly connected *if any pair of vertices is strongly connected.*

Definition 2.16 (Weakly connected)**.** *For a graph $G = (V, E)$ and two vertices $u, v \in V$, we say that u and v are* weakly connected *if they are strongly connected in the graph obtained from G by replacing each edge $(u, v) \in E$ by a pair of edges (u, v) and (v, u). We say that G is* weakly connected *if any pair of vertices is weakly connected.*

Note that an undirected graph is strongly connected if and only if it is weakly connected.

The strongly connected relation is an equivalence relation, and the collection of its equivalence classes forms a partition of the whole vertex set. Each of its equivalence classes is called a *strongly connected component.*

Definition 2.17 (Strongly connected component)**.** *For a graph $G = (V, E)$, a* strongly connected component (SCC) *is a maximal vertex set that is strongly connected.*

Definition 2.18 (Directed acyclic graph)**.** *A directed graph is called a* directed acyclic graph (DAG) *if no two distinct vertices are strongly connected.*

We now introduce the *condensation* of a directed graph, which is obtained by contracting each SCC in the graph into a single vertex.

Definition 2.19 (Condensation). *For a directed graph $G = (V, E)$, let $\mathcal{P} = C_1, \ldots, C_\ell$ be a collection of the SCCs in G. Then, the* condensation *of G is defined as a directed graph that contains vertices c_i corresponding to each SCC C_i and contains edges (c_i, c_j) if and only if $c_i \neq c_j$ and there is at least one edge connecting a vertex in C_i to a vertex in C_j. The* correspondence mapping *associated with the condensation is defined as a mapping $\pi : V \to V'$, which satisfies $\pi(v) = c_j$ such that $v \in C_j$.*

For a directed graph G, let G' be the condensation of G and π be the correspondence mapping associated with G'. Then,

$$V(G') = \{\pi(v) \mid v \in V\}, \tag{2.1}$$

$$E(G') = \{(\pi(u), \pi(v)) \mid \pi(u) \neq \pi(v),\ (u, v) \in E\}. \tag{2.2}$$

Notice also that for two vertices $u, v \in V$, u can reach v in G if and only if $\pi(u)$ can reach $\pi(v)$ in G'. We abuse the notation to let act π on a subset $S \subseteq V$ by writing $\pi(S) = \{\pi(v) \mid v \in S\}$.

We here also define vertex weights associated with the condensation. For a directed graph $G = (V, E)$, let $G' = (V', E')$ be the condensation of G, where $V' = \{c_1, \ldots, c_\ell\}$ and π be the associated correspondence mapping. Then, for each vertex $c_j \in V'$, we define the weight of c_j as the number of vertices in V that are mapped to c_j via π, i.e., $\mathsf{w}(c_j) = |C_j|$. Then, for any vertex set $S \subseteq V$, it holds that

$$\mathsf{r}_{G',\mathsf{w}}(\pi(S)) = \mathsf{r}_G(S). \tag{2.3}$$

2.1.5 Trees

Definition 2.20 (Tree). *An undirected graph $G = (V, E)$ is called a* tree *if it is weakly connected and $|E| = |V| - 1$.*

Definition 2.21 (Directed tree). *A directed graph $G = (V, E)$ is called a* directed tree *rooted at a vertex $z \in V$ if z can reach every vertex in V and $|E| = |V| - 1$.*

2.1.6 Influence Graph

We then introduce *influence graphs* to capture probabilistic processes among vertices.

Definition 2.22 (Influence graph). *An* influence graph *is defined as a triplet $\mathcal{G} = (V, E, p)$, where V is a set of vertices, E is a set of edges, and $p : E \to (0, 1]$ is* influence probabilities.

The influence probability represents the magnitude of influence for each edge. Intuitively, the higher the $p(u, v)$, the more v will be affected by u.

Since influence graphs can be regarded as graphs with an edge influence probability function, we use the definitions and notations defined so far for influence graphs. For example, for an influence graph $\mathcal{G} = (V, E, p)$ and a vertex $v \in V$, the set of the in-neighbors of v in $\mathcal{G}$ is defined as the set of the in-neighbors of v in (V, E), i.e., $\mathcal{N}^-_{\mathcal{G}}(v) = \mathcal{N}^-_{(V,E)}(v)$.

Definition 2.23 (Induced influence subgraph). *For an influence graph $\mathcal{G} = (V, E, p)$ and a vertex set $V' \subseteq V$, the influence subgraph of $\mathcal{G}$* induced *by V' is defined as an influence graph (V', E', p'), where (V', E') is a subgraph of G induced by V', and p' is the restriction of p to E'. We denote it by $\mathcal{G}[V']$.*

2.2 Basic Graph Algorithms

2.2.1 Breadth First Search

Breadth first search (BFS) [141] is a classical algorithm for graph search. We especially use BFSes to compute reachable sets. Given a graph $G = (V, E)$ and a *source* vertex $s \in V$, a BFS starting from s finds all vertices reachable from s. We also construct a *breadth first search tree (BFS-tree)*, which is defined as a tree on the reachable set of s in which any path is a shortest path in G.

More precisely, a BFS from s begins with a queue including a single s and a BFS-tree $T = (s, \emptyset)$. In each step, we remove a vertex u from the queue and visit each out-neighbor v of u. If this is the first time v has been visited, then we insert v into the queue and add v and (u, v) to $V(T)$ and $E(T)$, respectively. The procedure continues until the queue is empty. We finally return the set of the visited vertices and the BFS-tree T. The whole algorithm completes in $\mathcal{O}(|E|)$ time and requires $\mathcal{O}(|V|)$ space in addition to the space required to store G.

2.2.2 Depth First Search

Depth first search (DFS) [89, 178] is another classical algorithm for graph search. For a graph $G = (V, E)$ and a source vertex $s \in V$, a DFS starting from s finds the reachable set of s as the same as BFS. Conceptually, DFS recursively explores out-neighbors before backtracking.

More precisely, during a DFS from s, we visit each out-neighbor v of s. If it is the first time v has been visited, then we start a DFS from v recursively. After visiting all out-neighbors of s, we finish the DFS from s. The set of the visited vertices is the reachable set of s. The whole algorithm completes in $\mathcal{O}(|E|)$ time consumes $\mathcal{O}(|V|)$ space in addition to the space required to store G.

2.2.3 Finding Strongly Connected Components

There exist several linear-time algorithms for finding all SCCs of a graph [172, 178]. We explain the algorithm proposed by Sharir [172] which performs DFSes twice.

For a directed graph $G = (V, E)$, we define the *transpose* graph of G as $G^\top = (V, E^\top)$, where $E^\top = \{(v, u) \mid (u, v) \in E\}$. Note that G and $G^\top$ have the identical SCCs.

For a directed graph $G = (V, E)$, the algorithm of Sharir [172] works as follows. We first scan each vertex $v \in V$ in an arbitrary order and conduct a DFS from v on G if we have not visited v during the preceding DFSes on G. We next compute the transposed graph $G^\top$ of G. We then order the vertices of V in ascending order of time when we finish the corresponding DFS. We then scan each vertex $v \in V$ in the obtained order and perform a DFS from v on $G^\top$ if we have not visited v during the preceding DFSes on $G^\top$. Finally, we return the collection of the reachable sets, each of which is an SCC, obtained during the DFSes on $G^\top$. This algorithm completes in $\mathcal{O}(|V| + |E|)$ time and requires $\mathcal{O}(|V| + |E|)$ space.

2.3 Submodular Set Functions

The submodularity is a property of set functions that, informally speaking, the marginal increase of utility obtained by adding a single element to an input set decreases as the input set gets larger. Submodular functions capture *diminishing* *marginal returns*, which arises in natural problem formulations including influence maximization. During the past twenty years, submodular set functions have played an immense role in problem formulation in a wide range of applications, e.g., outbreak detection [23, 118], document summarization [125, 126], network inferring [74], optimizing CPU scheduling [174], and image segmentation [31]. This section presents the definition, properties, and optimization of submodular set functions.

2.3.1 Definitions and Properties

We begin with the definition of submodular set functions.

Definition 2.24 (Submodular function)**.** *For a set function $f : 2^V \to \mathbb{R}$, where V is a finite set, f is said to be* submodular *if*

$$\forall S \subseteq T \subseteq V,\ \forall e \in V \setminus T,\ f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T). \tag{2.4}$$

Another definition of submodular set functions is the following equivalent condition:

$$\forall S \subseteq V,\ \forall T \subseteq V,\ f(S) + f(T) \geq f(S \cup T) + f(S \cap T). \tag{2.5}$$

We often meet the case where the function of interest is *monotone* or *symmetric* as defined below.

Definition 2.25 (Monotone function)**.** *For a set function $f : 2^V \to \mathbb{R}$, where V is a finite set, f is said to be* monotone *if*

$$\forall S \subseteq T \subseteq V,\ f(S) \leq f(T). \tag{2.6}$$

Definition 2.26 (Symmetric function)**.** *For a set function $f : 2^V \to \mathbb{R}$, where V is a finite set, f is said to be* symmetric *if*

$$\forall S \subseteq V,\ f(S) = f(V \setminus S). \tag{2.7}$$

Monotone submodular functions include a class of linear functions, weighted coverage functions, the entropy function over a set of discrete-valued random variables [66], and matroid rank functions [24]. Symmetric (non-monotone) submodular functions include a class of graph cuts and the mutual information.

Submodular functions have useful properties which enable us to express problems of interest flexibly.

- For any submodular functions $f_1, \ldots, f_\ell : 2^V \to \mathbb{R}$ and non-negative numbers $\alpha_1, \ldots, \alpha_\ell$, the *non-negative linear combinations* defined as $g(S) := \sum_{i \in [\ell]} \alpha_i f_i(S)$ is submodular.
- For any submodular function $f : 2^V \to \mathbb{R}$ and any set $A \subseteq V$, the *residual* defined as $g(S) := f(S \cup A) - f(A)$ is submodular.
- For any monotone submodular function $f : 2^V \to \mathbb{R}$ and any constant $c \in \mathbb{R}$, the *truncation* defined as $g(S) := \min\{f(S), c\}$ is submodular.

2.3.2 Monotone Submodular Function Maximization

We consider the problem of maximizing monotone submodular functions. In this section, we assume to be given a *value oracle* of an input function f, a blackbox that returns $f(S)$ for any S. In particular, we deal with a *cardinality constraint*:

$$\text{maximize } f(S) \text{ subject to } |S| \leq k. \tag{2.8}$$

This problem is NP-hard because it includes the NP-hard maximum coverage problem [63] as a special case. In the following, we describe an approximation algorithm for this problem.

Algorithm 2.1 Greedy algorithm for monotone submodular function maximization under a cardinality constraint [144].

```
Input: a monotone submodular set function $f : 2^V \to \mathbb{R}$ and a budget $k$
1: $S_0 \leftarrow \emptyset$.
2: for $\ell = 1$ to $k$ do
3:     $v_\ell \leftarrow \mathrm{argmax}_{v \in V \setminus S_{\ell-1}} f(S_{\ell-1} \cup \{v\}) - f(S_{\ell-1})$.
4:     $S_\ell \leftarrow S_{\ell-1} \cup \{v_\ell\}$.
5: return $S_k$.
```

Algorithm 2.1 shows the gold-standard *greedy algorithm*. Given a monotone submodular function $f : 2^V \to \mathbb{R}$ and a budget k, it begins with an empty set $S_0 = \emptyset$, and for each iteration $\ell \in [k]$, it picks up an element v_ℓ that makes the maximum increase of f, i.e., $v_\ell = \mathrm{argmax}_{v \in V \setminus S_{\ell-1}} f(S_{\ell-1} \cup \{v\}) - f(S_{\ell-1})$, and adds it to the solution, i.e., $S_\ell = S_{\ell-1} \cup \{v_\ell\}$. Note that this algorithm accesses a value oracle of f at most $k|V|$ times. Theorems 2.27 guarantees that the greedy algorithm approximates the optimum solution within a factor slightly better than 63%.

Theorem 2.27 (Nemhauser, Wolsey, and Fisher [144])**.** *For a non-negative, monotone, and submodular function $f : 2^V \to \mathbb{R}$, let $S \subseteq V$ be a set of size k obtained by the greedy algorithm. Then, it holds that $f(S) \geq (1 - \mathrm{e}^{-1}) f(S^*)$, where $S^* \subseteq V$ is the optimal solution of size k.*

Remark that a factor of $(1 - \mathrm{e}^{-1})$ is tight in the worst case due to the inapproximability result of the maximum coverage problem [63]. However, for practical instances, it provides better solutions than those expected from the bound [110, 118, 173].

2.4 Diffusion Models

Diffusion models define the process by which influence or information diffuses over a network. We review two well-established probabilistic diffusion models: *independent cascade* and *linear threshold*.

2.4.1 Independent Cascade Model

The *independent cascade (IC)* model was formed by Goldenberg, Libai, and Muller [70, 71]. This model mimics the dynamics of infectious diseases and generalizes a susceptible-infected-recovered (SIR) model [101] of epidemics.

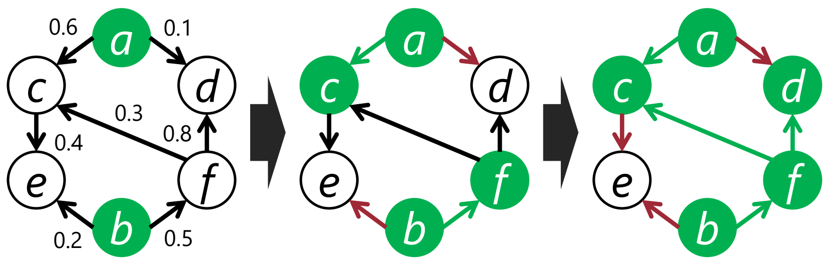


Figure 2.1: Illustration of the IC model. Green and white vertices are active and inactive, respectively. Green, red, black edges correspond to successful activation trials, failed activation trials, and undetermined trials, respectively.

Definition.

In the IC model, each vertex takes either of two states, *active* and *inactive*. An inactive vertex may become active, but not vice versa. Given an influence graph $\mathcal{G} = (V, E, p)$ and a *seed set* $S \subseteq V$, the diffusion process begins by activating vertices in S; all the other vertices are inactive. Then the process unfolds in discrete steps according to the following "randomized" rule. When a vertex u becomes active for the first time in the step t, it is given a single chance to activate each current inactive vertex v among u's out-neighbors. It succeeds with probability $p(u, v)$. If u succeeds, then v will become active in the step $t + 1$. Whether or not u succeeds, it cannot make any further attempt to activate v in subsequent steps. The process runs until no more activation is possible. Note that this diffusion process terminates in finite steps. Figure 2.1 shows an example of the IC process.

In order to optimize social influence, we now define the notion of *influence spread.*

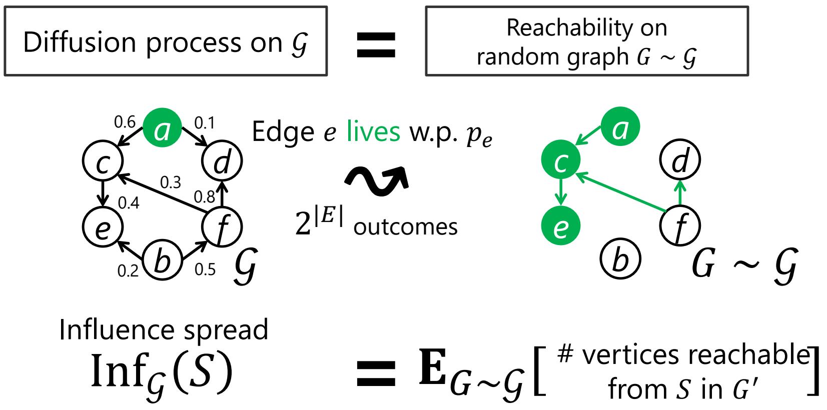
Definition 2.28 (Influence spread)**.** *For an influence graph $\mathcal{G} = (V, E, p)$ and a vertex set $S \subseteq V$, the* influence spread *of S in $\mathcal{G}$, denoted $\mathsf{Inf}_{\mathcal{G}}(S)$, is defined as the expected number of active vertices by initially activating vertices in S. Moreover, given vertex weights w, the* influence spread *of S in $\mathcal{G}$, denoted $\mathsf{Inf}_{\mathcal{G},\mathsf{w}}(S)$, is defined as the expected total weight of active vertices by initially activating vertices in S.*

$\mathsf{Inf}_{\mathcal{G}}(\cdot)$ can be viewed as a function on a subset of vertices, and thus we often call $\mathsf{Inf}_{\mathcal{G}} : 2^V \to \mathbb{R}_{\geq 0}$ an *influence function.*

Here, we describe the *random-graph interpretation* [98] of the IC model that characterizes its diffusion process (Figure 2.2). For an influence graph $\mathcal{G} = (V, E, p)$, consider the distribution over graphs (V, E'), where E' is obtained from E by maintaining each edge e with probability $p(e)$. We use $G \sim \mathcal{G}$ to mean that G is a *random graph* sampled from the distribution. Hereafter, for an influence probability function $p : E \to (0, 1]$ and two edge subsets X and Y with $X \subseteq Y \subseteq E$, we denote the probability of obtaining X from Y by maintaining each edge with its influence probability as $p(X \mid Y)$, i.e.,

$$p(X \mid Y) = \prod_{e \in X} p(e) \prod_{e \in Y \setminus X} (1 - p(e)). \tag{2.9}$$

Note that the probability of sampling a fixed graph $G = (V, E')$ from $\mathcal{G}$ is exactly $p(E' \mid E)$. Then, Kempe, Kleinberg, and Tardos [98] proved that the influence of a seed set S in $\mathcal{G}$ is equal to the expected number of vertices reachable from S in



Diffusion process on $\mathcal{G}$
=
Reachability on random graph $G \sim \mathcal{G}$
0.6
a
0.1
c
d
0.3
0.4
0.8
e
f
b
0.2
0.5
$\mathcal{G}$
Edge e lives w.p. p_e
$2^{|E|}$ outcomes
$G \sim \mathcal{G}$
Influence spread
$\mathrm{Inf}_{\mathcal{G}}(S)$
$= \mathbf{E}_{G \sim \mathcal{G}}$ [# vertices reachable from S in G']

Figure 2.2: Random-graph interpretation of the IC model.

the random graph sampled from $\mathcal{G}$, which is expressed as

$$\mathsf{Inf}_{\mathcal{G}}(S) = \mathop{\mathbf{E}}_{G \sim \mathcal{G}}[\mathsf{r}_G(S)] = \sum_{E' \subseteq E} p(E' \mid E) \cdot \mathsf{r}_{(V,E')}(S). \tag{2.10}$$

If we have weights w for vertices in V, the following holds:

$$\mathsf{Inf}_{\mathcal{G},\mathsf{w}}(S) = \mathop{\mathbf{E}}_{G \sim \mathcal{G}}[\mathsf{r}_{G,\mathsf{w}}(S)] = \sum_{E' \subseteq E} p(E' \mid E) \cdot \mathsf{r}_{(V,E'),\mathsf{w}}(S). \tag{2.11}$$

Random-graph interpretation.

2.4.2 Linear Threshold Model

The *linear threshold (LT)* model was proposed by Kempe, Kleinberg, and Tardos [98]. Intuitively, an inactive vertex becomes active when a sufficiently large fraction of its neighbors becomes active. Note that Granovetter [83] and Schelling [171] were the first to propose models that reflect this process.

Definition.

In the LT model, each vertex takes active or inactive, in the same way as the IC model. Given an influence graph $\mathcal{G} = (V, E, p)$ where an edge weight p satisfies that $\sum_{u \in \mathcal{N}^-(v)} p(u, v) \leq 1$, and a seed set $S \subseteq V$, we first activate vertices in S. We next assign an *activation threshold* θ_v for each vertex v chosen from the interval $[0, 1]$ uniformly at random. Then, the process unfolds in discrete steps according to the following "deterministic" rule. In the step t, an inactive vertex v will become active in the next step $t + 1$ when the following holds:

$$\sum_{u \in \mathcal{N}^-(v) : u \text{ is active}} p(u, v) \geq \theta_v. \tag{2.12}$$

The process runs until no more activation is possible. Note that this diffusion process terminates in finite steps.

Random-graph interpretation.

Similar to the IC model, the LT model has a random-graph interpretation. Consider the randomized process in which each vertex picks up at most one entering edge with probability equal to the edge weight. Let E' be a set consisting of the all selected edges, and we consider a distribution over possible subgraphs (V, E'). Then, the influence spread of any vertex set S is equal to the expected number of vertices reachable from S over the possible subgraphs [98].

2.5 Influence Maximization

2.5.1 Definition

Now that, we start this section by mathematically formulating the influence maximization problem as a discrete optimization problem according to [98].

Problem 1 (Influence maximization problem [98])**.** *Given an influence graph $\mathcal{G} = (V, E, p)$, a diffusion model, and a seed size k, the* influence maximization problem *asks to find a seed set $S \subseteq V$ of k vertices such that the influence spread of S under the give diffusion model is maximized.*

Besides influence maximization, estimating the influence spread of a particular seed set itself is important.

Problem 2 (Influence estimation problem [131])**.** *Given an influence graph $\mathcal{G} = (V, E, p)$, a diffusion model, and a seed set $S \subseteq V$, the* influence estimation problem *asks to compute the influence spread of S under the given diffusion model.*

2.5.2 Hardness Results

The complexity of influence maximization and influence estimation has been thoroughly analyzed. We begin with the inapproximability of influence maximization in the general case.

Theorem 2.29 ([98, Theorem 4.1])**.** *In general, it is NP-hard to approximate the influence maximization problem within a factor of $|V|^{1-\epsilon}$ for any $\epsilon > 0$.*

The proof was done by a reduction from the NP-complete set cover problem. Hereafter, we focus on the IC model and the LT model.

Theorem 2.30 ([98, Theorem 2.4 and Theorem 2.7])**.** *Influence maximization under both the IC and LT models is NP-hard. Moreover, it is NP-hard even if an oracle for the influence function is given.*

The proof was done by showing that influence maximization under the IC and LT model includes the NP-complete set cover problem and the NP-complete vertex cover problem as a special case, respectively.

In fact, influence estimation is even difficult under both the IC and LT model.

Theorem 2.31 ([182, Theorem 1] and [41, Theorem 1])**.** *Given an influence graph and a seed set, it is #P-hard to compute the influence spread of the seed set under both IC model and LT model.*

The proof for the IC model was done by a reduction from the #P-complete counting problem of s-t connectedness in a directed graph [181], while the proof for the LT model was done by a reduction from the #P-complete counting problem of simple paths in a directed graph [181].

Utilizing the above results, we can show that solving influence maximization with $k = 1$, i.e., identifying the most influential vertex is even hard.

Theorem 2.32 ([44, Corollary 3.3])**.** *Influence maximization is #P-hard under both IC and LT models, even if $k = 1$.*

2.5.3 Approximability Results

While the exact computation of the two problems is quite difficult, it is possible to obtain approximate solutions. We begin with the celebrated result of Kempe, Kleinberg, and Tardos [98] as follows.

Theorem 2.33 ([98, Theorem 2.2]). *For any influence graph, the influence function is non-negative, monotone, and submodular under both IC and LT models.*

Hence, if we are given an oracle for an influence function, running the greedy algorithm on the influence function derives a $(1 - \mathrm{e}^{-1})$-approximate solution against the optimal solution. Fortunately, albeit #P-hardness of influence estimation, we are able to approximate the influence spread by performing Monte-Carlo simulations.

Theorem 2.34 ([100, Proposition 4.1]). *If we simulate the diffusion process starting from a vertex set $A \subseteq V$*

$$\Theta(\epsilon^{-2}|V|^2 \ln \delta^{-1}) \tag{2.13}$$

times, then the estimate is a $(1 \pm \epsilon)$-approximation to $\mathsf{Inf}(A)$ with probability at least $1 - \delta$.

Combining the above two theorems, we have the main result of [100].

Theorem 2.35 ([100, Theorem 1.1]). *In the IC and LT models, there is a randomized polynomial-time algorithm that returns a seed set that approximates the maximum influence spread within a factor of $(1 - \mathrm{e}^{-1} - \epsilon)$.*

2.6 Risk Measures

A *risk measure* ρ of a random variable X is defined as a function mapping X to a real number. We use $\rho[X]$ to denote the risk of X. X usually represents loss and hence we want to avoid a large loss. However, we regard X as profit in Chapter 8 because we want to avoid to have the number of influenced individuals small. Hence, we use slightly different definitions from the standard ones.

2.6.1 Definition of Coherent Risk Measures

The notion of coherence defines the properties that risk measures should have. We first define several properties on risk measures.

Definition 2.36 (Monotonicity). *A risk measure ρ is said to be* monotone *if for any two random variables X_1, X_2,*

$$\mathbf{Pr}[X_1 \leq X_2] = 1 \Rightarrow \rho[X_1] \leq \rho[X_2]. \tag{2.14}$$

Definition 2.37 (Super-additivity). *A risk measure ρ is said to be* super-additive *if for any two random variables X_1, X_2,*

$$\rho[X_1 + X_2] \geq \rho[X_1] + \rho[X_2]. \tag{2.15}$$

Definition 2.38 (Homogeneity). *A risk measure ρ is said to be* homogeneous *if for any random variable X and any non-negative number α,*

$$\rho[\alpha X] = \alpha\rho[X]. \tag{2.16}$$

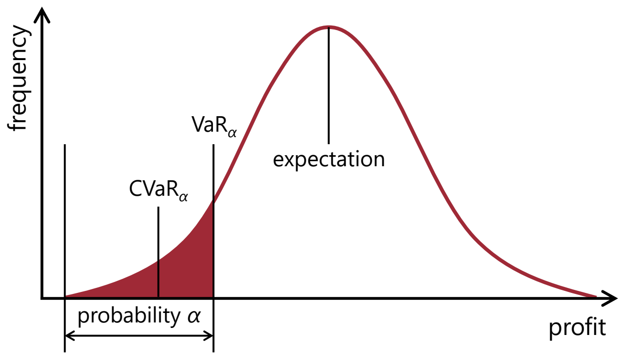


Figure 2.3: Illustration of expectation, value at risk, and conditional value at risk.

Definition 2.39 (Translation invariance). *A risk measure ρ is said to be* translation invariant *if for any random variable X and any real number α,*

$$\rho[X + \alpha] = \rho[X] + \alpha. \tag{2.17}$$

Then, *coherent risk measures* are defined as follows.

Definition 2.40 (Coherent risk measure [16]). *A risk measure is said to be a* coherent risk measure *if it satisfies monotonicity, sub-additivity, homogeneity, and translational invariance.*

2.6.2 Examples of Risk Measures

Here, we define two popular risk measures (Figure 2.3).

Definition 2.41 (Value at risk). *For a random variable X and $\alpha \in (0, 1)$, let $F_X : \mathbb{R} \to \mathbb{R}$ be the cumulative distribution function of the distribution of X. Then, the* value at risk (VaR) *of X at a significance level α, denoted by $\mathsf{VaR}_\alpha[X]$, is the α-percentile of X, i.e.,*

$$\mathsf{VaR}_\alpha[X] = \inf\{\tau \in \mathbb{R} \mid F_X(\tau) \geq \alpha\}. \tag{2.18}$$

It is known that VaR is not a coherent risk measure since it does not satisfy the super-additivity [16].

We then define *conditional value at risk (CVaR)*, which is also called expected shortfall, expected tail loss.

Definition 2.42 (Conditional value at risk [34]). *For a random variable X and $\alpha \in (0, 1)$, The* conditional value at risk (CVaR) *of X at a significance level α, denoted by $\mathsf{CVaR}_\alpha[X]$, is defined as*

$$\mathsf{CVaR}_\alpha[X] = \frac{1}{\alpha}\int_0^\alpha \mathsf{VaR}_\gamma[X]\mathrm{d}\gamma. \tag{2.19}$$

Theorem 2.43 ([8]). *CVaR is a coherent risk measure.*

It is known that $\mathsf{CVaR}_\alpha[X]$ can be written as a solution to the following optimization problem [163]:

$$\mathsf{CVaR}_\alpha[X] = \max_{\tau\in\mathbb{R}}\left\{\tau - \frac{1}{\alpha}\mathop{\mathbf{E}}_X[\max\{\tau - X, 0\}]\right\}. \tag{2.20}$$

Algorithm 2.2 The multiplicative weights update algorithm [15].

```
1: fix $\eta \leq 1/2$ and set $\mathbf{w}^{(1)} \leftarrow \mathbf{1}$.
2: for $t = 1, 2, \ldots, T$ do
3:     choose a strategy $i \in S$ with probability $d_i^{(t)} \leftarrow \mathbf{w}_i^{(t)} / \|\mathbf{w}^{(t)}\|_1$.
4:     observe the costs of the strategies $c_1^{(t)}, \ldots, c_s^{(t)}$.
5:     for every strategy $i \in S$, set $w_i^{(t+1)} \leftarrow w_i^{(t)}(1 - \eta c_i^{(t)})$.
```

The maximum is attained by choosing $\tau = \mathsf{VaR}_\alpha[X]$ [163].

We finally explain how to approximate CVaR with the empirical CVaR. Let X be a random variable. For a positive integer r, we define the *empirical distribution* made by r samples from X, denoted $\hat{\mathcal{D}}_{X,r}$, as the uniform distribution over $\{X^1, \ldots, X^r\}$, where $X^1, \ldots, X^r$ are independent samples from the distribution of X. Let $\hat{X} \sim \hat{\mathcal{D}}_{X,r}$ be a random variable. Then, the gap $\mathsf{CVaR}_\alpha[X]$ and $\mathsf{CVaR}_\alpha[\hat{X}]$ is bounded as follows.

Lemma 2.44 ([155, Lemma 4.1]). *Let X be a discrete random variable bounded in $[0,1]$ and $\alpha, \epsilon, \delta \in (0,1)$. Let $\hat{X} \sim \hat{\mathcal{D}}_{X,r}$ be a random variable for $r = \Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$. Then, with probability at least $1-\delta$, we have*

$$|\mathsf{CVaR}_\alpha[X] - \mathsf{CVaR}_\alpha[\hat{X}]| \leq \epsilon. \tag{2.21}$$

Notice that this technical contribution has been derived by Yuchi Yoshida, a co-author of the extended abstract [155] published in WWW 2017.

In Chapter 8, we adopt CVaR as a risk measure to be optimized.

2.7 Multiplicative Weights Update Algorithm

This section explains the multiplicative weights update algorithm [15]. It is a meta-algorithm based on the idea that maintains a distribution over a set of decisions and updates it by multiplying factors determined based on the feedback obtained by running some other algorithm over the current distribution. This algorithm has been repeatedly rediscovered in diverse fields such as machine learning [65], optimization [14, 159], and game theory [84], and we describe the one unified by Arora, Hazan, and Kale [15].

Consider the following setting. We have a set S of r strategies, and we are required to select one strategy from S in each round. More specifically, in round t, we select a vector $\mathbf{d}^{(t)}$ in the r-dimensional simplex $\Delta_r = \{\mathbf{d} \in \mathbb{R}^r \mid \sum_{i \in [r]} d_i = 1\}$. Then, we sample a strategy $i \in S$ from the distribution determined by $\mathbf{d}^{(t)}$, i.e., we sample $i \in S$ with probability $d_i^{(t)}$. Each strategy incurs a certain cost, determined by nature or an adversary. After devising our strategy, all the costs are revealed in the form of the vector $\mathbf{c}^{(t)} \in \mathbb{R}^r$. The expected cost to the algorithm using the vector $\mathbf{d}^{(t)}$ is $\langle \mathbf{d}^{(t)}, \mathbf{c}^{(t)} \rangle$. Hence, after T rounds, the total expected cost is $\sum_{t \in [T]} \langle \mathbf{d}^{(t)}, \mathbf{c}^{(t)} \rangle$.

We wish to obtain an algorithm that achieves a total expected cost not too much more than the cost of the best single strategy, that is, $\min_{i \in S} \sum_{t \in T} c_i^{(t)}$. The *multiplicative weights update (MWU) algorithm* shown in Algorithm 2.2 is known to have this property. More specifically, we obtain the following:

Theorem 2.45 ([15, Corollary 4]). *Assume that all costs $c_i^{(t)} \in [-1, 1]$. By choosing $T = \frac{16 \log r}{\epsilon^2}$ and $\eta = \min\{\frac{\epsilon}{4}, \frac{1}{2}\}$, the MWU algorithm guarantees that,* *after T rounds, for any strategy $i \in S$,*

$$\frac{1}{T}\sum_{t\in[T]}\langle \mathbf{c}^{(t)}, \mathbf{d}^{(t)}\rangle \leq \frac{1}{T}\sum_{t\in[T]} c_i^{(t)} + \epsilon. \tag{2.22}$$

Chapter 3

Categorization of Influence Maximization Algorithms

In this chapter, we give a comprehensive review of existing research on efficient algorithms for influence maximization, some of which also deal with influence estimation.

3.1 Greedy algorithm of Kempe, Kleinberg, and Tardos [98]

Kempe, Kleinberg, and Tardos [98] were the first to propose an approximation algorithm. Their algorithm is an application of the greedy algorithm for monotone submodular functions (Algorithm 2.1) with $f = \mathsf{Inf}$. Due of the monotonicity and submodularity of the influence function under the IC and LT model (Theorem 2.33), the greedy algorithm given an oracle for the influence function provides a $(1 - e^{-1})$-approximate solution (Theorem 2.27). However, the difficulty of influence estimation poses an obstacle to run the above greedy algorithm. Kempe, Kleinberg, and Tardos [98] have remained efficient computation of the influence spread as an open problem and relied on Monte-Carlo simulations. While Monte-

Table 3.1: Categorization of existing influence maximization algorithms.

category	**perspective**	**representatives**
simulation	naive simulations	CELF [118], CELF++ [80], UBLF [191, 192], SIEA [150, 151].
	snapshot-based	Bond Percolation [106], NewGreedy [40], MixedGreedy [40], StaticGreedy and StaticGreedyDU [46], SKIM [52].
RIS [29]	bounding optimal influence	TIM$^+$ [176], IMM [177].
	degree-based thresholding	LISA [57, 149], BCT [147, 152].
	search and verify	SSA and D-SSA [148], TipTop [123].
heuristic	local region	SPM and SP1M [105], DegreeDiscount [40], CGA [184], PMIA [41, 182], LDAG [42], Simpath [81], SAEDV [94], CDH-Kcut and CDH-SHRINK [45], CINEMA [122], IPA [104].
	linear systems	GSbyStep [189], IRIE [96], IMRank [47].
	graph reduction	coarseNet [161], Spine [138].
	others	Belief Propagation [153], Inclusion-Exclusion [190], EaSyIM [67].

Algorithm 3.1 Naive algorithm for influence estimation.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a seed set $S$, number of simulations $r$
 1: inf ← 0.
 2: for $i = 1$ to $r$ do
 3:     $Q \leftarrow$ a queue containing vertices in $S$.
 4:     while $Q \neq \emptyset$ do
 5:         remove $u$ from $Q$.
 6:         inf ← inf + 1.
 7:         for all $v \in \mathcal{N}^{+}_{\mathcal{G}}(u)$ do
 8:             if $v$ is not visited so far and activation succeeds with probability $p(u, v)$ then
 9:                 insert $v$ onto $Q$.
10: return inf$/r$.
```

Carlo simulations provide accurate estimates (Theorem 2.34), running them for $\mathcal{O}(k|V|)$ vertex sets is computationally prohibitive even for small networks. In fact, exact computation of the influence spread was proven to be #P-hard by Chen, Wang, and Wang [41].

Under these circumstances, most of the research on scalable influence maximization have studied a way to efficiently and accurately estimate the influence spread. Table 3.1 gives a taxonomy of existing algorithms for influence maximization. Existing algorithms can be categorized into the following three approaches: simulation-based, RIS-based, and heuristic.

3.2 Simulation-based Algorithms

The first category is to run *Monte-Carlo simulations* of the diffusion process. We first explain *naive estimation* and then describe *snapshot-based estimation*, which adopts sample average approximation.

3.2.1 Naive Estimation

Concept.

Naive estimation methods repeatedly simulate the diffusion process for a given seed set and take the average number of activated vertices. Algorithm 3.1 shows its pseudocode. The running time is bounded by $\mathcal{O}(|E|)$ and a naive implementation consumes $\mathcal{O}(|V| + |E|)$ space. However, since each simulation touches $\mathsf{Inf}(S)$ vertices in expectation, the larger influence probabilities are, the slower each simulation becomes. Theorem 2.34 guarantees that $r = \Omega(\frac{|V|^2}{\epsilon^2} \ln \delta^{-1})$ simulations give a $(1 \pm \epsilon)$-approximation with probability of at least $1 - \delta$. In practice, tens of thousands of simulations are sufficient to obtain reasonable solutions [12, 98].

Applying to influence maximization and existing techniques.

Basically, naive simulation methods for influence maximization call Algorithm 3.1 multiple times during greedy seed selection. A basic approach for scaling up is to prune unnecessary influence evaluation. *Cost-Effective Lazy Forward (CELF)* proposed by Leskovec, Krause, Guestrin, Faloutsos, VanBriesen, and Glance [118] uses the greedy strategy with *lazy evaluations* [140]. In lazy evaluations, we use the fact that the increase of the influence spread that the addition of a vertex v to a vertex set S makes, i.e., $\mathsf{Inf}(S \cup \{v\}) - \mathsf{Inf}(S)$, decreases as S expands. Let S_ℓ

be an ℓ-vertex set that the greedy algorithm has selected. Then, for any positive integer k, $\mathsf{Inf}(S_\ell \cup \{v\}) - \mathsf{Inf}(S_\ell)$ for any $\ell \in [k-1]$ can be used as an *upper bound* of $\mathsf{Inf}(S_k \cup \{v\}) - \mathsf{Inf}(S_k)$. If an upper bound of $\mathsf{Inf}(S_k \cup \{v\}) - \mathsf{Inf}(S_k)$ is less than the actual value of $\mathsf{Inf}(S_k \cup \{u\}) - \mathsf{Inf}(S_k)$ for some u, then we can correctly declare that v is never selected by the greedy algorithm at that time without evaluating $\mathsf{Inf}(S_k \cup \{v\}) - \mathsf{Inf}(S_k)$. Goyal, Lu, and Lakshmanan [80] proposed *CELF++*, which is a slight improvement of CELF to achieve two times speed-up. However, it is still mandatory to evaluate $\mathsf{Inf}(v)$ for all vertices v at the first iteration. *Upper Bound based Lazy Forward (UBLF)* for the IC model by Zhou, Zhang, Guo, Zhu, and Guo [191] and the LT model by Zhou, Zhang, Guo, and Guo [192] uses linear equations to derive an upper bound of the influence spread.

Near-linear time influence estimation.

Influence Estimator (InfEst) proposed by Lucier, Oren, and Singer [132] ensures a relative error in *almost linear time*. The essential idea is to guess a number τ such that the actual influence is in the interval $[\tau, (1+\epsilon)\tau]$. Note that verifying the statement “the actual influence is at least τ” is possible by simulating the diffusion process $\frac{1}{\tau\epsilon^2}$ times. InfEst runs in $\mathcal{O}(\epsilon^{-2}|V| \log^5 |V|)$ time and produces $(1+\epsilon)$-approximate estimation.

Nguyen, Nguyen, Vu, and Dinh [150, 151] proposed *Scalable Outward Influence Estimation Algorithm (SIEA)*. SIEA introduces an importance sampling technique of the diffusion process and runs in time $\mathcal{O}(\epsilon^{-2}|V| \log |V|)$, which is a $\log^4 |V|$-time improvement over InfEst [132].

3.2.2 Snapshot-based Estimation

Concept.

Since influence maximization can be naturally viewed as stochastic discrete optimization, use of *sample average approximation* [109] is a major approach. In short, snapshot-based algorithms sample snapshots of the diffusion process in advance and optimizes the empirical influence function defined over the snapshots.

Specifically, snapshot-based algorithms sample r random graphs $G_1, \ldots, G_r$ from $\mathcal{G}$. Then, the *empirical influence function* $\hat{\mathsf{Inf}} : 2^V \to \mathbb{R}$ is defined as

$$\hat{\mathsf{Inf}}(S) := \frac{1}{r} \sum_{i \in [r]} \mathsf{r}_{G_i}(S), \tag{3.1}$$

which is equal to the average size of reachable sets. Note that $\hat{\mathsf{Inf}}(S)$ is an unbiased estimator for $\mathsf{Inf}_{\mathcal{G}}(S)$. Algorithm 3.2 shows a vanilla implementation of snapshot-based algorithms. The memory consumption is bounded by $\mathcal{O}(\sum_{i\in[r]} |G_i|) = \mathcal{O}(r(|V| + |E|))$, which is r times worse than naive estimation algorithms.

Notice that the above procedure is almost identical to naive estimation when estimating for a single set; however, when we are aware of estimating for multiple sets, there is much room for performance improvements.

Applying to influence maximization.

In order to solve influence maximization, we apply the greedy algorithm with $f = \hat{\mathsf{Inf}}(\cdot)$. It is easy to observe that $\hat{\mathsf{Inf}}(\cdot)$ is monotone and submodular, and thus, the greedy algorithm returns a $(1-\mathrm{e}^{-1})$-approximation to the optimal $\hat{\mathsf{Inf}}(\cdot)$.

Snapshot-based algorithms have the following advantages over naive estimation algorithms.

Algorithm 3.2 Snapshot-based algorithms for influence estimation and influence maximization.

```
1: procedure Preprocess($\mathcal{G} = (V, E, p)$, $r$)
2:     sample $r$ random graphs $G_1, \dots, G_r$ from $\mathcal{G}$.
3: procedure Estimate($S \subseteq V$)
4:     compute $\mathsf{r}_{G_1}(S), \dots, \mathsf{r}_{G_r}(S)$ by running BFSes.
5:     return $\frac{1}{r}\sum_{i\in[r]} \mathsf{r}_{G_i}(S)$.
6: procedure Maximize($k$)
7:     return a solution by greedy strategy with $f(\cdot) = \frac{1}{r}\sum_{i\in[r]} \mathsf{r}_{G_i}(\cdot)$.
```

- Running the greedy algorithm on $\hat{\mathsf{Inf}}$, we are able to obtain more influential vertices. For example, seed sets obtained with $r \approx 100$ random graphs are comparable to or even more influential than those obtained with 10,000 Monte-Carlo simulations [46, 106, 108].

- We are able to introduce acceleration techniques over fixed r random graphs as described below.

Existing techniques.

To the best of our knowledge, Kimura, Saito, and Nakano [106] were the first to use snapshot-based estimation. Their algorithm *Bond Percolation* introduces two techniques. One is to replace the random graphs with condensations. For each DAG G_i, let G'_i be its condensation DAG, $\pi_i : V(G_i) \to V(G'_i)$ be the correspondence mapping, and w_i be the vertex weights for G'_i defined in Section 2.1. Then, for any vertex set $S \subseteq V$,

$$\hat{\mathsf{Inf}}(S) = \frac{1}{r}\sum_{i\in[r]} \mathsf{r}_{G'_i,\mathsf{w}_i}(\pi_i(S)). \tag{3.2}$$

The other one is to shrink the DAGs over iterations. For a graph $G = (V, E)$, a vertex set $S \subseteq V$, a vertex $v \in V$, it follows that

$$\mathsf{r}_G(S \cup \{v\}) = \mathsf{r}_G(S) + \mathsf{r}_{G-\mathsf{R}_G(S)}(v). \tag{3.3}$$

Therefore, we can obtain $\mathsf{r}_G(S \cup \{v\})$ for all vertices v by first computing $\mathsf{r}_G(S)$ and then computing $\mathsf{r}_{G-\mathsf{R}_G(S)}(v)$ for all vertices v.

Chen, Wang, and Yang [40] proposed *NewGreedy*, which adopts a k-mins sketch [49] in order to speed-up the computation of the size of reachable sets.

Cheng, Shen, Huang, Zhang, and Cheng [46] proposed *StaticGreedy* and its variant *StaticGreedyDU*. StaticGreedy is essentially equivalent to Algorithm 3.2 without any optimization techniques. StaticGreedyDU stores the reachable sets for each vertex to efficiently compute the size of reachable sets. This technique achieved approximately ten times speed-up against StaticGreedy; however, storing all the sets into memory is severely prohibitive in the case of high influence probabilities [12, 156].

Cohen, Delling, Pajor, and Werneck [52] proposed *Sketch-based Influence Maximization (SKIM)*, which uses bottom-k min-hash sketch [50, 51] for approximate computation of the number of reachable vertices.

There have been also proposed different approaches such as graphics processing unit for parallel processing [128] and subgraph partitioning [131].

Bottleneck of the first iteration.

We conclude this subsection with the inefficiency issue of snapshot-based algorithms. In the first iteration, we compute $\hat{\mathsf{Inf}}(v)$ for all vertices v. This requires solving the following problem for each random graph $\{G_i\}_{i \in [r]}$.

Problem 3 (Descendant counting problem [27, 49])**.** *Given a directed graph $G = (V, E)$, the* descendant counting problem *asks to compute $\mathsf{r}_G(v)$ for every vertex $v \in V$.*

Borassi [28] has proven that the descendant counting problem is unsolvable in time $\mathcal{O}(|V|^{2-\epsilon})$ for any $\epsilon > 0$, unless the strong exponential time hypothesis is false. Most of the snapshot-based algorithms suffer from the difficulty of descendant counting; conducting BFSes starting from every vertex may require $\mathcal{O}(|V||E|)$ time. This is too expensive.

3.3 Reverse Influence Sampling

3.3.1 Concept

Borgs, Brautbar, Chayes, and Lucier [29] established *reverse influence sampling (RIS)*, the first near-linear time algorithm (for constant k). Roughly speaking, RIS builds sketches called reverse reachable sets in which influential vertices are frequently appearing and uses them to estimate the influence spread. Let us begin with the definition of *reverse reachable sets.*

Definition 3.1 (Reverse reachable set [29])**.** *For an influence graph $\mathcal{G} = (V, E, p)$ and a* target *vertex $z \in V$, a* reverse reachable (RR) set *for z (under the IC model) is defined as a random set R of vertices that can reach z in a random graph G sampled from $\mathcal{G}$. For an RR set for a target which is selected from V uniformly at random, we simply refer it an* RR set*.*

An important observation is the following:

Lemma 3.2 ([29, Observation 3.2])**.** *For an influence graph $\mathcal{G} = (V, E, p)$ and a vertex set $S \subseteq V$, S intersects an RR set with probability $\mathsf{Inf}_{\mathcal{G}}(S)/|V|$.*

We now explain how to use RR sets for influence estimation and influence maximization. Let $\mathcal{R}$ be a collection of RR sets. For a vertex set S, let $F_{\mathcal{R}}(S)$ denote the fraction of RR sets in $\mathcal{R}$ intersecting S, i.e.,

$$F_{\mathcal{R}}(S) = \frac{|\{R \in \mathcal{R} \mid R \cap S \neq \emptyset\}|}{|\mathcal{R}|}. \tag{3.4}$$

Then, due to [29, Observation 3.2], $|V| \cdot F_{\mathcal{R}}(S)$ is an unbiased estimator of $\mathsf{Inf}(S)$, i.e.,

$$\mathop{\mathbf{E}}_{\mathcal{R}}[|V| \cdot F_{\mathcal{R}}(S)] = \mathsf{Inf}(S). \tag{3.5}$$

Therefore, in RIS-based algorithms, influence estimation turns into the computation of $F_{\mathcal{R}}(\cdot)$ and influence maximization turns into the maximization of $F_{\mathcal{R}}(\cdot)$.

Algorithm 3.3 Reverse influence sampling for influence maximization [29].

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a diffusion model $\mathcal{M}$, a seed size $k$.
 1: $\mathcal{R} \leftarrow \emptyset$.
 2: $i \leftarrow 1$.
 3: repeat
 4:     $z_i \leftarrow$ a vertex chosen from $V$ uniformly at random.
 5:     $R_i \leftarrow$ a random RR set for $z$ under $\mathcal{M}$.
 6:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_i\}$.
 7:     $i \leftarrow i + 1$.
 8: until $\mathcal{R}$ includes a sufficient number of vertex sets
 9: $S \leftarrow \emptyset$.
10: for $\ell = 1$ to $k$ do
11:     $s_\ell \leftarrow \mathrm{argmax}_{v \in V} F_{\mathcal{R}}(v)$.        ▷ $F_{\mathcal{R}}(v)$ is defined as $\frac{|\{R \in \mathcal{R} | R \cap \{v\} \neq \emptyset\}|}{|\mathcal{R}|}$.
12:     $S_\ell \leftarrow S_{\ell-1} \cup \{s_\ell\}$.
13:     remove RR sets including $v_\ell$ from $\mathcal{R}$.
14: return $S_k$.
```

3.3.2 Applying to Influence Estimation

If we are aware of influence estimation, a simple application of Hoeffding's inequality gives the required number of RR sets.

Theorem 3.3 (Additive error of influence estimation by RR sets)**.** *Assume that we generated a collection $\mathcal{R}$ of θ RR sets. For a vertex set $S \subseteq V$, let $\hat{\mathsf{Inf}}(S) := |V| \cdot F_{\mathcal{R}}$. Then, for any $\epsilon > 0$,*

$$\mathbf{Pr}\left[|\hat{\mathsf{Inf}}(S) - \mathsf{Inf}(S)| \geq |V|\epsilon\right] \leq 2\exp(2\theta\epsilon^2). \tag{3.6}$$

Proof. Let $\mathcal{R} = \{R_1, \ldots, R_\theta\}$. Let X_i be a random variable that takes 1 if $S \cap R_i \neq \emptyset$ and 0 otherwise, and let $\bar{X} = \frac{1}{\theta}\sum_{i \in [\theta]} X_i$. Note that $\mathsf{Inf}(S) = |V|\,\mathbf{E}[\bar{X}]$ and $\hat{\mathsf{Inf}}(S) = |V|\bar{X}$. Since each X_i is bounded in the interval $[0, 1]$, applying Hoeffding's inequality [87] yields

$$\begin{aligned}\mathbf{Pr}\left[|\hat{\mathsf{Inf}}(S) - \mathsf{Inf}(S)| \geq |V|\epsilon\right] &= \mathbf{Pr}\left[|\bar{X} - \mathbf{E}[\bar{X}]| \geq \epsilon\right] \\ &\leq 2\exp\left(\frac{2\theta^2\epsilon^2}{\sum_{i\in[\theta]}(1-0)^2}\right) \\ &= 2\exp\left(2\theta\epsilon^2\right). \end{aligned} \tag{3.7}$$

□

3.3.3 Applying to Influence Maximization

Now, we describe RIS-based influence maximization. Algorithm 3.3 shows pseudocode of the framework of RIS. Given an influence graph $\mathcal{G} = (V, E, p)$, a diffusion model $\mathcal{M}$ (e.g., the IC model), and a seed size k, it performs the following two stages. In the first stage, beginning with an empty collection $\mathcal{R} = \emptyset$, it iteratively generates an RR set and adds it to $\mathcal{R}$. The above repetition continues until $\mathcal{R}$ includes a "sufficiently large" number of RR sets. In the second stage, it computes an approximate solution S_k for the *maximum coverage problem*, which requires selecting a set of k vertices from V that intersects the maximum number of RR sets in $\mathcal{R}$, i.e., $\mathrm{argmax}_{S \in \binom{V}{k}} F_{\mathcal{R}}(S)$, by the greedy algorithm. Finally, it returns the obtained solution S_k. Since $F_{\mathcal{R}}(\cdot)$ is monotone and submodular, it turns out that $F_{\mathcal{R}}(S_k) \geq (1 - \mathrm{e}^{-1}) \max_{S \in \binom{V}{k}} F_{\mathcal{R}}(S)$ Note that the second stage completes in linear time in the size of $\mathcal{R}$ [29], the first stage dominates the total running time.

3.3.4 Stopping Conditions of RR Set Generation

The central problem of RIS-based algorithms is the time at which we stop RR set generation. Intuitively, $|V| \cdot F_{\mathcal{R}}$ should be close to Inf. Borgs, Brautbar, Chayes, and Lucier [30] first prove a stopping condition with a theoretical guarantee on the accuracy.

Theorem 3.4 ([30, Theorem 3.1]). *For a parameter $\epsilon \in (0, 1/2)$, assume that Algorithm 3.3 terminates when the number of edges touched so far in the process of RR set generation exceeds $W = ck\epsilon^{-2}|E| \log |V|$, where $c = 4(1+\epsilon)(1+\frac{1}{k}) \leq 12$. Then, the seed set of size k that Algorithm 3.3 returns is a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution with probability at least $3/5$. Moreover, the running time is $\Theta(W)$.*

Notice here that a factor $k \log |V|$ is obtained by taking a union bound over all sets of size at most k; there are at most $|V|^k$ sets. (In the conference version [29] presented at SODA 2014, the authors claimed that $\Theta(\epsilon^{-3}(|V| + |E|) \log |V|)$ is sufficient, but this claim was fixed later.)

Albeit the near linear complexity, a hidden constant sorely limits the practical efficiency against large-scale networks. On the other hand, in practice, the number of RR sets required for high-quality influence maximization is much smaller than that envisioned from the theoretical bound. Hence, there have been developed strategies for bounding the required number of RR sets more tightly.

Most of the studies [29, 30, 57, 92, 147, 148, 149, 152, 176, 177] considered the following requirement:

> Given parameters ϵ and δ in addition to an influence graph $\mathcal{G} = (V, E, p)$ and a seed size k, the algorithm is required returning a collection of RR sets $\mathcal{R}$ such that the greedy strategy on $F_{\mathcal{R}}$ yields a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution against the optimum with probability at least $1 - \delta$.

Hereafter, we review existing approaches.

Bounding the optimum influence.

The first approach is to use the optimum influence. Tang, Xiao, and Shi [176] have shown an upper bound of the required number of RR sets as

$$\theta_{\mathrm{TIM}} = (8 + 2\epsilon)|V| \frac{\ln(2/\delta) + \ln \binom{|V|}{k}}{\epsilon^2 \mathsf{OPT}_k}, \tag{3.8}$$

where $\mathsf{OPT}_k = \max_{S \in \binom{V}{k}} \mathsf{Inf}(S)$. Since the actual value of OPT_k is unknown beforehand, we need to estimate its lower bound. Tang, Xiao, and Shi [176] proposed *Two-phase Influence Maximization (TIM)*. TIM consists of the two phases. The first phase estimates a lower bound of OPT_k and θ_{TIM}, and the second phase generates θ_{TIM} RR sets and then executes the greedy algorithm on them. Specifically, $\binom{|V|}{k}^{-1} \sum_{S \in \binom{V}{k}} \mathsf{Inf}(S)$ is used as a lower bound of OPT_k. Also, TIM$^+$ was proposed as an improvement upon TIM that incorporates a heuristic method. However, these lower bounds can be terribly small in the worst case, which leads to an excessive number of RR set generation.

Influence Maximization via Martingales (IMM) proposed by Tang, Shi, and Xiao [177] introduced a martingale approach, where we reuse RR sets in both lower-bound estimation of OPT_k and the greedy algorithm. The authors proposed a statistical test to decide "$\mathsf{OPT}_k > x$?". Then, IMM calls the test for $x = |V|, |V|/2, |V|/4, \ldots$.

Degree-based thresholding.

Since it is still hard to estimate OPT_k, some strategies decide the stopping point *without* estimating OPT_k. *Linear-time Influence Spectrum Algorithm (LISA)* in Dinh, Nguyen, Ghosh, and Mayo [57] and Nguyen, Ghosh, Mayo, and Dinh [149] continues RR set generation until the maximum of $|\mathcal{R}| \cdot F_{\mathcal{R}}(v)$ among all vertices v exceeds a threshold value $1 + 4.6\epsilon^{-2} \ln \frac{2}{\delta}$.

BCT in Nguyen, Dinh, and Thai [147] and Nguyen, Thai, and Dinh [152] adopts a slightly different approach. For a given collection $\mathcal{R}$ of RR sets and a k-vertex set S_k obtained by running the greedy algorithm on $F_{\mathcal{R}}$, BCT verifies whether $|\mathcal{R}| \cdot F_{\mathcal{R}}(S_k)$ is greater than a threshold value

$$\psi_{\mathrm{BCT}} \approx 7.4\left[\ln \frac{1}{\delta} + \ln \binom{|V|}{k} + \frac{2}{|V|}\right]. \tag{3.9}$$

If this is the case, then it stops RR set generation and returns S_k. Otherwise, it doubles the number of RR sets in $\mathcal{R}$.

Search-and-verify approach.

The last approach attempts to find an approximate solution and verify its quality forthwith. *Stop-and-Stare Algorithm (SSA)* of Nguyen, Thai, and Dinh [148] directly verifies the influence spread of candidate solutions in a statistical manner. SSA first applies the greedy strategy on a current collection $\mathcal{R}$ of RR sets to obtain a seed set S_k. It then generates a *separate* collection $\mathcal{R}'$ of RR sets to test "$F_{\mathcal{R}}(S_k) \leq (1 + \epsilon')F_{\mathcal{R}'}(S_k)$?" for some ϵ', which ensures that $|V| \cdot F_{\mathcal{R}}(S)$ is sufficiently close to $\mathsf{Inf}(S)$. If the verification was rejected, then it doubles the number of RR sets in $\mathcal{R}$. *Dynamic Stop-and-Stare Algorithm (D-SSA)* automatically adjusts internal parameters of SSA.

Li, Smith, Dinh, and Thai [123] used integer programming. *Tiny Integer Program with Theoretically OPtimal (TipTop)* algorithm generates a small number of RR sets and solves the corresponding maximum coverage problem using an integer programming solver. Then, it generates a relatively large number of RR sets separately to verify whether the obtained solution is sufficiently influential in a similar manner to [148].

We here note that these techniques are severely affected by the network structure and edge probability settings. In fact, most of the studies [57, 92, 148, 176, 177] have tested each proposed method under only one specific setting (in-degree weighted in Chapter 4).

3.3.5 RR Set Generation under the IC Model

Algorithm 3.4 shows pseudocode of RR set generation under the IC model. This is used as a subroutine in RIS-based algorithms (Algorithm 3.3, line 5). We define an *activation function* as $x : E \to [0, 1]$, each of which is sampled from

Algorithm 3.4 RR set generation under the IC model.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a target vertex $z \in V$
1: $Q \leftarrow$ a queue containing $z$.
2: while $Q \neq \emptyset$ do
3:     remove $v$ from $Q$.
4:     for all $u \in \mathcal{N}^-_{\mathcal{G}}(v)$ do
5:         determine the value $x(u,v)$ in the interval $[0,1]$ randomly.
6:         if $(u,v)$ is live w.r.t. $x$ and $u$ is not inserted into $Q$ so far then
7:             insert $u$ onto $Q$.
8: return the set of vertices that have been inserted into $Q$ so far.
```

$[0,1]$ uniformly at random. An edge (u,v) is called *live* with respect to x if $x(u,v) < p(u,v)$ and *blocked* otherwise, i.e., (u,v) is live with probability $p(u,v)$. At the beginning, we sample a *target* vertex z from V uniformly at random and prepare a queue consisting of z. Then, we iteratively determine vertices that would influence z via the following BFS-like procedure. For each iteration, we remove a vertex v from the queue. For each in-neighbor u of v, we determine the value of $x(u,v)$. If (u,v) is live and u has not been inserted into the queue so far, then we insert u into the queue. The procedure continues until the queue is empty. We finally return the set of vertices that have been inserted into the queue. Note that the expected number of edges touched is bounded by $\frac{|E|}{|V|} \max_v \mathsf{Inf}_{\mathcal{G}}(v)$ [30].

3.4 Heuristics

Heuristic algorithms perform neither direct Monte-Carlo simulations nor RR set generations. Abstractly, these algorithms assume ad-hoc conditions, that make easy to estimate the influence spread. However, the conditions are hard to hold in general, which results in the expense of the quality of solutions.

3.4.1 Restricting the Range of Influence

One approach is to make an assumption that the spread of influence stays inside a specific type of local region, e.g., shortest paths [105], neighbors [40, 94], communities [45, 122, 184], and maximum influence paths [41, 42, 104, 182] so as to make it easier to estimate the influence spread.

Shortest paths.

Kimura and Saito [105] proposed the *Shortest-Path Model (SPM)* under which influence diffuses along with shortest paths and its relaxation *SP1 Model* further allows paths whose length is one more than the shortest. The authors provided an efficient way to compute the influence spread under both assumptions.

Neighbors.

Chen, Wang, and Yang [40] assumed that influence from a seed cannot spread outside the seed's neighbors. Thus the influence spread of a seed equals the total influence probabilities of edges leaving the seed. The proposed method, called *DegreeDiscount*, exploits the above fact and runs in near-linear time. *Simulated Annealing with Effective Diffusion Values (SAEDV)* proposed by Jiang, Song, Cong, Wang, Si, and Xie [94] also adopts a similar assumption to DegreeDiscount and uses simulated annealing instead of the greedy strategy.

Communities.

Wang, Cong, Song, and Xie [184] assume that influence stays inside small subgraphs called communities, where vertices of a community are connected with more vertices inside the same community than others. *Community-based Greedy Algorithm (CGA)* first divides a given influence graph into communities, then solves influence maximization on each community, and finally puts the solutions together via a dynamic programming.

Maximum influence paths.

Prefix excluding Maximum Influence Arborescence (PMIA) algorithm proposed by Chen, Wang, and Wang [41] assumes that influence diffuses along with a tree so as to compute the influence spread in linear time. The trees are constructed so that a path from a vertex s to a vertex t is the most frequently appearing in random graphs among all possible paths.

3.4.2 Linear System Approximation

The next approach assumes that the influence spread of a vertex can be expressed as a linear system, though it is not the case in reality. Yang, Chen, Liu, Xiang, Xu, and Shad [189] used a linear system to approximate the activation probability of each vertex and proposed an iterative algorithm to solve the linear system. *Influence Rank Influence Estimation (IRIE)* by Jung, Heo, and Chen [96] expresses the marginal influence of each vertex over a certain set of vertices as simultaneous linear equations. *IMRank* by Cheng, Shen, Huang, Chen, and Cheng [47] uses IRIE as a subroutine for influence spread estimation.

3.4.3 Graph Reduction

A few studies attempted to reduce an input influence graph in advance and then apply any existing algorithms on the resulting smaller influence graph.

Mathioudakis, Bonchi, Castillo, Gionis, and Ukkonen [138] proposed *Sparsification of Influence Networks (Spine)* algorithm, which eliminates a specified number of edges from an input graph so as to maximize the likelihood of reproducing a given log of user actions. Purohit, Prakash, Kang, Zhang, and Subrahmanian [161] proposed *coarseNet* algorithm, which contracts unimportant edges in an input graph that are identified based on the spectrum of the adjacency matrix.

3.4.4 Others Strategies

There have been other strategies such as Inclusion-Exclusion theorem [190], simple path enumeration [81], path counting [67], and belief propagation [153].

Chapter 4

Analysis of the Trends of Diffusive Behaviors

In this chapter, we analyze the diffusive behaviors of the IC model using the *configuration* of network data and influence probability settings. The objective is to answer the following questions.

- **Q1.** Why, how, and to what extent do diffusive properties differ among the configurations?
- **Q2.** Under which configuration does a specific category of influence maximization algorithms require a long time or consume a large space?
- **Q3.** Under which configuration can we cut off redundant computations that naive algorithms induce?

For this purpose, we investigate structures of two concepts below using seven influence probability settings and eighteen real-world networks.

1. Reachable sets on random graphs, whose computation is a key step in snapshot-based algorithms (Section 4.3).
2. RR sets, whose generation is a key step in RIS-based algorithms (Section 4.4).

4.1 Strategies of Influence Probability Assignment

This section defines several strategies of influence probability assignment.

- **Uniform** ($\textsc{uc}_x$): each edge has a constant influence probability x. The value of x takes either 0.1 or 0.01. This setting was firstly proposed by Kempe, Kleinberg, and Tardos [98].
- **Exponential** ($\textsc{exp}_x$): each influence probability is chosen independently from the exponential distribution with mean x. Our work [158] presented in SIGMOD 2017 adopted this setting motivated by empirical evidence of influence probabilities [21, 56]. The mean x takes either 0.1 or 0.01.
- **Trivalency** ($\textsc{tri}$): each influence probability is chosen randomly from a set $\{0.1, 0.01, 0.001\}$, which corresponds to high, medium, and low influences. This setting was first proposed to be used by Chen, Wang, and Wang [41] so as to make influence probabilities *non-uniform*.
- **In-degree weighted cascade** ($\textsc{iwc}$): the influence probability of edge (u, v) is set to $1/|\mathcal{N}^{-}(v)|$. Thus, for each vertex v, $\sum_{u \in \mathcal{N}^{-}(v)} p(u, v) = 1$. Kempe, Kleinberg, and Tardos [98] have incorporated this setting into experimental evaluation as a mimic of the LT model.

Table 4.1: Datasets examined in Chapter 4. (d) and (u) denote "directed" and "undirected," respectively. All networks were downloaded from SNAP [116].

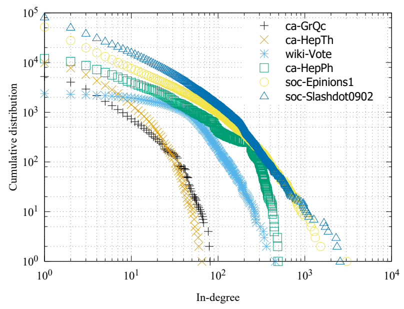
name	**type**	$\lvert V\rvert$	$\lvert E\rvert$
ca-GrQc	collaboration(u)	5,242	28,968
ca-HepTh	collaboration(u)	9,877	51,946
wiki-Vote	social(d)	7,115	103,689
ca-HepPh	collaboration(u)	12,008	236,978
soc-Epinions1	social(d)	75,879	508,837
soc-Slashdot0922	social(d)	82,168	870,161
web-NotreDame	web(d)	325,729	1,469,679
ego-Twitter	social(d)	81,306	1,768,135
loc-Gowalla	social(u)	196,591	1,900,654
web-Stanford	web(d)	281,903	2,312,497
wiki-Talk	social(d)	2,394,385	5,021,410
web-Google	web(d)	875,713	5,105,039
com-Youtube	social(u)	1,134,890	5,975,248
web-BerkStan	web(d)	685,230	7,600,595
higgs-twitter	social(d)	456,626	14,855,819
soc-Pokec	social(d)	1,632,803	30,622,564
soc-LiveJournal1	social(d)	4,847,571	68,475,391
com-Orkut	social(u)	3,072,441	234,370,166

- **Out-degree weighted cascade** (OWC): the influence probability of edge (u, v) is set to $1/|\mathcal{N}^+(v)|$. Thus, for each vertex u, $\sum_{v\in\mathcal{N}^+(u)} p(u, v) = 1$. In other words, each vertex activates just one neighbor in expectation. Our work [155] presented in WWW 2017 have adopted this setting so that each vertex has a uniform influence.

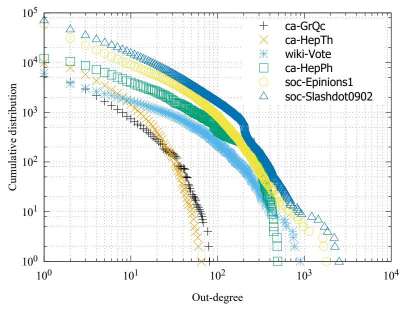
We use the *degree-weighted setting* as a general term for IWC and OWC, where influence probabilities are determined based on degrees. We use the *unweighted setting* as a general term for UC_x, EXP_x, and TRI, where influence probabilities are unweighted and independent each other. Intuitively, the degree-weighted setting equalizes vertices' impact on the neighbors. Hence, one can expect that influence does not diffuse widely. On the other hand, under the unweighted setting, vertices of a high degree have a high chance to influence neighbors. Hence, what to expect is that influence diffusion triggered by high-degree vertices reaches a significant portion of the network.

4.2 Network Data

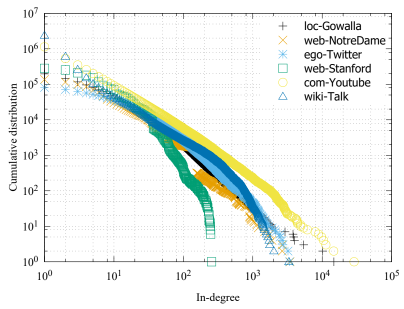
We here describe network data examined in this chapter. We use eighteen networks, which were downloaded from Stanford Large Network Dataset Collection in Stanford Network Analysis Project (denoted SNAP) [116], which is maintained by Jure Leskovec. Table 4.1 summarizes the basic statistics of each network. We have eleven social networks, three collaboration networks, and four web graphs. The number of vertices varies from five thousand to three million and the number of edges varies from 30 thousand to 200 million. Detailed descriptions are as follows.



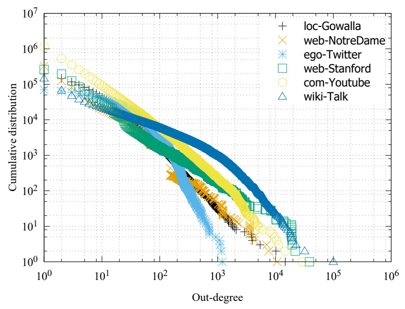
(a) In-degree distribution of smaller six networks



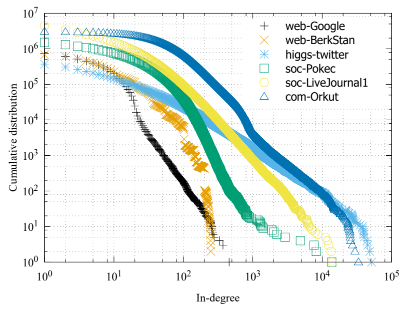
(b) Out-degree distribution of smaller six networks



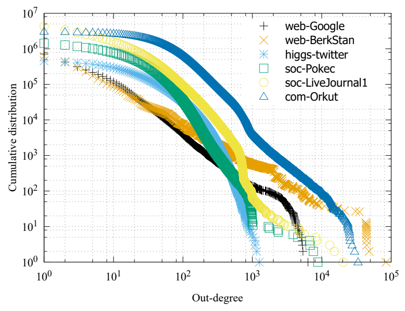
(c) In-degree distribution of medium-sized six networks



(d) Out-degree distribution of medium-sized six networks



(e) In-degree distribution of larger six networks



(f) Out-degree distribution of larger six networks

Figure 4.1: Cumulative distribution of in-degree and out-degree of each network.

4.2.1 Detailed Description

Social networks.

In *social networks*, each vertex corresponds to a user of a social networking service, and each edge represents social interactions or friendships between the users. Note that social networks can be directed or undirected.

- wiki-Vote: This is a who-votes-on-whom network extracted from a free encyclopedia Wikipedia (wikipedia.org). Each directed edge (u, v) means that the user associated with v voted on the user associated with u.
- soc-Epinions1: This is a who-trust-whom network of a consumer review site Epinions.com (epinions.com).
- soc-Slashdot0922: This is an online social network of a news website Slashdot (slashdot.org) collected in February 2009. This graph contains friend or foe links between the users.

- ego-Twitter: This is a social network extracted from Twitter (twitter.com). Twitter users can create a list of users, and this network contains an undirected edge (u, v) if users u and v belong to the same list.
- loc-Gowalla: This is a friendship network of a location-based SNS Gowalla collected in the period February 2009 to October 2010.
- com-Youtube: This is a social network of a video-sharing website YouTube (www.youtube.com).
- wiki-Talk: This is a social network extracted from a free online encyclopedia Wikipedia (www.wikipedia.org) during the beginning of Wikipedia to January 2008. Each user has a talk page, and edge (u, v) means that user u has edited the talk page of user v.
- higgs-twitter: This is a social network of an SNS Twitter (twitter.com). This network consists of users who mentioned the discovery of a new particle with the features of the Higgs boson in the period 1st July 2012 to 7th July 2012.
- soc-Pokec: This is an online social network of an SNS Pokec (pokec.azet.sk) in Slovakia. Each friendship relation is directed.
- soc-LiveJournal1: This is an online social network of an SNS LiveJournal (livejournal.com).
- com-Orkut: This is an online social network of an SNS Orkut (www.orkut.com).

Collaboration networks.

In *collaboration networks*, each vertex corresponds to an author and if an author has a co-authored paper with another author, then an "undirected" edge connects them. Note that if k authors have co-authored with the same paper, then it yields $\binom{k}{2}$ edges.

- ca-GrQc: This is a collaboration network extracted from the General Relativity and Quantum Cosmology category in the e-print arXiv (arxiv.org) in the period January 1993 to April 2003.
- ca-HepTh: This is a collaboration network extracted from the High Energy Physics - Theory category in the e-print arXiv (arxiv.org) in the period January 1993 to April 2003.
- ca-HepPh: This is a collaboration network extracted from the High Energy Physics - Phenomenology category in the e-print arXiv (arxiv.org) in the period January 1993 to April 2003.

Web graphs.

In *web graphs*, vertices and edges correspond to web pages and hyperlinks connecting them, respectively. Note that web graphs are directed.

- web-NotreDame: This is a web graph of the University of Notre Dame domain (nd.edu) collected in 1999.

- web-Stanford: This is a web graph extracted from the Stanford University domain (`stanford.edu`) collected in 2002.
- web-Google: This is a web graph released by Google in 2002.
- web-BerkStan: This is a web graph extracted from the University of California, Berkeley domain (`berkely.edu`) and Stanford University domain (`stanford.edu`) collected in 2002.

4.2.2 Preprocessing

Since we are aware of the diffusion process, we applied the following preprocessing for each network. We first removed self-loops, multi-edges, and isolated vertices from each network, which does not affect the diffusion process. Note, therefore, that the numbers of vertices and edges in some network differ from those provided by the source websites. If the graph is undirected, we replace it with its directed version. Further, all edges in the web graphs were reversed because it is natural to assume that information follows hyperlinks in the opposite direction.

4.2.3 Structural Properties of Complex Networks

Networks introduced in this section are called *complex networks* and have common structural properties. These properties of complex networks have been studied in graph mining and network science communities. In this subsection, we briefly review popular properties of complex networks.

Power-law degree distribution: The degree distribution follows a power-law [19, 62], that is, the fraction of vertices of degree k is proportional to $k^{-\gamma}$, where $2 < \gamma < 3$ typically. Figure 4.1 shows the cumulative degree distribution of each network.

Short average shortest-path length: The average length of shortest-paths over all possible pairs of vertices is small. It is known by the name of "six degrees of separation" phenomenon [139, 180].

Large clustering coefficient: The global clustering coefficient is defined as the number of triplets of vertices which are connected by exactly three edges divided by the number of triplets of vertices which are connected at least two edges. Complex networks show large global clustering coefficients [185], which intuitively says that "friends of a friend are likely to be friends" [88].

Core-fringe structure: Complex networks can be generally decomposed into two parts [32, 120, 135]; the *core*, which is well connected and contains many edges, and the *fringe*, which looks like trees and contains few edges.

4.3 Analysis of Reachable Sets

We first focus on the reachable sets on random graphs, computing whose size is a key step in snapshot-based algorithms. The goal is to discover the configurations for which reachable sets are large and hence snapshot-based algorithms become less efficient. To that end, we sampled a random graph G from an influence graph $\mathcal{G} = (V, E, p)$ randomly, and we compute $\mathsf{r}_G(v)$ for each vertex v in V, i.e., solve descendant counting for G. In other words, we computed the cascade size for each vertex given a fixed outcome of the activation trials.

4.3.1 Average Size

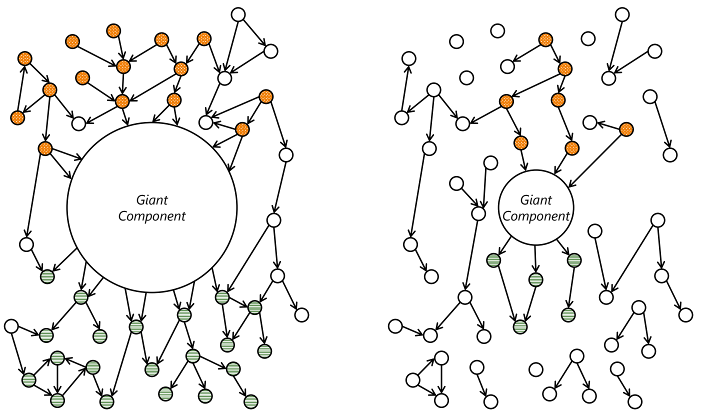
Let us begin with the average trends. Table 4.2 reports the average size of reachable sets, denoted $\bar{\mathsf{r}} = \frac{1}{|V|}\sum_{v \in V} \mathsf{r}(v)$, for each configuration.

As can be seen, there are remarkable differences in the average size among the "probability settings." Consider soc-Slashdot0922 network as an example, $\bar{\mathsf{r}}$ dominates only a small fraction of the graph under $\textsc{uc}_{0.01}$ ($\bar{\mathsf{r}} = 4.2$, 0.0051% of $|V|$) and $\textsc{exp}_{0.01}$ ($\bar{\mathsf{r}} = 7.2$, 0.0088% of $|V|$) while a large portion under $\textsc{uc}_{0.1}$ ($\bar{\mathsf{r}} =$ 6,540.3, 8.0% of $|V|$) and $\textsc{exp}_{0.1}$ ($\bar{\mathsf{r}} =$ 6,483.5, 7.9% of $|V|$). In general, the largest was obtained under both $\textsc{uc}_{0.1}$ and $\textsc{exp}_{0.1}$, followed in order by $\textsc{tri}$, $\textsc{uc}_{0.01}$ and $\textsc{exp}_{0.01}$, which coincides with monotonic behaviors of the activation process with regard to the influence probability. Note also that $\textsc{uc}_x$ and $\textsc{exp}_x$ lead to almost equivalent results, and thus we often omit the results for either $\textsc{uc}_x$ or $\textsc{exp}_x$. This is not surprising as they exhibit the same local process of activation trials. Under degree-weighted settings, $\bar{\mathsf{r}}$ is small (< 100) at all times.

In addition, the "network structure" has a significant impact on the average size. For example, the largest network com-Orkut shows the highest $\bar{\mathsf{r}}$ for every probability assignment. However, larger graphs do not necessarily result in larger $\bar{\mathsf{r}}$, e.g., those of web graphs including web-NotreDame, web-Stanford, web-Google, and web-BerkStan are less than 500 even when using $\textsc{uc}_{0.1}$ and $\textsc{exp}_{0.1}$.

Table 4.2: Average size of reachable sets for each configuration.

graph	$\lvert V\rvert$	$\text{UC}_{0.1}$	$\text{UC}_{0.01}$	$\text{EXP}_{0.1}$	$\text{EXP}_{0.01}$	TRI	IWC	OWC
ca-GrQc	5,242	9.3	1.1	11.1	1.1	1.8	4.0	4.2
ca-HepTh	9,877	45.5	1.1	68.6	1.1	1.4	4.2	4.3
wiki-Vote	7,115	381.8	1.2	386.8	1.3	57.0	1.7	3.6
ca-HepPh	12,008	1,274.4	18.4	1,319.8	21.5	238.5	7.4	7.7
soc-Epinions1	75,879	1,818.8	2.5	1,830.6	2.7	310.8	5.0	12.7
soc-Slashdot0922	82,168	6,540.3	4.2	6,483.5	7.2	1,357.4	20.7	8.9
web-NotreDame	325,729	29.3	1.4	27.3	1.4	5.2	2.3	6.0
ego-Twitter	81,306	17,018.3	5.0	16,925.5	2.7	3,134.7	11.1	7.8
loc-Gowalla	196,591	14,137.0	23.2	14,146.4	23.4	1,740.1	10.5	8.6
web-Stanford	281,903	153.5	1.1	135.3	1.1	2.4	4.8	4.5
wiki-Talk	2,394,385	4,185.5	36.1	4,258.5	37.1	737.7	13.0	1.2
web-Google	875,713	13.6	1.1	11.6	1.1	1.4	4.0	3.9
com-Youtube	1,134,890	25,820.5	60.5	25,612.7	63.7	3,396.9	5.9	7.0
web-BerkStan	685,230	371.4	1.4	429.0	1.3	12.0	5.0	4.4
higgs-twitter	456,626	104,515.0	523.6	105,157.6	483.2	30,722.2	11.5	12.6
soc-Pokec	1,632,803	369,986.2	1.6	370,236.8	1.5	66,168.5	16.8	14.5
soc-LiveJournal1	4,847,571	587,481.7	405.5	587,711.4	386.6	86,734.9	12.0	10.0
com-Orkut	3,072,441	2,359,514.1	137,098.7	2,360,092.6	137,890.3	1,414,539.3	37.0	50.1



(a) Random graph with high influence probability (b) Random graph with small influence probability

Figure 4.2: These figures explain the mechanism of getting bimodal distributions or decreasing distributions of reachable sets. Orange vertices can reach the giant component (GC) and green vertices are reachable from the GC. When the GC exists, vertices that can reach the GC have large reachable sets (left figure). When the GC is small or does not exists, no vertices have huge reachable sets (right figure). Note that white vertices have seldom large reachable sets.

4.3.2 Size Distribution of Reachable Sets

Now, let us look the size distribution of reachable sets. Figures 4.3–4.8 illustrate the size distribution of reachable sets $\{\mathsf{r}(v)\}_{v \in V}$ in a logarithmic scale. Black arrows point to the average $\bar{\mathsf{r}}$. See Appendix A for the complete experimental results. At a first glance, there are two extreme types of distribution as below.

1. A *bimodal* distribution, where the left mode is decreasing and the right mode dominates the total amount, which is likely to result in a large average size.
2. A *decreasing* distribution, where the frequency decreases as the size increases, which results in a small average size.

Bimodal distributions.

We then reveal the mechanism of getting bimodal distributions. Remark that we obtain bimodal ones only if we adopt unweighted settings. Under unweighted settings, each influence probability is drawn from the same distribution; each vertex's degree is expected to be multiplied by the average probability. Recall also that the core part contains a vast amount of high-degree vertices. Therefore, a high-degree connected component inside the core remains to be (strongly) connected, while the fringe part can be easily broken. This is not the case with the degree-weighted setting which equalizes the structural difference between the core and the fringe. Therefore, if influence probabilities are adequately high, a major part of the core remains strongly connected (Figure 4.2a). We call the largest SCC in random graphs the *giant component (GC)*. In this case, we have three trends of reachable sets $\mathsf{R}(v)$.

- **v can reach the GC** (orange vertices in Figure 4.2a). Then, $\mathsf{R}(v)$ contains the reachable set of the GC, which is typically extremely large.

- v **is reachable from the GC** (green vertices in Figure 4.2a). Then, $\mathsf{R}(v)$ may be small.
- v **cannot reach the GC and** v **is not reachable from the GC.** (white vertices in Figure 4.2a). Then, $\mathsf{R}(v)$ is small generally.

On the other hand, if influence probabilities are quite small, as shown in Figure 4.2b, the core gets almost completely disconnected, so we do not obtain a bimodal distribution.

The following observation summarizes the above discussion.

Observation 4.1 (Mechanism of getting bimodal distributions.)**.** *The size distribution of the reachable sets is bimodal only if we use unweighted settings and the GC is present in a random graph. Then, the average is dominated by "the fraction of vertices that can reach the GC" times "the size of the GC." Moreover, the latter term decreases as the influence probability decreases.*

From another perspective, BFSes from a lot of vertices visit the GC's reachable sets repeatedly. This is pretty redundant. In Chapter 5, we exploit this and devise a simple but effective algorithm for descendant counting.

Remark from [54]. It is known that a random cascade starting from a fixed vertex is either small or large [54], called the *phase-transition phenomenon*. Cui, Yang, and Homan [54] explained this with the concept of *giant propagation component* in which once any vertex of the component has been activated, most of the remaining vertices in it will be eventually activated with a high probability. If no vertices of the component become active, then the cascade quickly ends with small size. The authors [54] also pointed out that the phase-transition phenomenon does not occur in real-world social networks and introduced temporal and spatial factors into the diffusion model in order to reproduce the actual cascade, which follows a power-law distribution [119], accurately. Notice that we investigate the size distribution of diffusion starting from all vertices under a fixed result of activation trials, which is of interest from a computational point of view.

Decreasing distributions.

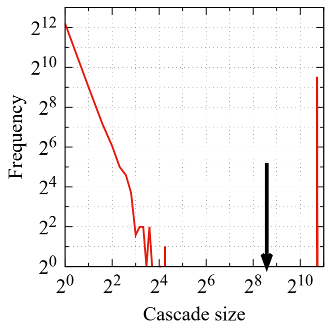
We next examine the case where we obtain decreasing distributions. For unweighted settings, we obtain a decreasing distribution on ca-GrQc ($\textsc{uc}_{0.01}$), ca-HepTh ($\textsc{uc}_{0.01}$ and $\textsc{tri}$), wiki-Vote ($\textsc{uc}_{0.01}$), and web-Google ($\textsc{uc}_{0.1}$, $\textsc{uc}_{0.01}$, $\textsc{tri}$). This is the case when the GC is sufficiently small, and so $\bar{\mathsf{r}}$ is small (Figure 4.2b). On the other hand, for degree-weighted settings, size distributions are always decreasing. We here give an intuition behind the mechanism of the degree-weighted settings. Recall that the sum of influence probabilities over the edges leaving or entering a vertex is one, i.e., any vertex has just one in- or out-neighbor in expectation. The resulting graph looks like a "tree" and it prevents most of the vertices from having a large number of reachable vertices.

The following gives a short summary.

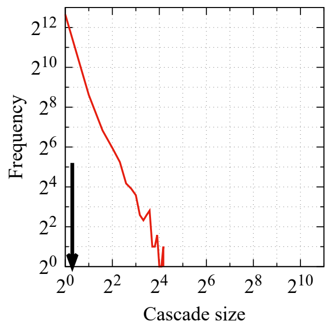
Observation 4.2 (Mechanism of getting decreasing distributions.)**.** *The size distribution of reachable sets is decreasing if either (1) the GC is not present in random graphs under unweighted settings, or (2) degree-weighted settings are used. Moreover, it this is the case, the average size of reachable sets is relatively small.*

Comparison between IWC and OWC.

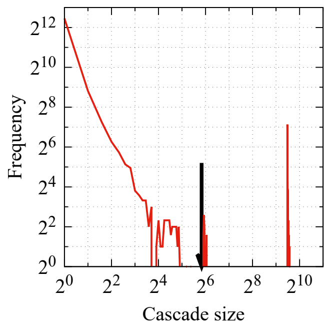
We observe a slightly different type of distribution between IWC and OWC. When using IWC, a small number of vertices have large reachable sets. Meanwhile, we do not observe such vertices having large $\mathsf{R}(\cdot)$ using OWC. This is because the vertices under OWC are expected to have just one out-neighbor while vertices may have two or more out-neighbors in expectation under IWC.



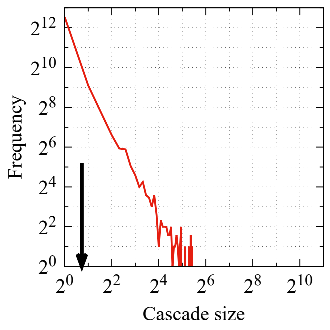
(a) wiki-Vote (UC$_{0.1}$)



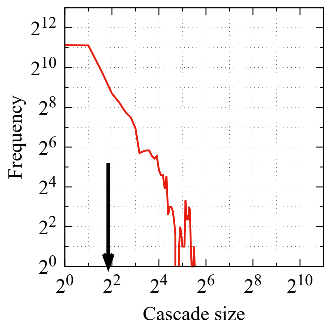
(b) wiki-Vote (UC$_{0.01}$)



(c) wiki-Vote (TRI)

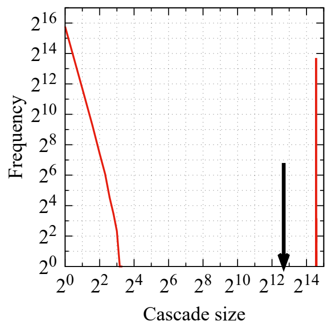


(d) wiki-Vote (IWC)

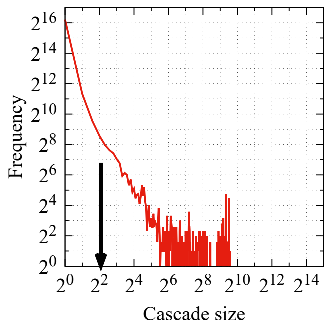


(e) wiki-Vote (OWC)

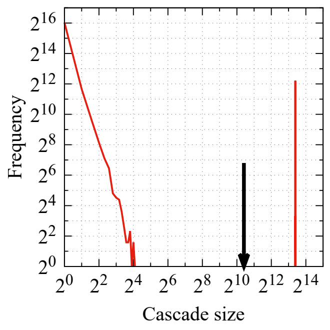
Figure 4.3: Size distribution of reachable sets in wiki-Vote network.



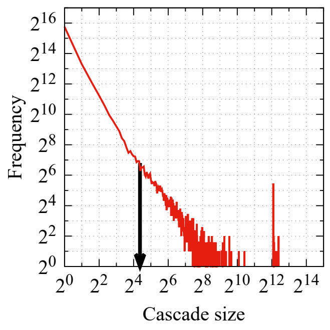
(a) soc-Slashdot0922 (UC$_{0.1}$)



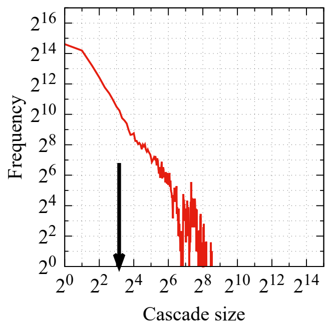
(b) soc-Slashdot0922 (UC$_{0.01}$)



(c) soc-Slashdot0922 (TRI)

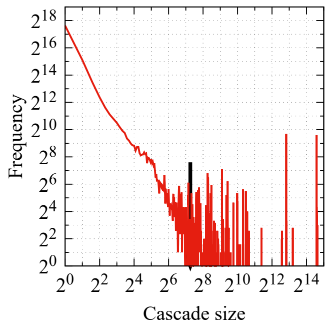


(d) soc-Slashdot0922 (IWC)

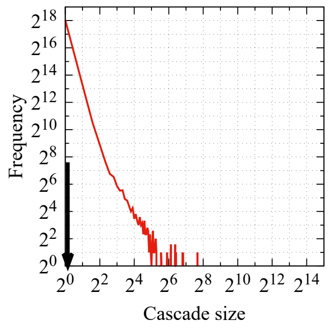


(e) soc-Slashdot0922 (OWC)

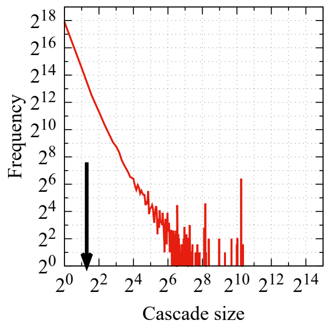
Figure 4.4: Size distribution of reachable sets in soc-Slashdot0922 network.



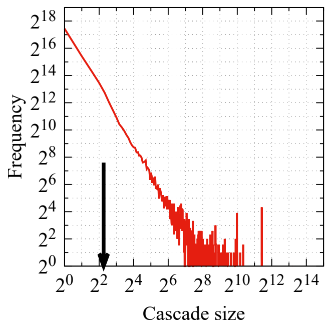
(a) web-Stanford (UC$_{0.1}$)



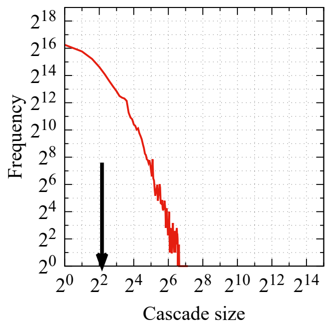
(b) web-Stanford (UC$_{0.01}$)



(c) web-Stanford (TRI)

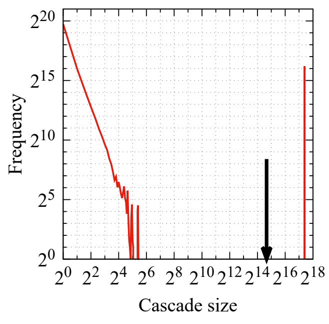


(d) web-Stanford (IWC)

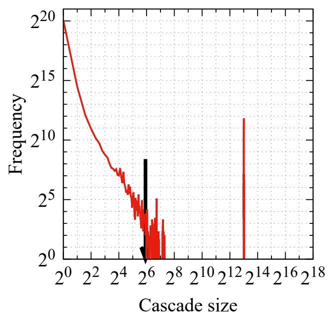


(e) web-Stanford (OWC)

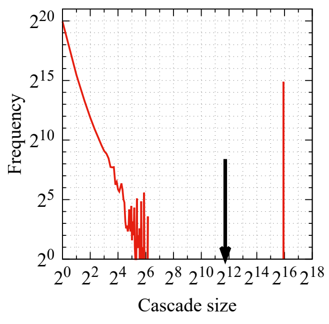
Figure 4.5: Size distribution of reachable sets in web-Stanford network.



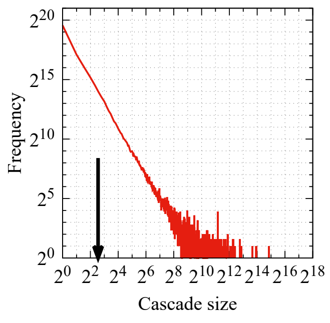
(a) com-Youtube (UC$_{0.1}$)



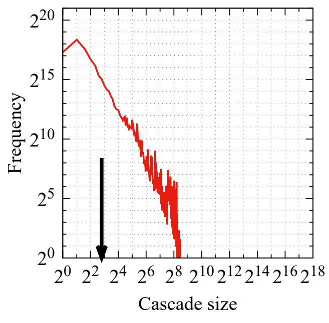
(b) com-Youtube (UC$_{0.01}$)



(c) com-Youtube (TRI)

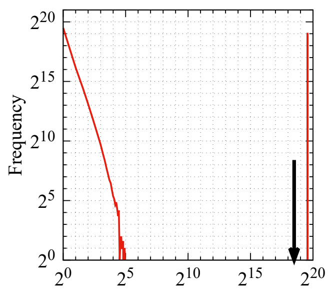


(d) com-Youtube (IWC)

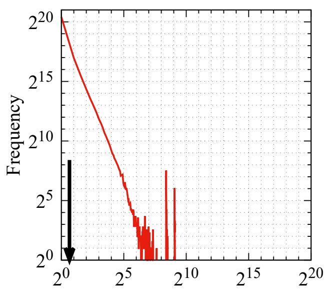


(e) com-Youtube (OWC)

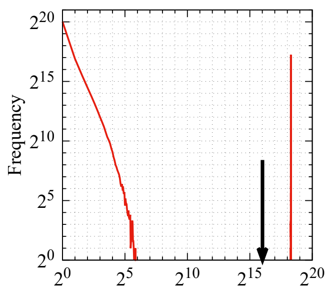
Figure 4.6: Size distribution of reachable sets in com-Youtube network.



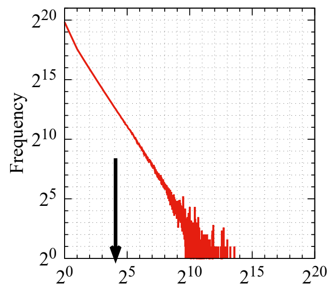
(a) soc-Pokec (UC$_{0.1}$)



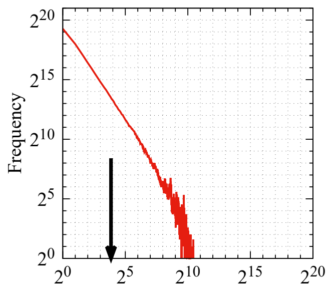
(b) soc-Pokec (UC$_{0.01}$)



(c) soc-Pokec (TRI)

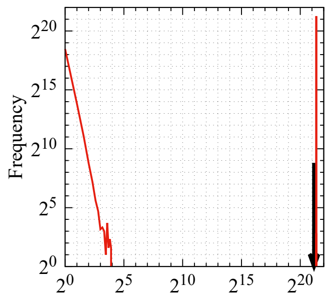


(d) soc-Pokec (IWC)

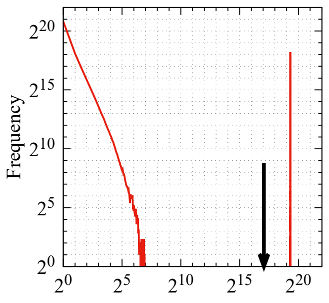


(e) soc-Pokec (OWC)

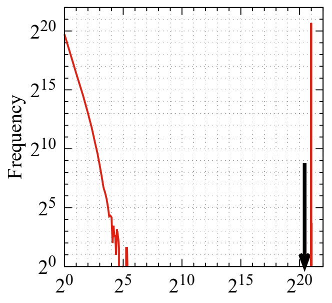
Figure 4.7: Size distribution of reachable sets in soc-Pokec network.



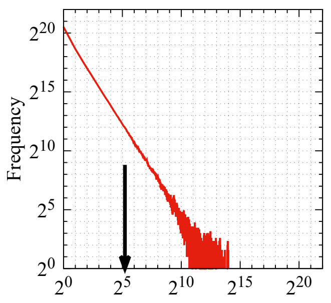
(a) com-Orkut (UC$_{0.1}$)



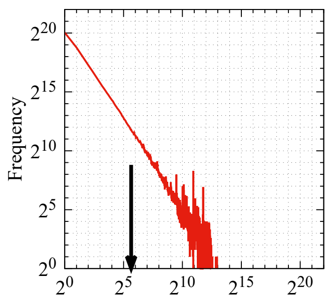
(b) com-Orkut (UC$_{0.01}$)



(c) com-Orkut (TRI)



(d) com-Orkut (IWC)



(e) com-Orkut (OWC)

Figure 4.8: Size distribution of reachable sets in com-Orkut network.

4.4 Analysis of RR Sets

We next focus on RR sets, a key notion in RIS-based algorithms. We generated 10,000 RR sets from an influence graph $\mathcal{G} = (V, E, p)$ independently at random. For each RR set, we recorded a quadruplet (z, R, L, B), where $z \in V$ is the target vertex, $R \subseteq V$ is the RR set, $L \subseteq (R \times R) \cap E$ is the set of live edges, and $B \subseteq (V \times R) \cap E$ is the set of blocked edges. Note that $L \cup B$ contains all edges entering vertices in R.

Table 4.3 reports the average number of vertices in RR sets for each configuration. Figures 4.9–4.12 illustrate the scatter plot, where each point corresponds to an RR set, and $(x, y) = (|R|, |L|)$ or $(x, y) = (|R|, |L| + |B|)$. The black curve is the curve $y = x - 1$. See Appendix A for the complete experimental results. Similarly to the reachable set, there are two types of distribution of $|R|$. Bimodal distributions are obtained only if we use unweighted settings, and both $(|R|, |L|)$ and $(|R|, |L| + |B|)$ concentrate around a point in the upper right corner. In general, $|L| + |B|$ is much larger than $|L|$, and thus, we have

Observation 4.3 (The number of edges entering RR sets)**.** *Storing $(R, L \cup B)$'s consumes much more space than (R, L)'s.*

4.4.1 Unweighted Settings

We then examine unweighted settings. In short, if we use unweighted settings, we may obtain bimodal distribution. In such a case, RR sets are relatively large on average and so RIS-based algorithms may consume a significant amount of space. Table 4.3 tells us that $\textsc{uc}_{0.1}$ may yield RR sets of orders of magnitude greater than $\textsc{uc}_{0.01}$. We can observe the relation $|L| \approx (|L| + |B|) \cdot \bar{p}$, where $\bar{p}$ be the average influence probability. This is not surprising since each edge becomes live with probability $\bar{p}$ and blocked with probability $1-\bar{p}$. Small $(|R|, |L|)$'s lie slightly above the curve $y = x - 1$. Such RR sets seem not intersect the GC. On the other hand, large $(|R|, |L|)$'s are denser than small ones, e.g., soc-Slashdot0922 ($\textsc{uc}_{0.1}$ and $\textsc{tri}$), because they intersect the GC in random graphs.

Observation 4.4 (RR sets under unweighted settings)**.** *If the GC is present, a certain percentage of RR sets are quite large. Moreover, large RR sets include more live edges than small ones.*

4.4.2 Degree-weighted Settings

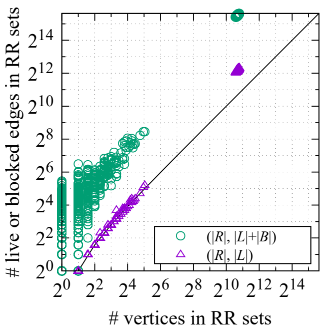
Now, let us look at the results with $\textsc{iwc}$ and $\textsc{owc}$. As expected from the discussion in the previous section, Table 4.3 demonstrates that RR sets are always small even for the three largest networks. Thus, RIS-based algorithms can produce tons of RR sets and perform the greedy algorithm on them efficiently. $(|R|, |L|)$ is well concentrated and slightly above $y = x - 1$; each vertex is expected to have just one in-neighbor and out-neighbor in a random graph under $\textsc{iwc}$ and $\textsc{owc}$, respectively. On the other hand, $(|R|, |L|+|B|)$ is diversely distributed. This is because $(|L| + |B|)/|L|$ can be as large as the maximum in-degree ($\textsc{iwc}$) or out-degree ($\textsc{owc}$).

Observation 4.5 (RR sets under degree-weighted settings)**.** *RR sets under degree-weighted settings are consistently small.*

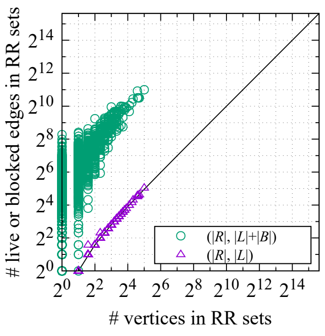
In Chapter 6, we use these observations for the design of a space-efficient index and efficient index update algorithms.

Table 4.3: Average number of vertices in 10,000 RR sets for each configuration of network data and influence probability.

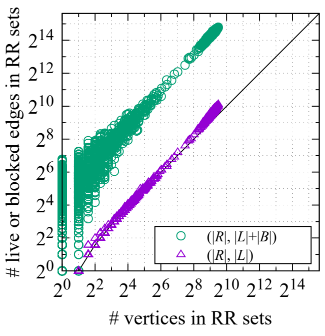
graph	$\lvert V\rvert$	$\text{UC}_{0.1}$	$\text{UC}_{0.01}$	$\text{EXP}_{0.1}$	$\text{EXP}_{0.01}$	TRI	IWC	OWC
ca-GrQc	5,242	12.3	1.1	11.1	1.1	1.8	3.9	4.0
ca-HepTh	9,877	58.3	1.1	51.5	1.1	1.4	4.6	4.6
wiki-Vote	7,115	386.8	1.3	374.7	1.3	52.1	1.8	3.8
ca-HepPh	12,008	1,287.3	19.3	1,302.4	19.1	240.3	7.1	7.3
soc-Epinions1	75,879	1,807.6	1.9	1,860.7	1.9	300.9	5.2	10.1
soc-Slashdot0922	82,168	6,481.1	7.8	6,538.7	7.0	1,386.0	11.5	8.8
web-NotreDame	325,729	26.5	1.5	26.4	1.5	4.8	2.3	8.1
ego-Twitter	81,306	17,303.5	4.0	17,214.3	3.7	3,064.3	9.8	8.0
loc-Gowalla	196,591	14,157.9	17.8	14,345.9	19.0	1,675.1	8.1	10.0
web-Stanford	281,903	107.4	1.1	112.3	1.1	2.2	5.1	5.4
wiki-Talk	2,394,385	4,114.6	36.5	3,919.2	32.4	751.4	13.6	1.2
web-Google	875,713	12.5	1.1	13.4	1.1	1.4	4.1	4.0
com-Youtube	1,134,890	24,788.4	50.4	25,200.8	66.7	3,338.0	5.9	4.6
web-BerkStan	685,230	540.8	1.3	500.1	1.4	12.4	5.4	4.4
higgs-twitter	456,626	106,242.1	558.5	106,529.4	477.6	30,670.9	12.9	26.8
soc-Pokec	1,632,803	367,293.6	1.4	374,522.3	1.5	64,445.8	16.3	15.7
soc-LiveJournal1	4,847,571	600,373.3	426.7	590,242.2	355.6	90,555.5	11.0	10.7
com-Orkut	3,072,441	2,367,933.1	137,191.6	2,359,413.3	138,229.2	1,407,658.5	49.5	46.5



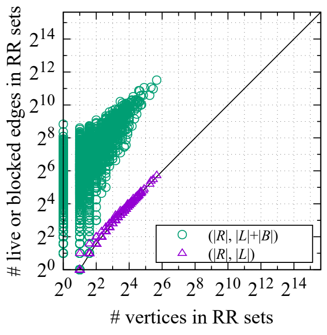
(a) wiki-Vote ($\text{UC}_{0.1}$)



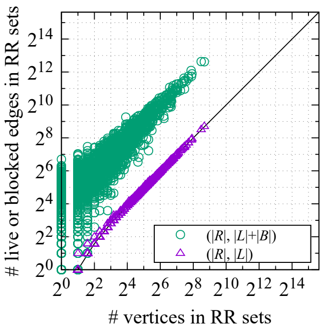
(b) wiki-Vote ($\text{UC}_{0.01}$)



(c) wiki-Vote (TRI)

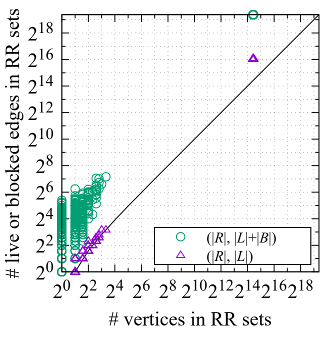


(d) wiki-Vote (IWC)

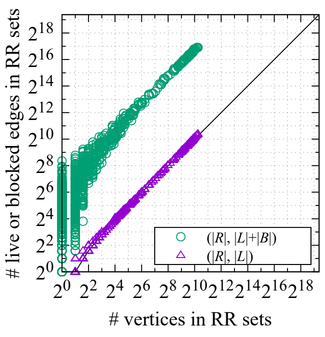


(e) wiki-Vote (OWC)

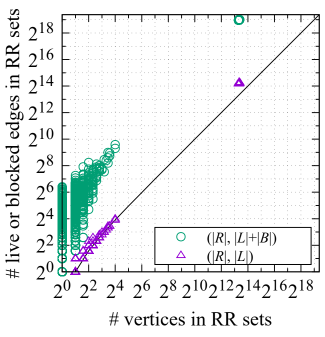
Figure 4.9: Structures of RR sets in wiki-Vote network.



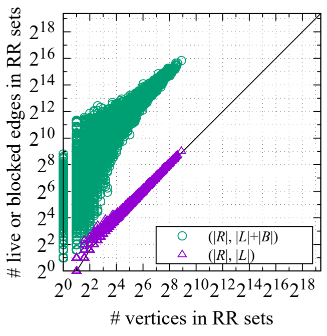
(a) soc-Slashdot0922 ($\text{UC}_{0.1}$)



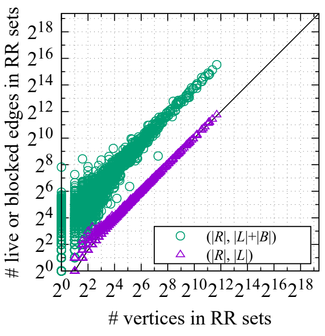
(b) soc-Slashdot0922 ($\text{UC}_{0.01}$)



(c) soc-Slashdot0922 (TRI)

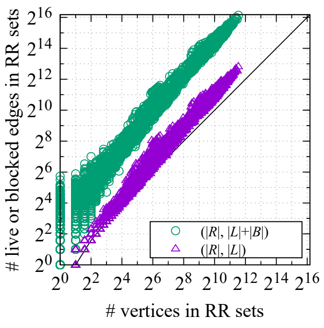


(d) soc-Slashdot0922 (IWC)

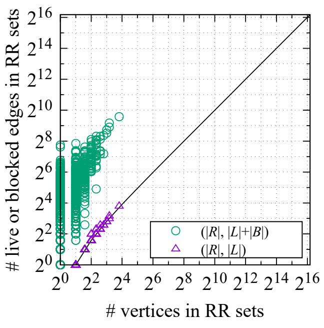


(e) soc-Slashdot0922 (OWC)

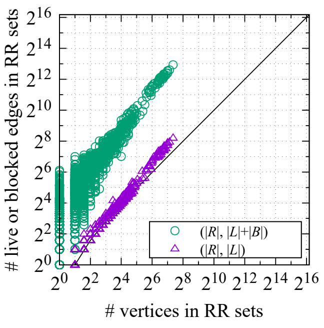
Figure 4.10: Structures of RR sets in soc-Slashdot0922 network.



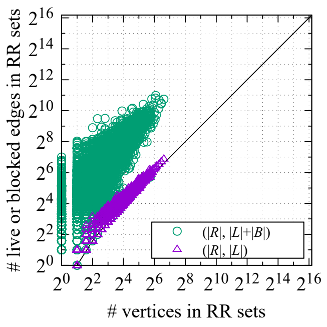
(a) web-Stanford (UC$_{0.1}$)



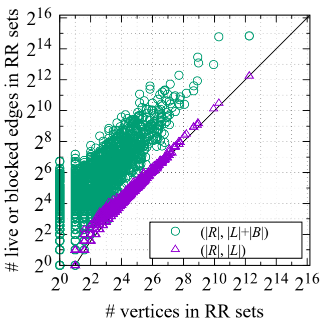
(b) web-Stanford (UC$_{0.01}$)



(c) web-Stanford (TRI)

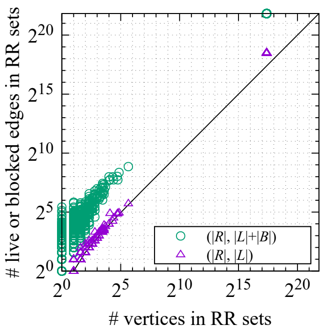


(d) web-Stanford (IWC)

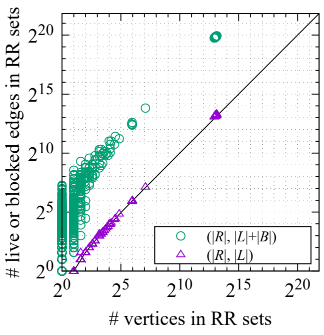


(e) web-Stanford (OWC)

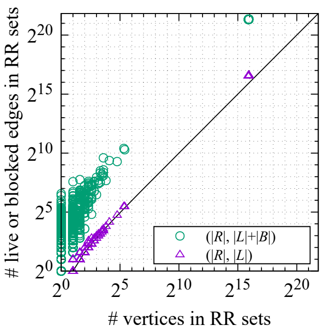
Figure 4.11: Structures of RR sets in web-Stanford network.



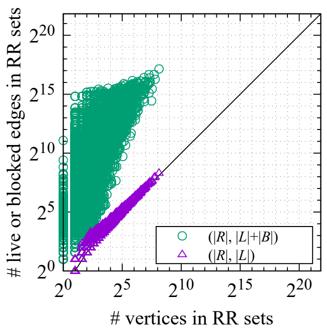
(a) com-Youtube (UC$_{0.1}$)



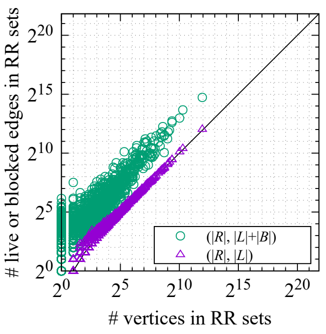
(b) com-Youtube (UC$_{0.01}$)



(c) com-Youtube (TRI)

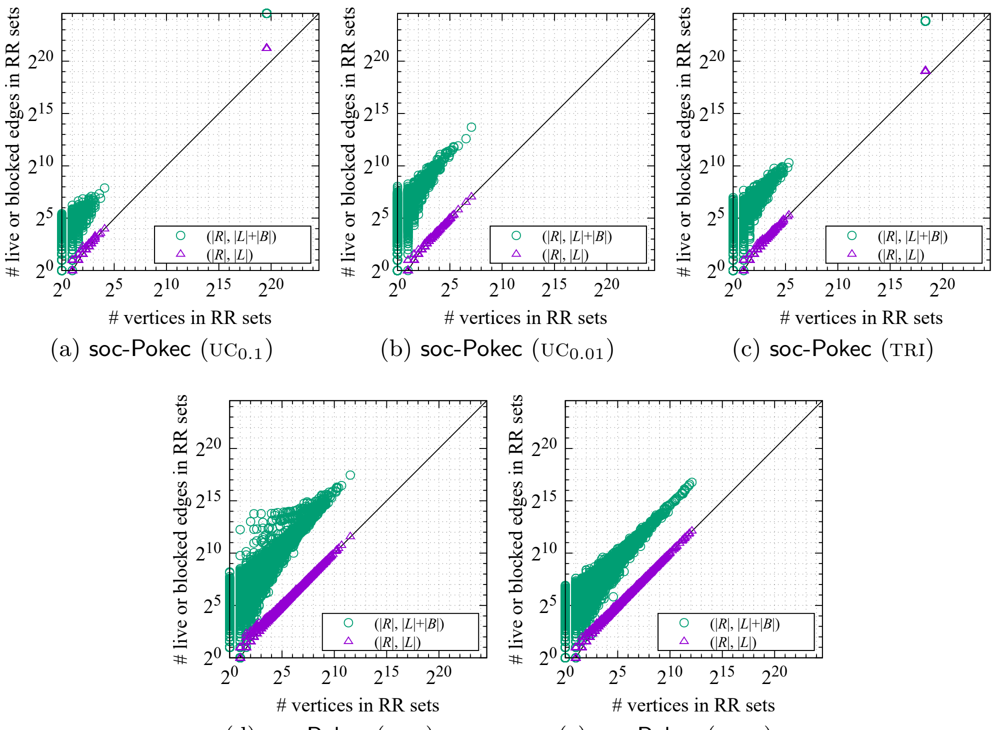


(d) com-Youtube (IWC)



(e) com-Youtube (OWC)

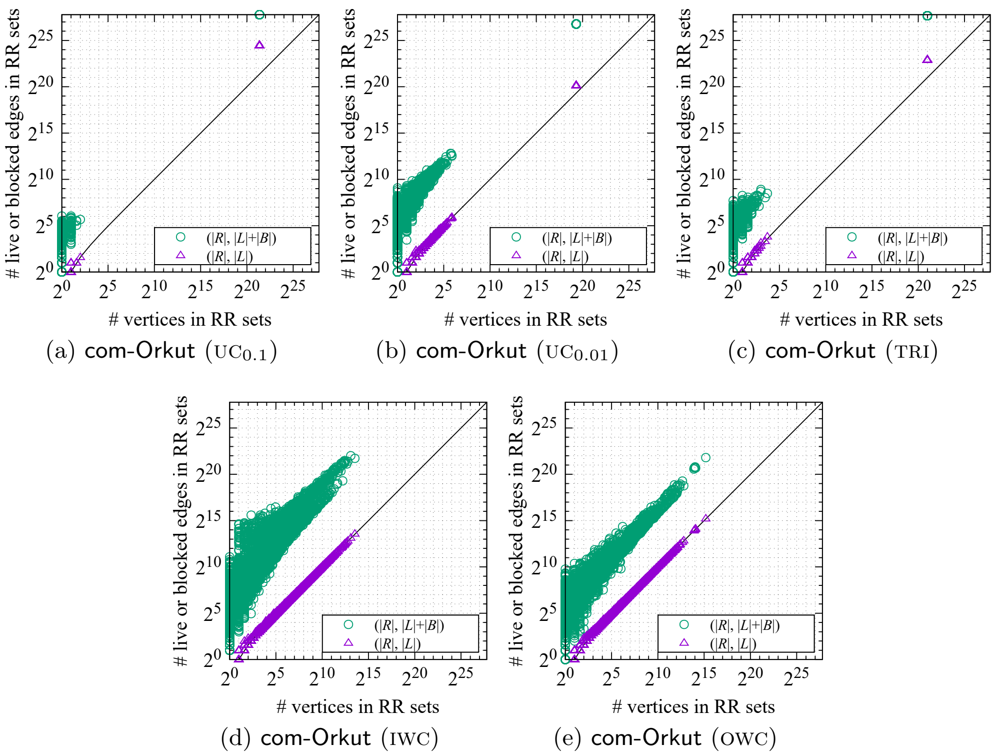
Figure 4.12: Structures of RR sets in com-Youtube network.



(a) soc-Pokec ($\text{UC}_{0.1}$) (b) soc-Pokec ($\text{UC}_{0.01}$) (c) soc-Pokec (TRI)

(d) soc-Pokec (IWC) (e) soc-Pokec (OWC)

Figure 4.13: Structures of RR sets in soc-Pokec network.



(a) com-Orkut ($\text{UC}_{0.1}$) (b) com-Orkut ($\text{UC}_{0.01}$) (c) com-Orkut (TRI)

(d) com-Orkut (IWC) (e) com-Orkut (OWC)

Figure 4.14: Structures of RR sets in com-Orkut network.

Chapter 5

Fast Algorithm for Influence Maximization

In this chapter, we propose an efficient algorithm for influence maximization called *pruned Monte-Carlo simulations (PMC)*. Followed by an overview of PMC (Section 5.1), we describe the proposed techniques in detail (Sections 5.2 and 5.3), and then we bring them together (Section 5.4). We conduct extensive experiments to compare PMC with existing algorithms and analyze the effectiveness of the proposed techniques (Section 5.5).

5.1 Overview

Our algorithm *pruned Monte-Carlo (PMC)* is a snapshot-based algorithm. Recall from Section 3.2.2 that snapshot-based algorithms [46, 106, 108] repeatedly solve descendant counting for r random graphs $\{G_i\}_{i\in[r]}$ which were sampled from an input influence graph $\mathcal{G} = (V, E, p)$. In fact, it suffices to perform descendant counting for those condensation DAGs. Let $\{(G'_i, \pi_i, \mathsf{w}_i)\}_{i\in[r]}$ be condensations, correspondence mappings, vertex weights obtained from $\{G_i\}_{i\in[r]}$ by taking those condensations. Let $s_\ell \in V$ denote the seed vertex chosen at the ℓ^{th} iteration during greedy seed selection and let $S_\ell := \{s_1, \ldots, s_\ell\}$, where $S_0 := \emptyset$. Then, for each $\ell \in [k]$,

$$s_\ell = \operatorname*{argmax}_{v\in V\setminus S_{\ell-1}} \sum_{i\in[r]} \mathsf{r}_{G'_i,\mathsf{w}_i}(\pi_i(S_{\ell-1}\cup\{v\})). \tag{5.1}$$

We can further use the technique of [106, 108] to compute $\mathsf{r}_{G'_i,\mathsf{w}_i}(\pi_i(S_{\ell-1}\cup\{v\}))$ for each $v \in V$. For ease of notation, let ${G'_i}^{(\ell)}$ denote the subgraph of G'_i obtained by removing the reachable set of $\pi_i(S_\ell)$, i.e.,

$${G'_i}^{(\ell)} := G'_i - \mathsf{R}_{G'_i}(\pi_i(S_\ell)), \tag{5.2}$$

or equivalently, ${G'_i}^{(\ell)}$ can be defined in a recursive way:

$${G'_i}^{(0)} := G'_i, \tag{5.3}$$

$${G'_i}^{(\ell)} := {G'_i}^{(\ell-1)} - \mathsf{R}_{{G'_i}^{(\ell-1)}}(\pi_i(s_\ell)) \quad (\ell \in [k]). \tag{5.4}$$

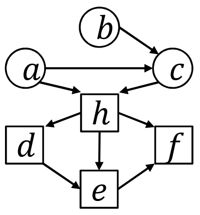
Then, it follows that for each $\ell \in [k]$,

$$s_\ell = \operatorname*{argmax}_{v\in V\setminus S_{\ell-1}} \sum_{i\in[r]} \mathsf{r}_{{G'_i}^{(\ell-1)},\mathsf{w}_i}(\pi_i(v)). \tag{5.5}$$

Hence, we are able to identify the ℓ^{th} seed s_ℓ by solving descendant counting on ${G'_i}^{(\ell-1)}$'s.

Algorithm 5.1 Proposed pruned BFS.

```
 1: function PREPROCESS($G'_i$)
 2:     $h_i \leftarrow$ a vertex with the maximum degree in $V(G'_i)$.
 3:     $D_i \leftarrow$ a set of $h_i$'s descendants in $G'_i$.
 4:     $A_i \leftarrow$ a set of $h_i$'s ancestors in $G'_i$ (excluding $h_i$ itself).
 5:     compute $\mathsf{r}_{G'_i,\mathsf{w}_i}(h_i)$ by a BFS.
 6: function PRUNEDBFS($s \in G'_i$, $G'_i$)
 7:     if $s \in A_i$ then
 8:         $\mathtt{gain}_i[s] \leftarrow \mathsf{r}_{G'_i}(h_i)$.                ▷ In constant time.
 9:     else
10:         $\mathtt{gain}_i[s] \leftarrow 0$.
11:     $Q \leftarrow$ a queue with only one element $s$.
12:     while $Q \neq \emptyset$ do
13:         dequeue $u$ from $Q$.
14:         if $s \in A_i \wedge u \in D_i$ then
15:             continue.                                       ▷ Pruning works.
16:         $\mathtt{gain}_i[s] \leftarrow \mathtt{gain}_i[s] + \mathsf{w}_i(u)$.
17:         for all $v \in \mathcal{N}^+_{G'_i}(u)$ do
18:             if $v$ has not been visited so far then
19:                 enqueue $v$ onto $Q$.
```



vertex	# vertices visited during naive BFS	pruned BFS
a	6	2
b	6	2
c	5	1

Figure 5.1: An example of pruned BFS. Square vertices are temporarily removed during a BFS from a circular vertex.

However, this is still not efficient enough to deal with large networks because we have to compute $\mathsf{r}_{G'^{(\ell)}_i}(v)$ for all $v \in V(G'_i)$, $i \in [r]$, and $\ell \in [k]$. We resolve this issue with new boosting algorithms, i.e., *pruned BFS* and *BFS avoidance* as will be described in the following. Note that both techniques do not affect the estimates of the influence spread.

5.2 Technique 1: Pruned BFS

We first propose *pruned BFS*. Hereafter, for two vertices u, v such that u can reach v, u is called an *ancestor* of v and v is called a *descendant* of u. Recall from Observation 4.1 that the computational cost of descendant counting becomes large if the GC is present. However, if we "know" the existence of the GC, we can significantly reduce the cost as follows. For an input graph G, let G', π, and w be the condensation, correspondence mapping, vertex weights obtained from G. We take a *hub* vertex h in G', which corresponds to the GC in G. Then, for any vertex $v \in V(G')$, the following holds:

$$h \in \mathsf{R}_{G'}(v) \Rightarrow \mathsf{r}_{G'}(v) = \mathsf{r}_{G'}(h) + \mathsf{r}_{G'-\mathsf{R}_{G'}(h)}(v). \tag{5.6}$$

Algorithm 5.2 Proposed BFS avoidance.

```
1: function UPDATE($G_i'^{(\ell-1)}$, $s_\ell \in V$)
2:     compute a set $F_i$ of vertices reachable from $\pi_i(s_\ell)$ in $G_i'^{(\ell-1)}$.
3:     compute a set $B_i$ of vertices that can reach some vertex in $F_i$ in $G_i'^{(\ell-1)}$.
4:     $\mathtt{old}_i \leftarrow B_i$.        ▷ $\mathtt{gain}_i[v]$ for $v \in \mathtt{old}_i$ is outdated.
5:     remove vertices in $F_i$ and edges leaving or entering $F_i$ from $G_i'^{(\ell-1)}$ to
   obtain $G_i'^{(\ell)}$.
6: function BFS($s \in V(G_i')$, $G_i'^{(\ell-1)}$)        ▷ After the first iteration.
7:     if $s \notin G_i'^{(\ell-1)}$ then
8:         $\mathtt{gain}_i[s] \leftarrow 0$.
9:     else if $s \notin \mathtt{old}_i$ then
10:        NOP.        ▷ BFS avoided.
11:    else
12:        run BFS from $s$ on $G_i'^{(\ell-1)}$ to obtain $\mathsf{r}_{G_i'^{(\ell-1)},\mathsf{w}_i}(s)$ and set it to $\mathtt{gain}_i[s]$.
```

In other words, if v can reach h, then we can compute $\mathsf{r}_{G'}(v)$ by conducting a BFS starting from v on $G' - \mathsf{R}_{G'}(h)$. We can approximately reduce the computational cost by the number of ancestors of v times the number of descendants of v. We simply select a vertex with the maximum sum of in-degree and out-degree as the hub vertex h.

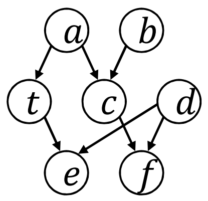
Algorithm 5.1 shows pseudocode of the proposed pruned BFS. In preprocessing for each G_i', we select a hub vertex h_i in G_i' and compute a set D_i of the descendants of h_i and a set A_i of the ancestors of h_i. Then, we run a BFS starting from h_i on G_i' to compute $\mathsf{r}_{G_i',\mathsf{w}_i}(h_i)$. This requires $\mathcal{O}(|V(G_i')|+|E(G_i')|)$ time. Having done, we compute $\mathsf{r}_{G_i',\mathsf{w}_i}(s)$ for each vertex $s \in V(G_i')$ as follows. First, we check whether s is an ancestor of h_i or not. If this is the case, we perform a pruned BFS from s. That is, we run a BFS starting from s on $G_i' - \mathsf{R}_{G_i'}(h_i)$ and return the sum of $\mathsf{r}_{G_i',\mathsf{w}_i}(h_i)$ and $\mathsf{r}_{G_i'-\mathsf{R}_{G_i'}(h_i),\mathsf{w}_i}(s)$, which is exactly equal to $\mathsf{r}_{G_i',\mathsf{w}_i}(s)$. Otherwise, we conduct a naive BFS from s on G_i'.

Example 5.1. *Figure 5.1 shows an example of pruned BFS. In this figure, a vertex h is a hub because having the maximum degree, and the ancestors and descendants of h are $\{a,b,c\}$ and $\{h,d,e,f\}$, respectively. If running a naive BFS starting from a, we will visit $\{a,c,h,d,e,f\}$. That is redundant because we already know that a can reach to h, and so are to the descendants of h. On other hand, a pruned BFS from a, which prunes $\{h,d,e,f\}$, will only visit two vertices $\{a,c\}$. Similarly, starting from b, a pruned BFS visits $\{b,c\}$ while a naive BFS visits $\{b,c,h,d,e,f\}$.*

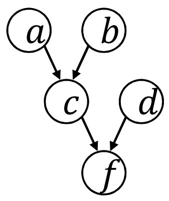
5.3 Technique 2: BFS Avoidance

Then, we propose *BFS avoidance.* Recall that the ℓth iteration during greedy seed selection requires to compute $\mathsf{r}_{G_i'^{(\ell-1)},\mathsf{w}_i}(v)$ for all $i \in [r]$ and all vertices $v \in V(G_i')$, and then the $(\ell+1)$st iteration asks to compute $\mathsf{r}_{G_i'^{(\ell)},\mathsf{w}_i}(v)$ for all $i \in [r]$ and all vertices $v \in V(G_i')$. Let us consider how to "guess" $\mathsf{r}_{G_i'^{(\ell)},\mathsf{w}_i}(v)$ by reusing the information at the last iteration. Our insight is the following.

$$\mathsf{R}_{G_i'^{(\ell-1)}}(v) \cap \mathsf{R}_{G_i'^{(\ell-1)}}(\pi_i(s_\ell)) = \emptyset \Longrightarrow \mathsf{r}_{G_i'^{(\ell)},\mathsf{w}_i}(v) = \mathsf{r}_{G_i'^{(\ell-1)},\mathsf{w}_i}(v). \tag{5.7}$$



(a) G.



(b) $G - \mathsf{R}_G(t)$.

v	$\mathsf{r}_G(v)$	$\mathsf{r}_{G-\mathsf{R}_G(t)}(v)$
a	5	3
b	3	3
c	2	2
d	3	2
e	1	0
f	1	1
t	2	0

Figure 5.2: Example of BFS avoidance.

Hence, detecting such vertices, we can avoid unnecessary BFSes. Fortunately, it is possible to find all the vertices efficiently.

Algorithm 5.2 shows pseudocode of BFS avoidance. Given the ℓ^{th} seed s_ℓ, we conduct the following update procedure to the DAGs $G_i'^{(\ell-1)}$. We first compute a set F_i consisting of vertices reachable from $\pi_i(s_\ell)$ by a BFS, and we then conduct a *reverse* BFS from F_i to compute a set B_i consisting of vertices that can reach some vertex in F_i. Here, the set B_i contains every vertex that does *not* satisfy the condition in Eq. (5.7), and we store them in $\texttt{old}_i$. Note that we remove vertices in F_i and edges leaving or entering F_i from $G_i'^{(\ell-1)}$ to obtain $G_i'^{(\ell)}$ at that time. At the $(\ell+1)^{\text{st}}$ iteration, we have three cases to compute $\mathsf{r}_{G_i'^{(\ell)},\mathsf{w}_i}(s)$.

- **Case 1:** s has been already removed, then the answer is zero.
- **Case 2:** If the reachable set of s did not change (denoted $s \notin \texttt{old}_i$ in the code), then we reuse the previous result (denoted $\texttt{gain}_i[s]$ in the code) without any BFS.
- **Case 3:** Otherwise, run a BFS from a vertex s naively.

Example 5.2. *Let us take an example in Figure 5.2. Suppose that a vertex t is chosen as a seed. Then, we remove the descendants of t, i.e., t and e, from the current graph G and obtain a new DAG H. At that time, we can see that $\mathsf{r}_G(\cdot)$ changes for vertices that can reach to t or e, that is, a, d, e, and t. This fact tells us that we do not need to recompute the gain of b, c, and f.*

5.4 Putting It Together

Combining the aforementioned two techniques, we obtain a new snapshot-based algorithm for influence maximization as shown in Algorithm 5.3. In the first iteration, we use pruned BFS for each DAG. In the second and following iterations, we use BFS avoidance. For each iteration, we select a seed vertex greedily. We notice that the proposed algorithm requires $\mathcal{O}(r(|V|+|E|))$ space.

5.4.1 Degree-1 Optimization

Here, we briefly describe a minor improvement upon BFSes. Assume that a vertex u has one unique out-neighbor v, then, it turns out that

$$\mathsf{r}(u) = \mathsf{r}(v) + \mathsf{w}(v), \tag{5.8}$$

and thus, once we have computed $\mathsf{r}(v)$, we need not to run a BFS from u.

Algorithm 5.3 Pruned Monte-Carlo simulation.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a seed set $k$, the number of random
    graphs $r$
 1: for $i = 1$ to $r$ do
 2:     sample a random graph $G_i$ independently from $\mathcal{G}$.
 3:     construct a DAG $G'_i$ with vertex weights $\mathsf{w}_i$ and a mapping $\pi_i$ from $G_i$.
 4:     call PREPROCESS($G'_i$).
 5: $S_0 \leftarrow \emptyset$.
 6: for $\ell = 1$ to $k$ do
 7:     if $\ell = 1$ then
 8:         call PRUNEDBFS($v$, $G_i'^{(\ell-1)}$) for all $v \in V(G'_i)$.
 9:     else
10:         call BFS($v$, $G_i'^{(\ell-1)}$) for all $v \in V(G'_i)$.
11:     $s_\ell \leftarrow \mathrm{argmax}_{v \in V} \frac{1}{r} \sum_{i \in r} \mathtt{gain}_i[\pi_i(v)]$.    ▷ $\mathtt{gain}_i[\pi_i(v)] = \mathsf{r}_{G_i'^{(\ell-1)}, \mathsf{w}_i}(\pi_i(v))$
12:     $S_\ell \leftarrow S_{\ell-1} \cup \{s_\ell\}$.
13:     UPDATE($G_i'^{(\ell-1)}$, $s_\ell$).
14: return $S_k$.
```

5.5 Experiments

5.5.1 Setup

Datasets.

We use real-world networks introduced in Chapter 4, Table 4.1 provides a summary of the networks. In brief, we use 18 networks, and the number of vertices varies from five thousand to three million and the number of edges varies from 30 thousand to 200 million.

Influence probability settings.

To investigate the behavior of each method below with various probability settings, we adopt the settings of influence probabilities introduced in Chapter 4, i.e., uniform cascade $\text{UC}_{0.1}$ and $\text{UC}_{0.01}$, exponential $\text{EXP}_{0.1}$ and $\text{EXP}_{0.01}$, trivalency TRI, in-degree weighted IWC, and out-degree weighted OWC.

Algorithms and implementations.

The proposed algorithm is parameterized by r. We set the value of r to 200. We compare our algorithm with the following algorithms. The value of each parameter is set to the one suggested by the corresponding paper.

The following simulation-based algorithms will be used.

- *CELF++* [80]: A standard greedy algorithm with lazy evaluations plus a pruning technique. We set the number of simulations as 10,000. We downloaded the implementation from the website of Amit Goyal [78].

- *StaticGreedy* [46]: A snapshot-based algorithm with no optimization. We set the number of simulations as 200. We downloaded the implementation from the repository [11] for influence maximization benchmarking [12].

- *StaticGreedyDU* [46]: A snapshot-based algorithm with speeding-up techniques. We set the number of random graphs as 200. We downloaded the implementation from the repository [11] for influence maximization benchmarking [12].

The following RIS-based algorithms will be used.

- *IMM* [177]: An RIS-based algorithm with a martingale technique and a statistical test method. We set the parameters as $\ell = 1$ and $\epsilon = 0.1$. We downloaded the implementation (version 1.1) from [90].

- *D-SSA* [148]: An RIS-based algorithm with a search-and-verify approach. We set the parameters as $\ell = 1$ and $\epsilon = 0.1$. We downloaded the implementation (version 2.1) from [146].

The following heuristic algorithm will be used.

- *IRIE* [96]: A heuristic algorithm using linear systems. We set the parameters as $\alpha = 0.7$ and $\theta = 1/320$. We downloaded the implementation from [11].

- *IMRank* [47]: A heuristic algorithm based on non-greedy strategy. We set the parameter as $\ell = 1$. We downloaded the implementation from [11].

- *Degree* [98]: A baseline algorithm that chooses the topmost k vertices in decreasing order of out-degrees.

Environments.

All algorithms were implemented in C++ and compiled using g++v4.8.2 with the -O2 option. We conducted experiments on a Linux server with Intel Xeon E5-2670 (2.60GHz) CPU and 512 GB memory. However, we set limits on the system resources by the `setrlimit` system call so that each program cannot consume over 256 GB.

5.5.2 Performance Comparison with Existing Algorithms

We set $k = 1, 5, 10, 20, 40, 100, 200, 400, 1000$ and run the algorithms for each configuration of the graph, the influence probability, and the seed size. However, some of the settings resulted in out-of-memory error (i.e., the memory usage exceeds 256 GB) or did not finish in three hours. We were unable to obtain seed sets in these settings.

Influence spread.

We begin with the quality of solutions. Figures 5.3–5.12 show the influence spread of the solutions that algorithms produced for each configuration. See Appendix B for complete experimental results. We ran Monte-Carlo simulations of the diffusion process $10,000$ times for each seed set and took the average to obtain reasonable estimates of the influence spread.

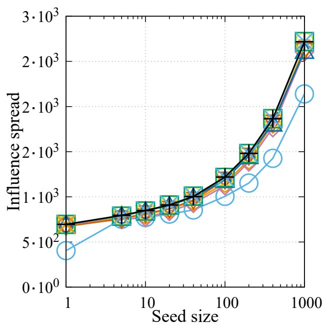
We first investigate an overall trend. In general, we can see the increase in the influence spread every time k increases. However, we can observe the influence immediately reaches a plateau on com-Orkut ($\textsc{uc}_{0.1}$, $\textsc{uc}_{0.01}$, and $\textsc{tri}$). This means that choosing the most influential vertex is enough to trigger wide cascades. Plus, such vertices are easy to guess even for *Degree* heuristic.

Now we compare *PMC* with each of the algorithms. As representative instances of simulations-based algorithms, we ran *CELF++*, *StaticGreedy*, and

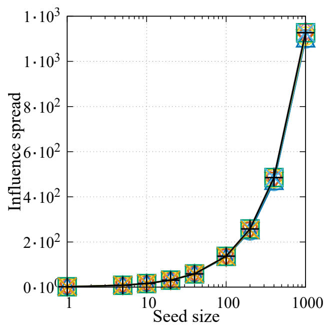
StaticGreedyDU. *CELF++* often produces less influential solutions than *PMC*, e.g., 16% less on ca-GrQc (TRI, $k = 100$) and 20% less wiki-Vote (OWC, $k = 1$). This is because naive estimation algorithms require more simulations than snapshot-based algorithms requires. *StaticGreedy* and *StaticGreedyDU* return almost identical solutions to those of *PMC*; all of the techniques incorporated into each algorithm do not affect influence estimates. However, these algorithms were not able to handle large-scale networks as discussed later.

As representative instances of RIS-based algorithms, we ran *IMM* and *D-SSA*. *IMM* gives closely-influential seed sets to *PMC* in general. Besides, *D-SSA* tends to generate a fewer number of RR sets than *IMM*, and hence, its solution quality is almost always worse than *IMM*. Further, it results in quite bad solutions under some configurations, e.g., the seed set of *D-SSA* on ca-HepTh (EXP$_{0.01}$, $k = 1$) and ca-HepTh (EXP$_{0.01}$, $k = 5$) is 38% and 12% less influential than *PMC*, respectively. One exceptional probability setting is OWC; *IMM* has clear advantages over *PMC*. For example, *IMM*'s solution is 23% more influential on soc-Epinions1 (OWC, $k = 1$), 19% more influential on soc-Pokec (OWC, $k = 1$), and 16% more influential on soc-LiveJournal1 (OWC, $k = 1$).

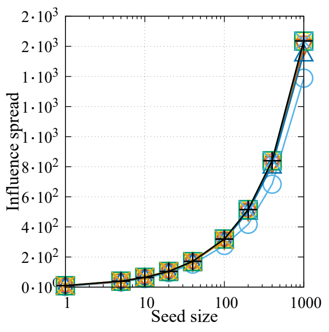
Finally, we compare *PMC* with heuristics including *IRIE*, *IMRank*, and *Degree*. These heuristics often misestimate the actual influence spread. Hence, they selected less influential seeds under IWC and OWC. This is the case even under unweighted settings. To take a few example, compared to the solutions of *PMC*, *IMRank* ($\ell = 1$) shows 82% and 23% less influence on wiki-Talk (UC$_{0.01}$, $k = 1$) and web-BerkStan (TRI, $k = 5$), respectively; *IMRank* ($\ell = 2$) shows 14% and 12% less influence on web-BerkStan (UC$_{0.01}$, $k = 5$) and web-NotreDame (UC$_{0.1}$, $k = 10$), respectively; *Degree* shows 82% and 12% less influence on wiki-Talk (UC$_{0.01}$, $k = 1$) and higgs-twitter (UC$_{0.01}$, $k = 1,000$), respectively.



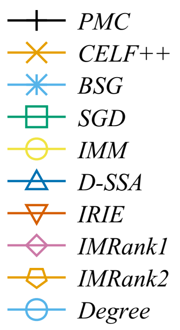
(a) ca-HepTh (UC$_{0.1}$)

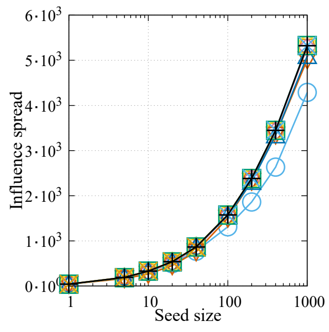


(b) ca-HepTh (UC$_{0.01}$)

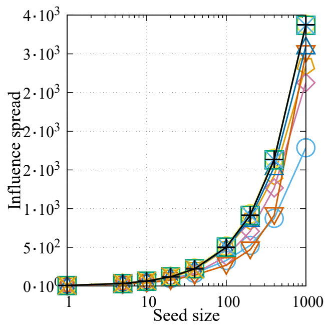


(c) ca-HepTh (TRI)



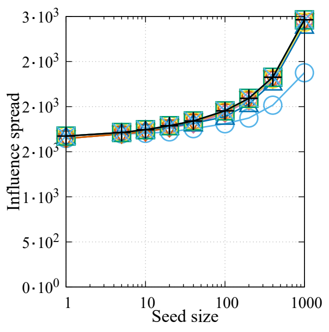


(d) ca-HepTh (IWC)

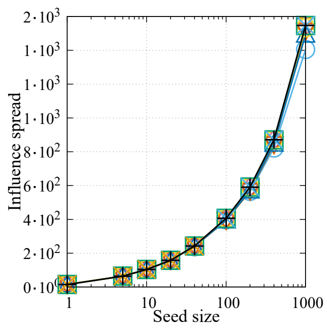


(e) ca-HepTh (OWC)

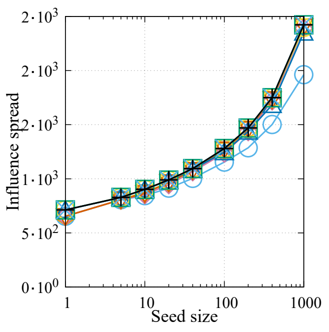
Figure 5.3: Influence spread of each algorithm for ca-HepTh network.



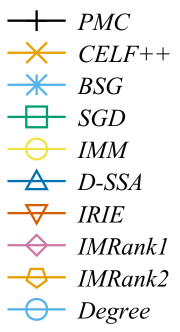
(a) wiki-Vote (UC$_{0.1}$)

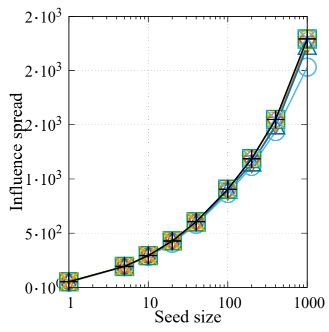


(b) wiki-Vote (UC$_{0.01}$)

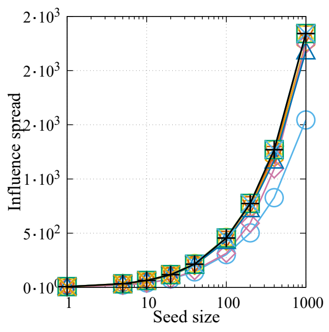


(c) wiki-Vote (TRI)



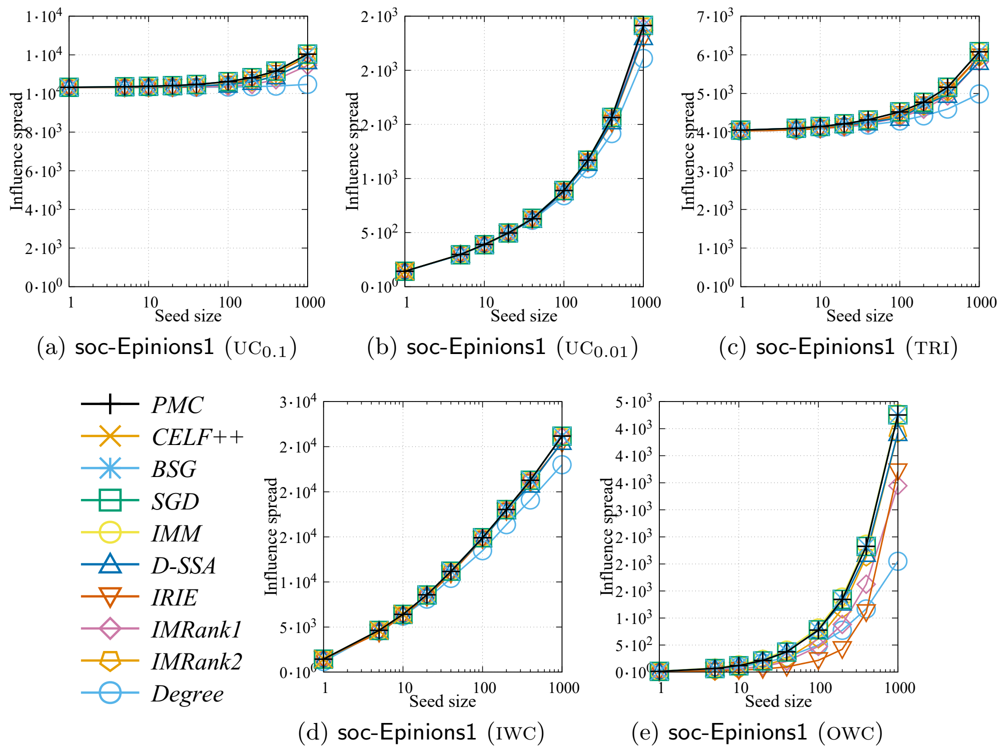


(d) wiki-Vote (IWC)



(e) wiki-Vote (OWC)

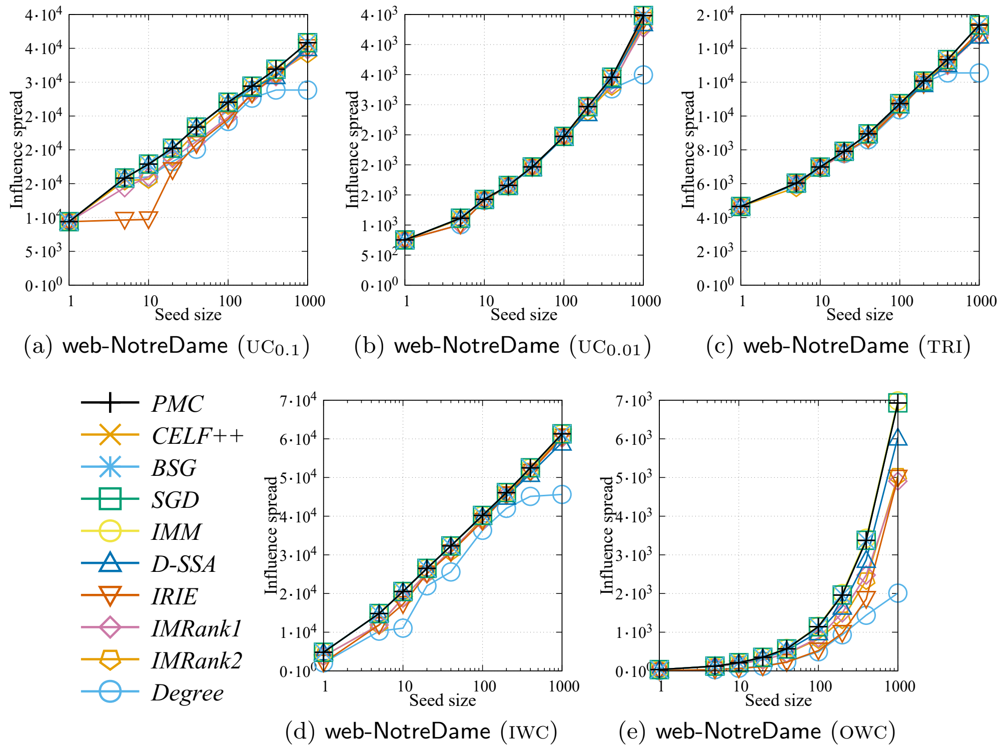
Figure 5.4: Influence spread of each algorithm for wiki-Vote network.



(a) soc-Epinions1 ($\text{UC}_{0.1}$) (b) soc-Epinions1 ($\text{UC}_{0.01}$) (c) soc-Epinions1 (TRI)

(d) soc-Epinions1 (IWC) (e) soc-Epinions1 (OWC)

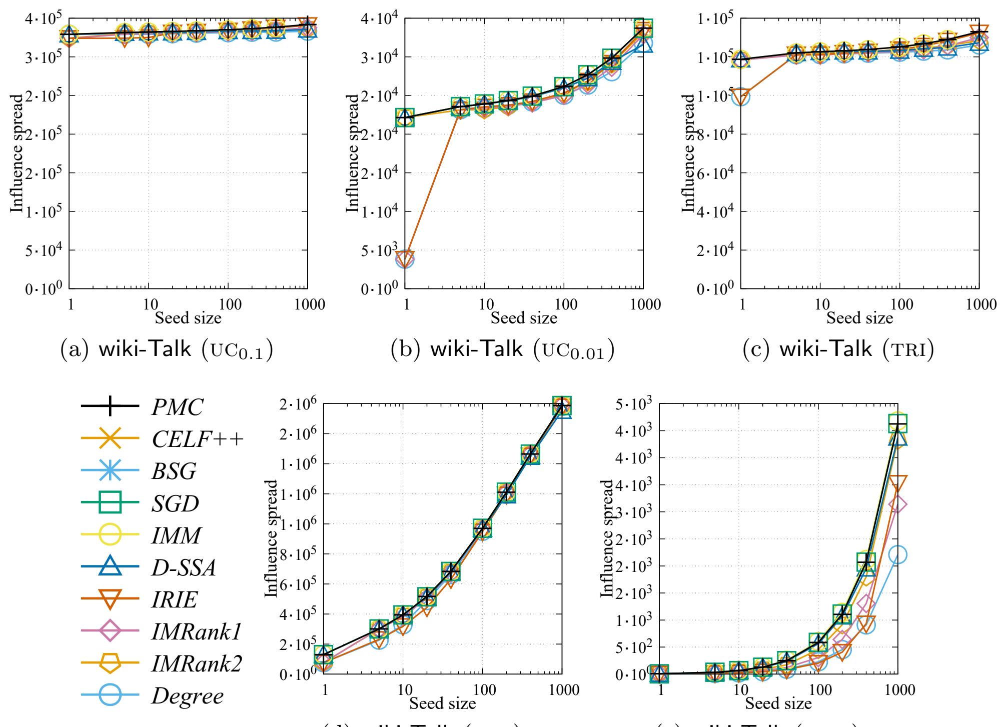
Figure 5.5: Influence spread of each algorithm for soc-Epinions1 network.



(a) web-NotreDame ($\text{UC}_{0.1}$) (b) web-NotreDame ($\text{UC}_{0.01}$) (c) web-NotreDame (TRI)

(d) web-NotreDame (IWC) (e) web-NotreDame (OWC)

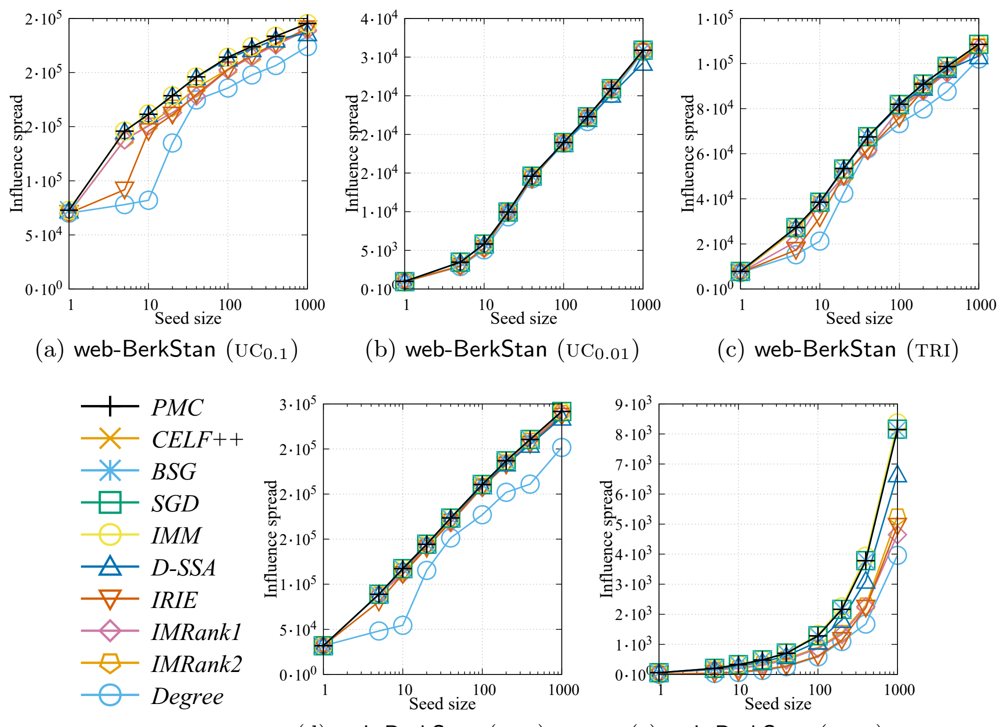
Figure 5.6: Influence spread of each algorithm for web-NotreDame network.



(a) wiki-Talk ($\text{UC}_{0.1}$) (b) wiki-Talk ($\text{UC}_{0.01}$) (c) wiki-Talk (TRI)

(d) wiki-Talk (IWC) (e) wiki-Talk (OWC)

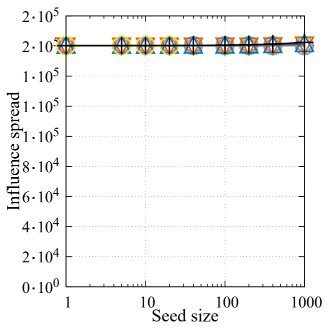
Figure 5.7: Influence spread of each algorithm for wiki-Talk network.



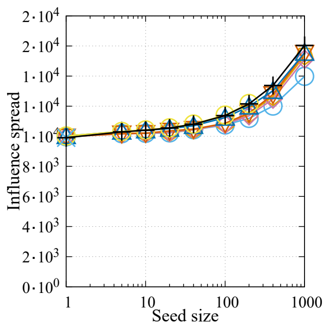
(a) web-BerkStan ($\text{UC}_{0.1}$) (b) web-BerkStan ($\text{UC}_{0.01}$) (c) web-BerkStan (TRI)

(d) web-BerkStan (IWC) (e) web-BerkStan (OWC)

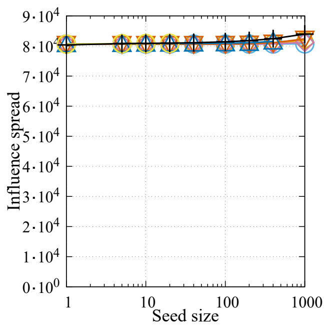
Figure 5.8: Influence spread of each algorithm for web-BerkStan network.



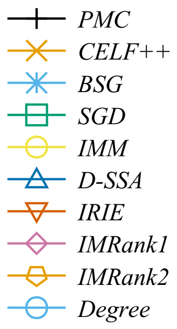
(a) higgs-twitter (UC$_{0.1}$)

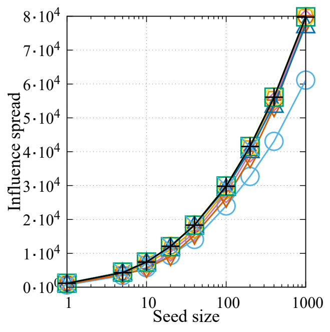


(b) higgs-twitter (UC$_{0.01}$)

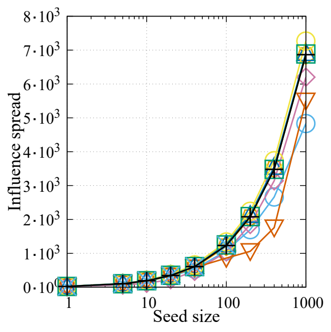


(c) higgs-twitter (TRI)



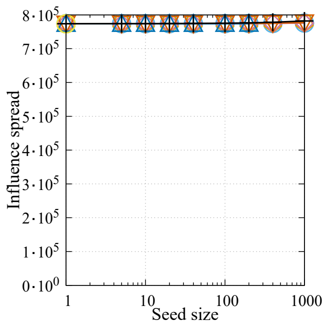


(d) higgs-twitter (IWC)

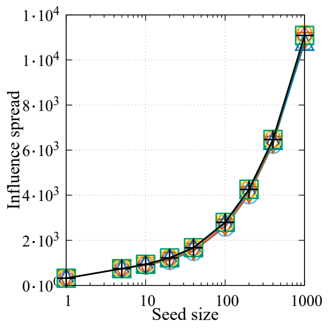


(e) higgs-twitter (OWC)

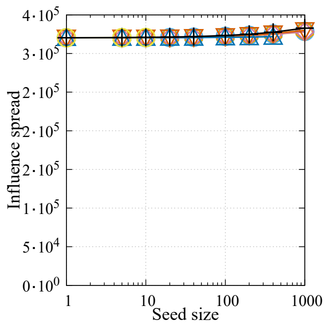
Figure 5.9: Influence spread of each algorithm for higgs-twitter network.



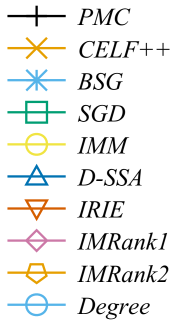
(a) soc-Pokec (UC$_{0.1}$)

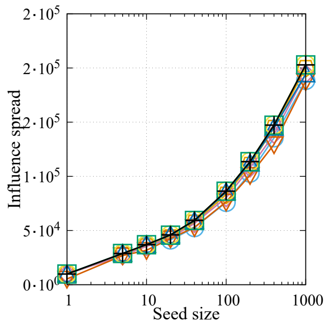


(b) soc-Pokec (UC$_{0.01}$)

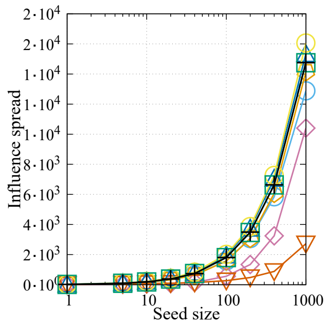


(c) soc-Pokec (TRI)





(d) soc-Pokec (IWC)



(e) soc-Pokec (OWC)

Figure 5.10: Influence spread of each algorithm for soc-Pokec network.

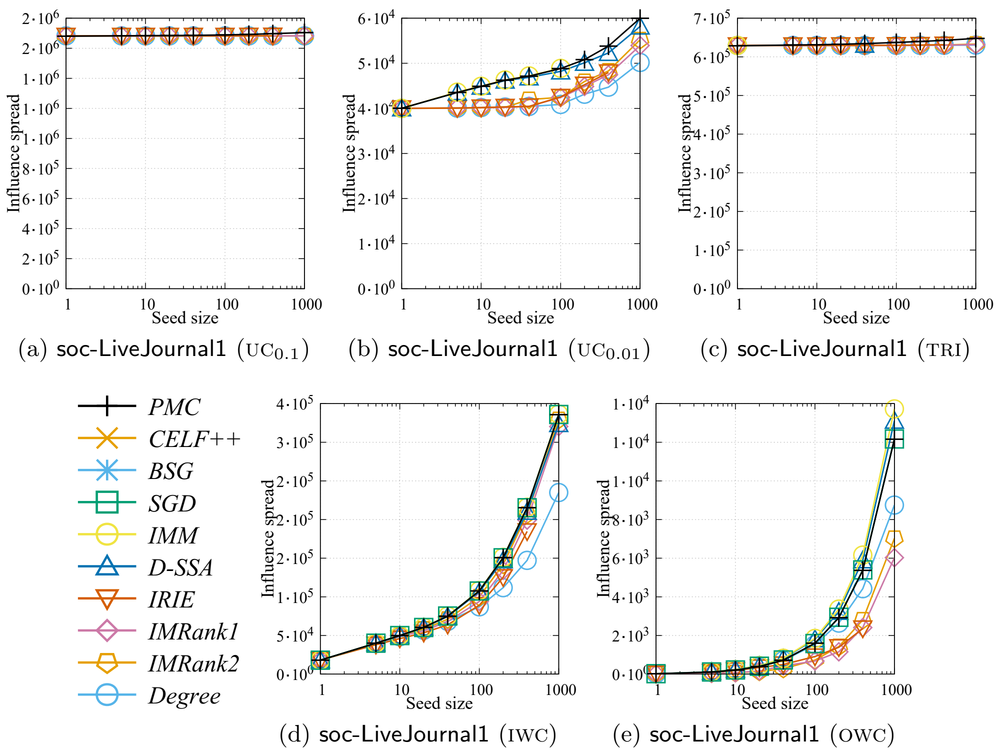


Figure 5.11: Influence spread of each algorithm for soc-LiveJournal1 network.

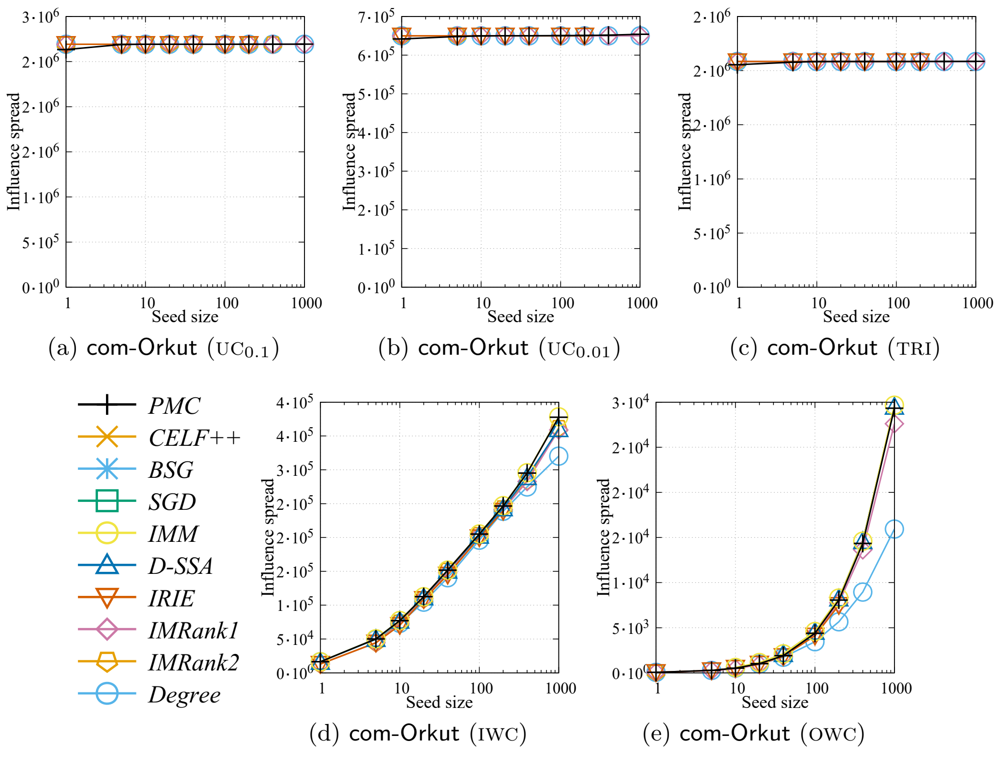


Figure 5.12: Influence spread of each algorithm for com-Orkut network.

Running time.

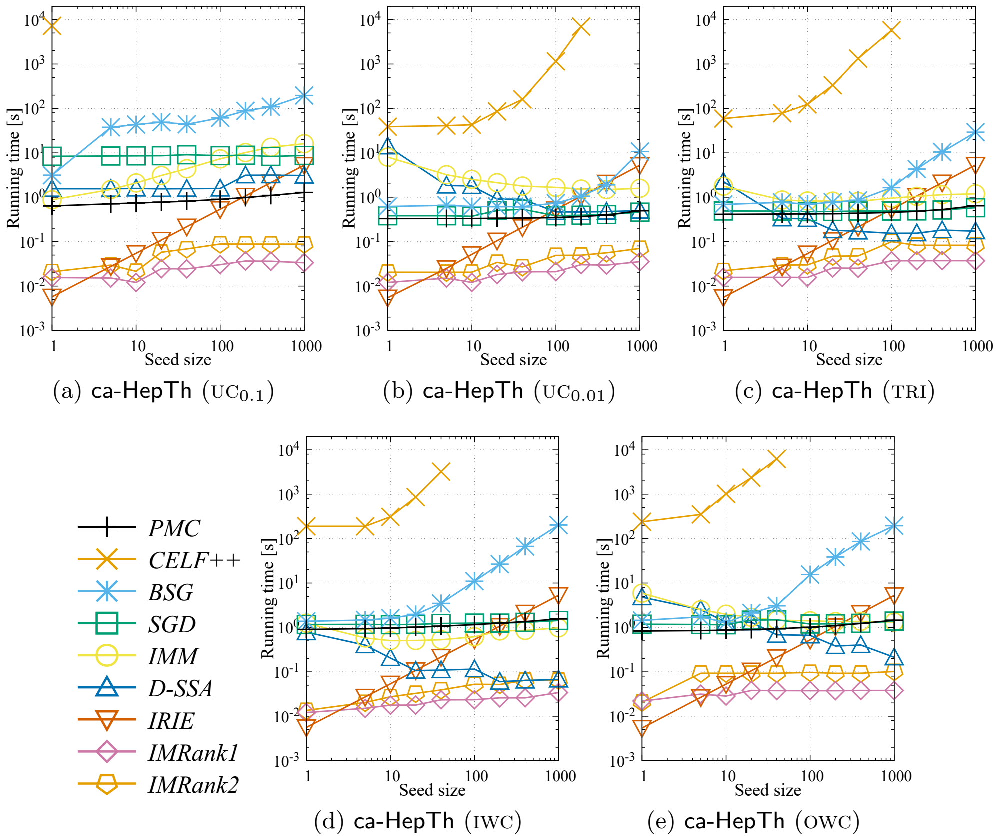
We then examine the efficiency and scalability of the algorithms. Here, we say that an algorithm "did not finish" if it did not finish in *three hours* and that it ran "out-of-memory error" if it consumed over 256GB. Figure 5.13–5.22 show the transition of the running time that each algorithm requires to compute a seed set. See Appendix B for complete experimental results. Note that the running time do not include the time for reading the input graph from secondary storage. We omit results for *Degree* because it always finished within one second.

We first examine the scalability trend of *PMC*. *PMC* finished in two hours for all configurations. It is worth to mention that *PMC* requires approximately 6,000 seconds even for com-Orkut network with hundreds of millions of edges. It also works robustly against the setting of influence probability. For a single seed selection, web-BerkStan yields the largest difference in running time among the probability settings; *PMC* requires 34.6 seconds on web-BerkStan ($\textsc{exp}_{0.01}$) while 155.6 seconds on web-BerkStan ($\textsc{exp}_{0.1}$). Moreover, *PMC* shows at most a tenfold increase in the running time from $k = 1$ to $k = 1,000$, and the largest increase was obtained on com-Orkut ($\textsc{uc}_{0.01}$).

Now, we compare *PMC* with each algorithm. We first examine simulation-based algorithms. *CELF++* was the slowest among the algorithms and did not finish all the networks except for the smallest five networks. *StaticGreedy* took longer time by several tens to several hundreds of times compared to *PMC* and did not finish even for medium-sized graphs at $k = 1$, e.g., ego-Twitter ($\textsc{uc}_{0.1}$). This is because *StaticGreedy* naively solves descendant counting for each random graph. The running time of *StaticGreedyDU* is not affected by k and almost always shorter than *StaticGreedy* for $k \geq 10$, However, *StaticGreedyDU* shows sensitive performance to the probability setting, e.g., it required approximately 4,600 seconds (900 times longer than *PMC*) on soc-Epinions1 ($\textsc{uc}_{0.1}$) while only 5.6 seconds on soc-Epinions1 ($\textsc{uc}_{0.01}$). Also, it ran out-of-memory error on soc-Slashdot0922 ($\textsc{uc}_{0.1}$). These are because *StaticGreedyDU* naively stores the reachable sets, which consume vast amounts of time and space.

We then proceed to RIS-based algorithms. Both *D-SSA* and *IMM* have the sensitive performance to the probability setting. Under $\textsc{iwc}$, they constantly terminate more quickly than *PMC*. On the other hand, they were a few times slower than *PMC* for most configurations under $\textsc{owc}$ and ran out-of-memory on soc-LiveJournal1 and com-Orkut at $k = 1$. Moreover, for unweighted settings, we confirm severe efficiency degradation. In particular, *IMM* and *D-SSA* were several tens of times slower than *PMC* for the largest four graphs (higgs-twitter, soc-Pokec, soc-LiveJournal1, and com-Orkut) except for soc-Pokec ($\textsc{uc}_{0.01}$), and running them on soc-LiveJournal1 ($\textsc{uc}_{0.1}$), com-Orkut ($\textsc{uc}_{0.1}$), and com-Orkut ($\textsc{tri}$) resulted in out-of-memory. We now explain why these algorithms ran out-of-memory. Recall that both algorithms generate a pre-specified number of RR sets as a first step and then iteratively double the number of RR sets. However, Chapter 4 has shown that the average size can be extremely large under unweighted setting because of the existence of the GC. Hence, even the first step ran out-of-memory error. On the other hand, both algorithms were able to handle web graphs and soc-Pokec ($\textsc{uc}_{0.01}$) in which the GC is tiny or not present.

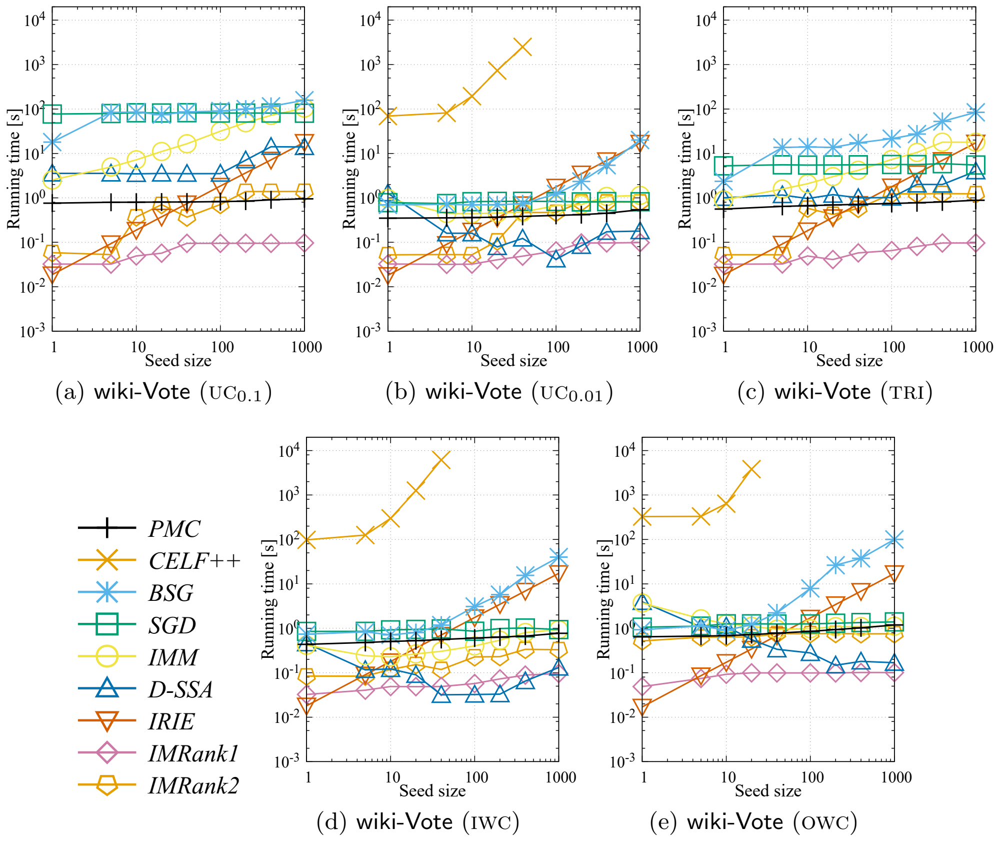
Finally, we evaluate heuristic algorithms. *IMRank* ($\ell = 1$) almost always ran faster than *PMC*. *IMRank* ($\ell = 2$) showed unstable performance against k and did not finish on com-Orkut with some configurations. The running time of *IRIE* is clearly proportional to k. Compared to *PMC*, *IRIE* required a longer time if $k \geq 100$, while a shorter time if $k \leq 100$.



(a) ca-HepTh (UC$_{0.1}$)   (b) ca-HepTh (UC$_{0.01}$)   (c) ca-HepTh (TRI)

(d) ca-HepTh (IWC)   (e) ca-HepTh (OWC)

Figure 5.13: Running time of each algorithm for ca-HepTh network.



(a) wiki-Vote (UC$_{0.1}$)   (b) wiki-Vote (UC$_{0.01}$)   (c) wiki-Vote (TRI)

(d) wiki-Vote (IWC)   (e) wiki-Vote (OWC)

Figure 5.14: Running time of each algorithm for wiki-Vote network.

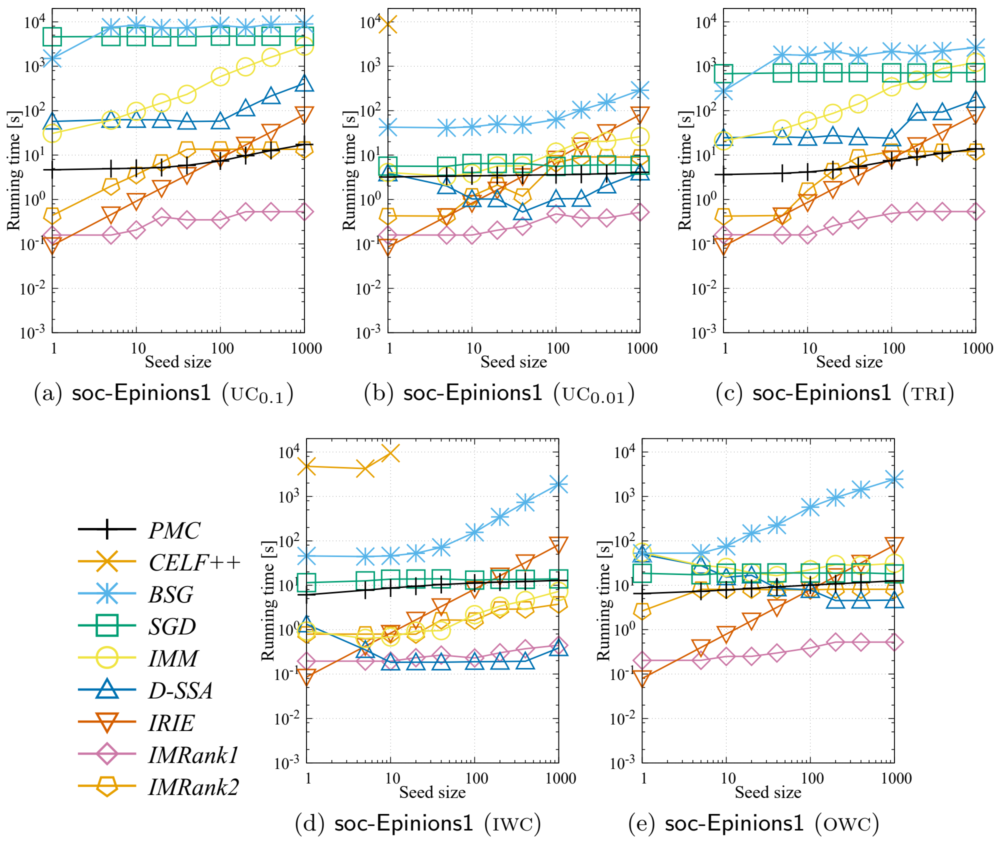


Figure 5.15: Running time of each algorithm for soc-Epinions1 network.

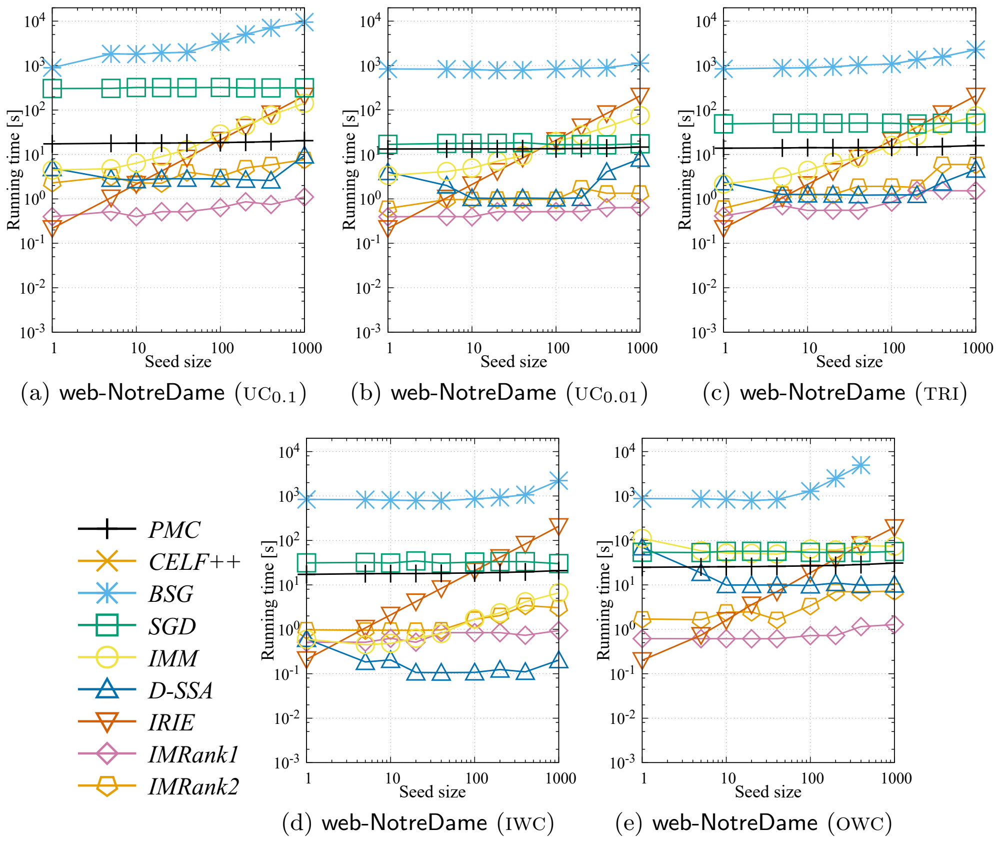
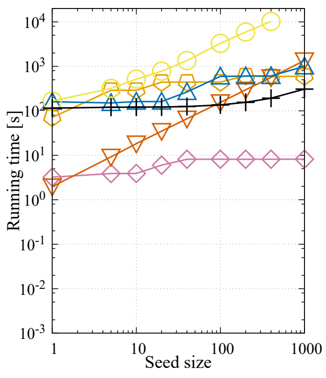
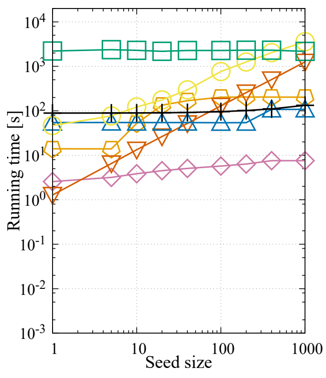


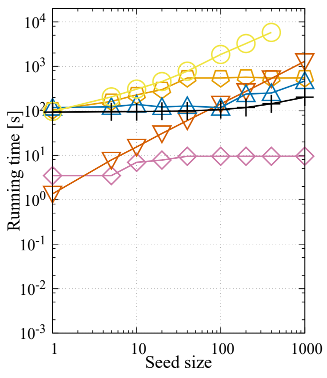
Figure 5.16: Running time of each algorithm for web-NotreDame network.



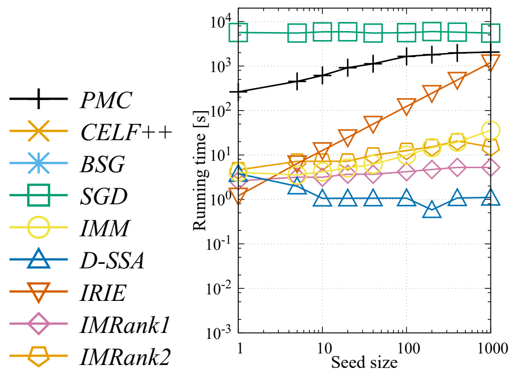
(a) wiki-Talk (UC$_{0.1}$)



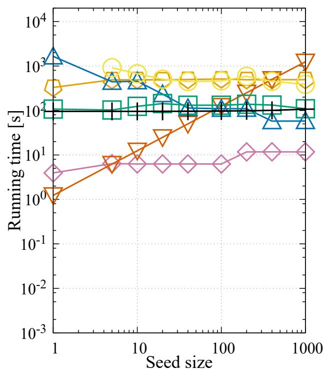
(b) wiki-Talk (UC$_{0.01}$)



(c) wiki-Talk (TRI)

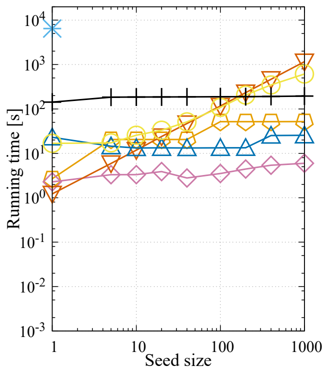


(d) wiki-Talk (IWC)

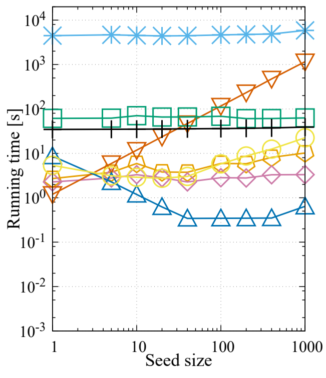


(e) wiki-Talk (OWC)

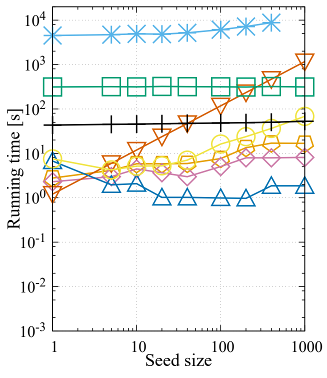
Figure 5.17: Running time of each algorithm for wiki-Talk network.



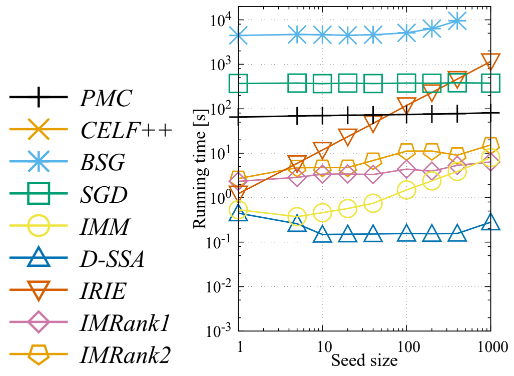
(a) web-BerkStan (UC$_{0.1}$)



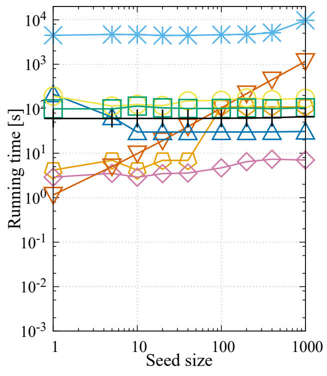
(b) web-BerkStan (UC$_{0.01}$)



(c) web-BerkStan (TRI)

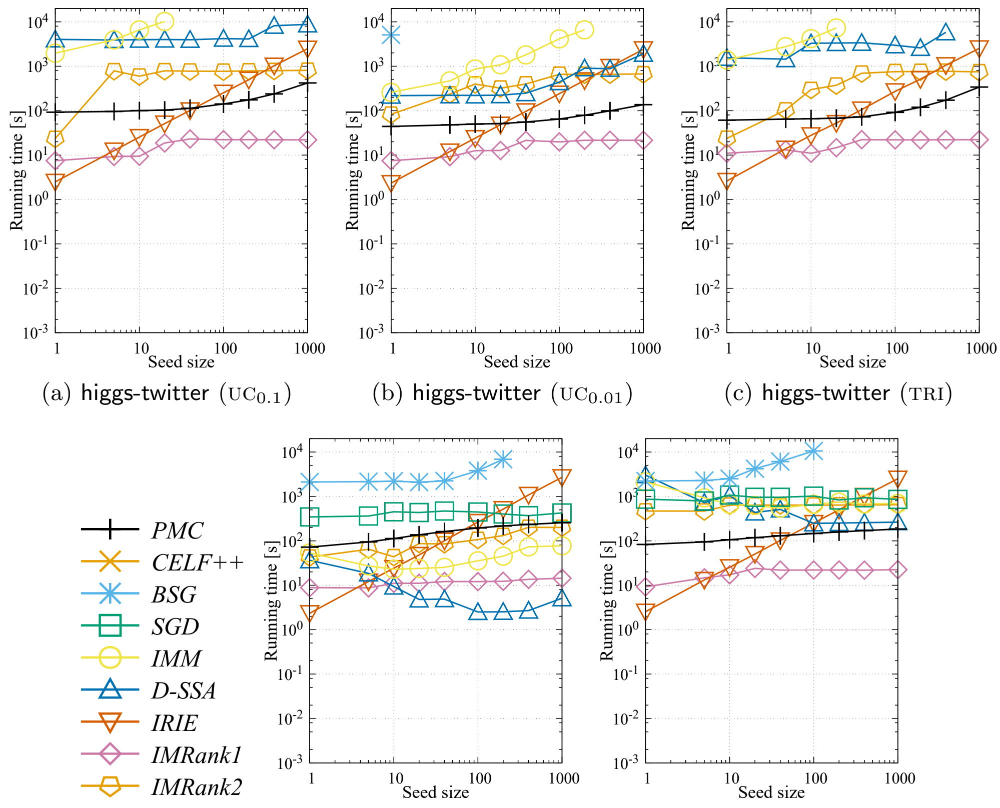


(d) web-BerkStan (IWC)



(e) web-BerkStan (OWC)

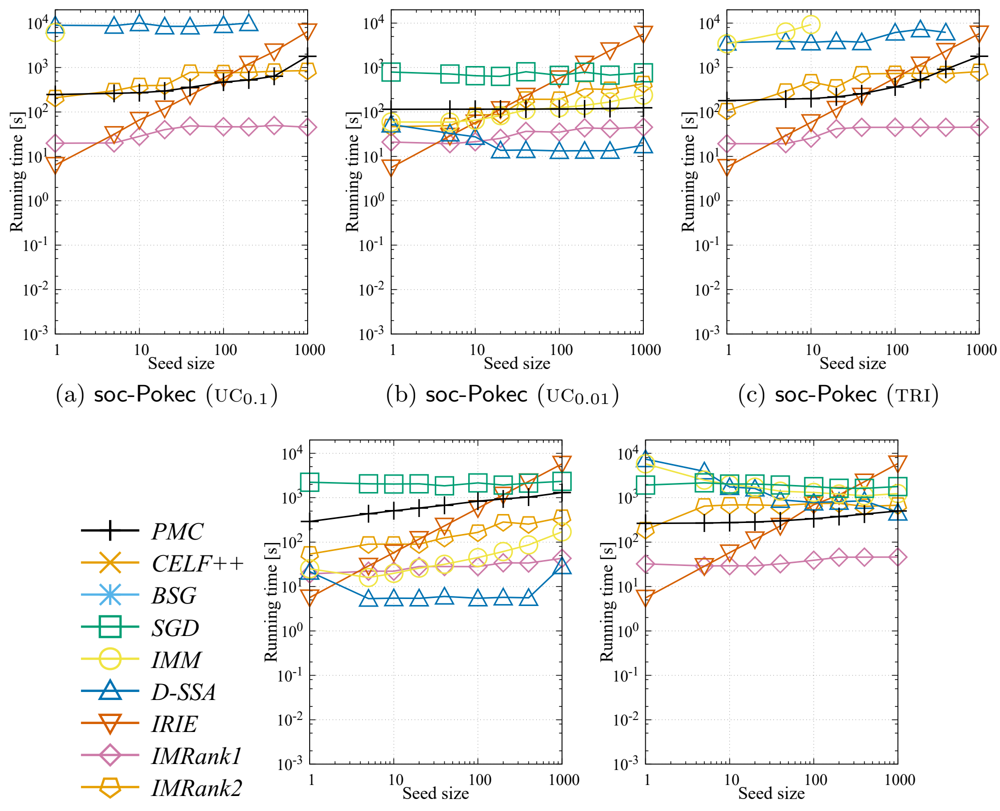
Figure 5.18: Running time of each algorithm for web-BerkStan network.



(a) higgs-twitter (UC$_{0.1}$) (b) higgs-twitter (UC$_{0.01}$) (c) higgs-twitter (TRI)

(d) higgs-twitter (IWC) (e) higgs-twitter (OWC)

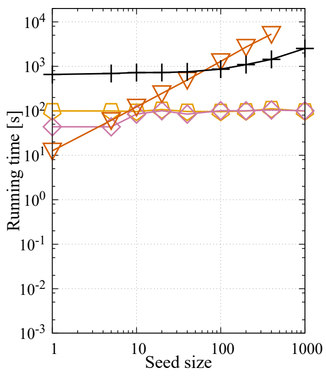
Figure 5.19: Running time of each algorithm for higgs-twitter network.



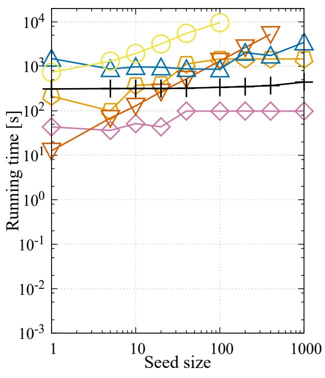
(a) soc-Pokec (UC$_{0.1}$) (b) soc-Pokec (UC$_{0.01}$) (c) soc-Pokec (TRI)

(d) soc-Pokec (IWC) (e) soc-Pokec (OWC)

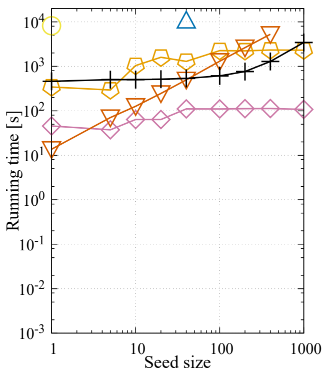
Figure 5.20: Running time of each algorithm for soc-Pokec network.



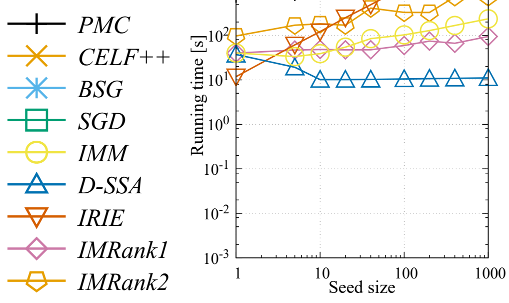
(a) soc-LiveJournal1 ($\text{UC}_{0.1}$)



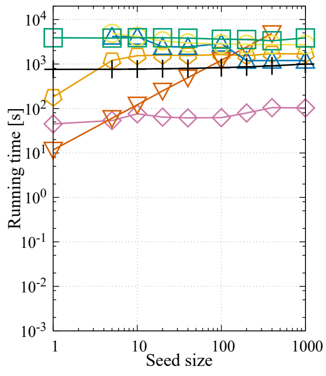
(b) soc-LiveJournal1 ($\text{UC}_{0.01}$)



(c) soc-LiveJournal1 (TRI)

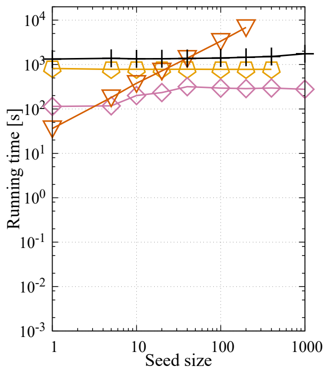


(d) soc-LiveJournal1 (IWC)

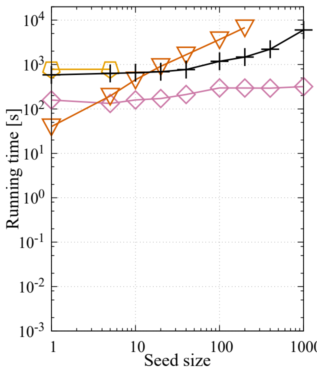


(e) soc-LiveJournal1 (OWC)

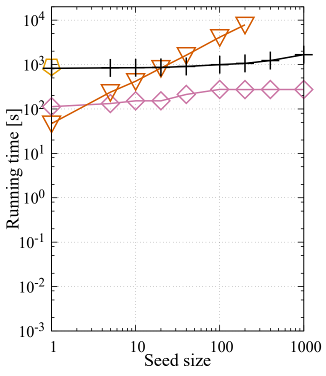
Figure 5.21: Running time of each algorithm for soc-LiveJournal1 network.



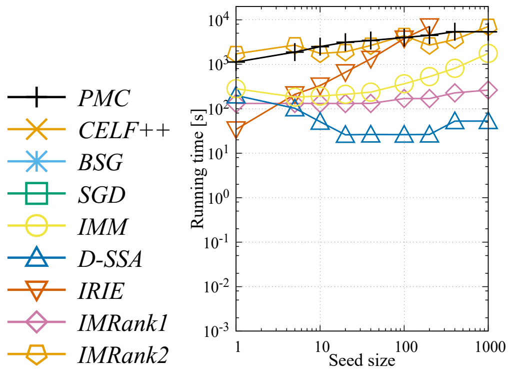
(a) com-Orkut ($\text{UC}_{0.1}$)



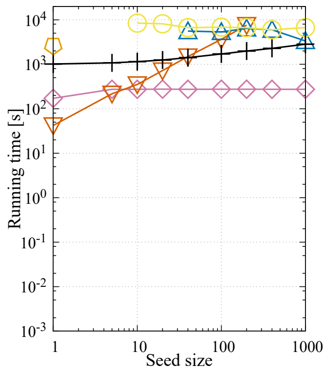
(b) com-Orkut ($\text{UC}_{0.01}$)



(c) com-Orkut (TRI)



(d) com-Orkut (IWC)



(e) com-Orkut (OWC)

Figure 5.22: Running time of each algorithm for com-Orkut network.

5.5.3 Summary

We summarize the experimental comparison. First of all, simulation-based algorithms (*CELF++*, *StaticGreedy*, and *StaticGreedyDU*) ran slower than *PMC* for almost all configurations and did not finish or ran out-of-memory error even for medium-sized networks under unweighted settings. In the following, for each influence probability setting, we compare the proposed algorithm *PMC* with RIS-based algorithms and heuristics.

High unweighted probability ($\textsc{uc}_{0.1}$ and $\textsc{exp}_{0.1}$). Overall, we did observe no large difference in the influence spread among the algorithms. This is due to the existence of the GC in random graphs. Identifying the most influential vertex, which seems to be included in the GC, is sufficient to obtain a large influence spread. Even *Degree* heuristic was comparable to *PMC* for some configurations and it was the fastest. RIS-based algorithms provided comparable solutions to *PMC*, but they were not able to finish for the largest three networks as explained so far. Hence, either *PMC* or heuristic algorithms is promising for this setting.

Low unweighted probability ($\textsc{uc}_{0.01}$ and $\textsc{exp}_{0.01}$). Finding influential vertices is not as easy as the above setting. Heuristic algorithms were no longer able to find high-quality solutions. RIS-based algorithms provided high-quality solutions to a larger extent. However, they exhibit unstable performance; when the average cascade size is large, both *IMM* and *D-SSA* ran orders of magnitude slower than *PMC*. On the other hand, *PMC* demonstrated stable and consistent scalability and provided comparable-quality solutions to *IMM* and *D-SSA*. Therefore, *PMC* is a promising algorithm for this setting.

Trivalency ($\textsc{tri}$). This setting resembles the trend of either high probability settings or low probability settings. Hence, either *PMC* or fast heuristics such as *Degree* is promising.

In-degree weighted ($\textsc{iwc}$). Heuristic algorithms were not reliable anymore since high-degree vertices may not be influential. RIS-based algorithms ran always quickly. More specifically, *D-SSA* is faster than *IMM* at the expense of quality. *PMC* produced comparable seed sets to them but ran slower than them in general. Hence, either of RIS-based algorithms is the best choice.

Out-degree weighted ($\textsc{owc}$). Heuristic algorithms failed to detect influential vertices at all. RIS-based algorithms required a longer time for smaller k. *IMM* was slower than *PMC* and did not finish on large networks for small k; however, it consistently provided the most influential seed sets. *D-SSA* ran faster than *IMM*, though its solutions were less influential than *IMM*. *PMC* was consistently faster than *IMM* and *D-SSA*, though its solutions were sometimes less influential than *IMM*. Thus, we are required to carefully select either RIS-based algorithms or *PMC* according to the trade-off between quality and efficiency.

5.5.4 Analyzing of the Proposed Techniques

Now, let us investigate the effectiveness of our two speed-up techniques, i.e., pruned BFS and BFS avoidance. We first examine the running time of snapshot-based algorithms without our techniques. Then, we analyze the effectiveness of each of the two techniques on cost reduction of BFSes.

Running time without the proposed techniques.

We first compare the efficiency of algorithms without the proposed speed-up techniques. We have three variants: one with pruned BFS (Section 5.2), one with BFS avoidance (Section 5.3), and one with no techniques. We run each algorithm in addition to our algorithm with $r = 10$ random graphs and seed size $k = 1,000$.

Table 5.1 shows the running time of each algorithm for larger eight networks. Variants without either of two techniques run several dozen times slower than our algorithm. Incorporating pruned BFS achieves up to 90 times speed-up on soc-LiveJournal1 ($\textsc{uc}_{0.1}$). The BFS-avoiding technique makes the naive algorithm over 100 times faster on soc-LiveJournal1 (IWC and OWC). Pruned BFS is apparently effective on large networks with $\textsc{uc}_{0.1}$ and TRI while less effective on IWC and OWC. On the other hand, BFS avoidance is effective on IWC and OWC. Incorporating both the techniques enhances the robust performance against various settings of influence probability.

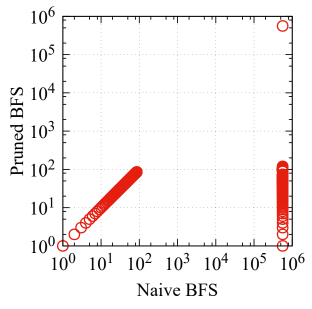
Table 5.1: The running time in second to compute a seed set of size 1,000 for each variant of the proposed method with $r = 10$.

network	**model**	PMC	BFS avoidance	pruned BFS	none
com-Youtube	UC$_{0.1}$	16.3 s	147.0 s	415.1 s	572.0 s
	UC$_{0.01}$	7.8 s	8.7 s	235.3 s	238.1 s
	TRI	13.7 s	43.1 s	316.1 s	334.8 s
	IWC	17.7 s	18.5 s	948.4 s	968.1 s
	OWC	12.1 s	12.1 s	789.9 s	790.4 s
wiki-Talk	UC$_{0.1}$	22.4 s	92.2 s	676.2 s	634.9 s
	UC$_{0.01}$	15.3 s	17.4 s	343.7 s	351.6 s
	TRI	18.2 s	38.1 s	462.9 s	474.5 s
	IWC	89.2 s	90.6 s	981.6 s	1,023.7 s
	OWC	14.6 s	13.9 s	480.6 s	552.7 s
web-Google	UC$_{0.1}$	9.7 s	10.0 s	584.2 s	659.1 s
	UC$_{0.01}$	5.3 s	5.2 s	240.4 s	246.5 s
	TRI	6.4 s	6.4 s	418.3 s	375.2 s
	IWC	10.0 s	9.8 s	838.5 s	814.4 s
	OWC	8.8 s	8.8 s	977.0 s	862.6 s
web-BerkStan	UC$_{0.1}$	12.1 s	16.7 s	261.1 s	259.7 s
	UC$_{0.01}$	4.7 s	4.8 s	138.4 s	128.8 s
	TRI	5.5 s	5.6 s	205.9 s	210.6 s
	IWC	6.9 s	6.8 s	371.7 s	356.1 s
	OWC	6.2 s	6.3 s	533.9 s	542.2 s
higgs-twitter	UC$_{0.1}$	19.1 s	144.6 s	154.4 s	281.6 s
	UC$_{0.01}$	7.0 s	20.5 s	192.3 s	201.8 s
	TRI	18.1 s	146.4 s	187.4 s	326.6 s
	IWC	12.8 s	13.0 s	551.7 s	557.2 s
	OWC	11.3 s	10.8 s	533.8 s	538.2 s
soc-Pokec	UC$_{0.1}$	80.3 s	2,158.1 s	715.8 s	2,899.8 s
	UC$_{0.01}$	13.6 s	13.6 s	953.9 s	777.7 s
	TRI	84.6 s	2,325.9 s	1,031.1 s	3,259.6 s
	IWC	61.9 s	59.1 s	3,406.3 s	3,513.2 s
	OWC	36.1 s	35.3 s	3,195.8 s	3,519.2 s
soc-LiveJournal1	UC$_{0.1}$	138.8 s	12,606.4 s	2,531.5 s	15,619.7 s
	UC$_{0.01}$	41.3 s	110.9 s	2,084.2 s	1,984.6 s
	TRI	146.1 s	6,641.7 s	2,775.7 s	9,547.3 s
	IWC	127.6 s	125.2 s	12,654.3 s	14,127.2 s
	OWC	77.3 s	78.5 s	9,989.1 s	11,964.8 s
com-Orkut	UC$_{0.1}$	118.8 s	413.1 s	638.9 s	950.7 s
	UC$_{0.01}$	177.7 s	8,696.6 s	2,643.0 s	11,271.4 s
	TRI	128.2 s	4,721.0 s	1,066.2 s	5,853.0 s
	IWC	273.1 s	280.9 s	9,108.6 s	9,653.8 s
	OWC	157.0 s	160.3 s	9,204.4 s	10,783.0 s

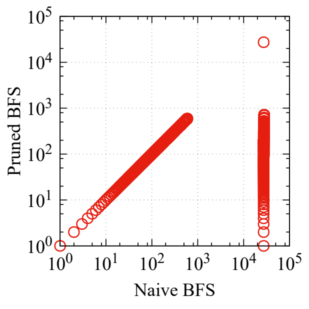
Effectiveness of pruned BFS.

We show the effectiveness of pruned BFS. Let us start with the computation cost of the first iteration in snapshot-based algorithms, i.e., the total cost of BFSes required to solve descendant counting. To this end, we sample a random graph G from $\mathcal{G}$ and compute $\mathsf{r}_G(v)$ for every vertex v using naive and pruned BFSes. Table 5.2 shows the average number of vertices visited during naive BFSes and pruned BFSes. While each naive BFS consumes a significant amount of costs under unweighted settings as investigated in Section 4.3, pruned BFS reduces this by several orders of magnitude, e.g., the cost ratio of pruned BFS to naive BFS is 0.0033 % for soc-Pokec ($\textsc{uc}_{0.1}$), 0.0017 % for soc-LiveJournal1 ($\textsc{uc}_{0.1}$), and 0.0014 % for com-Orkut ($\textsc{tri}$). However, pruned BFS has almost no effect under $\textsc{iwc}$ and $\textsc{owc}$ because there is no such GC that dominates the total amount of BFS costs.

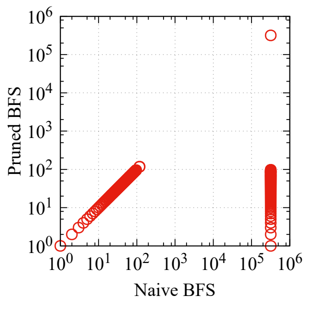
Figure 5.23 illustrates distributions of the number of visited vertices for each BFS, where each point corresponds to a vertex, and the x and y coordinates represent the number of vertices that a naive BFS and a pruned BFS starting from the vertex visits, respectively. We can see several trends from these figures. A group of points on the diagonal line (i.e., $x = y$) corresponds to vertices that cannot reach the hub. A group of points below diagonal line corresponds to vertices that can reach the hub, and hence pruning has occurred. Note that an isolated point in the upper right corresponds to the hub vertex itself. From an ancestor of the hub vertex on soc-LiveJournal1 ($\textsc{uc}_{0.1}$), each pruned BFS scans at most only 120 vertices though each naive BFS scans at least 560,000 vertices. Meanwhile, the effectiveness is weakened when using $\textsc{uc}_{0.01}$; we visit at most 721 vertices during pruned BFSes while at most 28,000 vertices during naive BFSes. Notice that if we use lower influence probabilities, e.g., $\textsc{uc}_{0.001}$, naive BFSes quickly finish though pruned BFS does not work anymore.



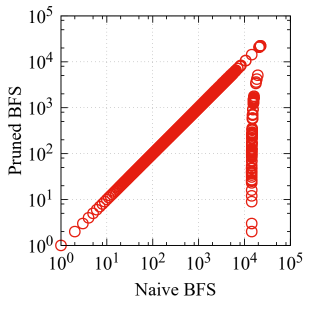
(a) soc-LiveJournal1 ($\textsc{uc}_{0.1}$)



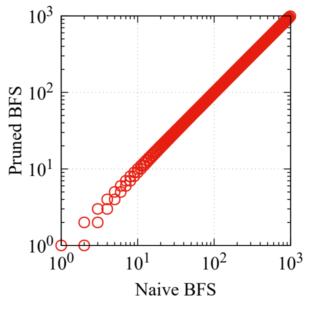
(b) soc-LiveJournal1 ($\textsc{uc}_{0.01}$)



(c) soc-LiveJournal1 ($\textsc{tri}$)



(d) soc-LiveJournal1 ($\textsc{iwc}$)



(e) soc-LiveJournal1 ($\textsc{owc}$)

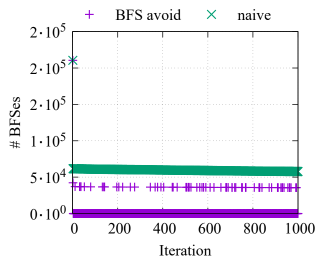
Figure 5.23: Distributions of the number of vertices visited during each BFS.

Table 5.2: The average number of visited vertices for solving the descendant counting problem.

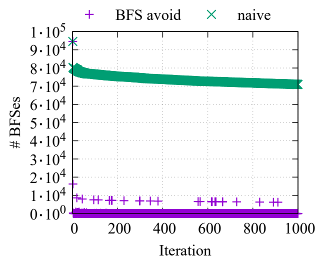
		average number of visited vertices		
network	**model**	naive BFS	pruned BFS	$\frac{\text{pruned BFS}}{\text{naive BFS}}$ %
com-Youtube	UC$_{0.1}$	1.29	9,075.26	0.0142 %
	UC$_{0.01}$	1.19	46.11	2.6 %
	TRI	1.19	1,618.45	0.0737 %
	IWC	4.94	4.94	100.0 %
	OWC	5.85	5.86	99.8 %
wiki-Talk	UC$_{0.1}$	1.15	1,756.78	0.0657 %
	UC$_{0.01}$	1.04	22.95	4.5 %
	TRI	1.07	370.41	0.2893 %
	IWC	12.35	12.35	100.0 %
	OWC	1.15	1.15	99.9 %
web-Google	UC$_{0.1}$	7.56	8.02	94.3 %
	UC$_{0.01}$	1.07	1.07	100.0 %
	TRI	1.39	1.39	100.0 %
	IWC	3.39	3.39	99.9 %
	OWC	3.39	3.39	100.0 %
web-BerkStan	UC$_{0.1}$	88.22	99.08	89.0 %
	UC$_{0.01}$	1.25	1.25	100.0 %
	TRI	2.48	2.49	99.4 %
	IWC	4.20	4.20	100.0 %
	OWC	3.66	3.66	100.0 %
higgs-twitter	UC$_{0.1}$	1.33	10,390.30	0.0128 %
	UC$_{0.01}$	2.06	321.29	0.6413 %
	TRI	1.46	6,471.26	0.0226 %
	IWC	10.80	10.80	100.0 %
	OWC	11.58	11.83	97.9 %
soc-Pokec	UC$_{0.1}$	1.53	46,528.40	0.0033 %
	UC$_{0.01}$	1.44	1.56	92.0 %
	TRI	1.84	20,144.80	0.0091 %
	IWC	15.90	15.90	100.0 %
	OWC	13.80	13.83	99.7 %
soc-LiveJournal1	UC$_{0.1}$	1.46	87,371.20	0.0017 %
	UC$_{0.01}$	1.59	207.46	0.7657 %
	TRI	1.60	24,708.30	0.0065 %
	IWC	10.58	10.93	96.8 %
	OWC	9.20	9.20	100.0 %
com-Orkut	UC$_{0.1}$	1.38	57,978.90	0.0024 %
	UC$_{0.01}$	2.25	43,103.60	0.0052 %
	TRI	1.53	112,618.00	0.0014 %
	IWC	35.86	35.87	100.0 %
	OWC	48.47	48.48	100.0 %

Effectiveness of BFS avoidance.

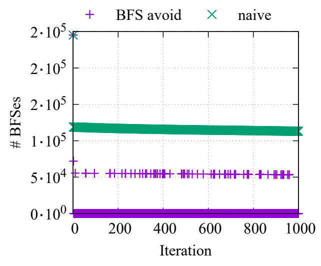
We finally discuss the effectiveness of the BFS-avoidance technique. Table 5.3 shows the total number of BFSes conducted in a certain DAG during $k = 1,000$ iterations. The reduction of the number of BFSes by our technique is 85.6–99.8%. Therefore, it turns out that a large portion of $\mathsf{r}(\cdot)$'s do not change during seed selection. Figure 5.24 shows the transition of the number of BFSes conducted in a certain DAG for each iteration. Note that the number of BFSes decreases even without our technique because the DAG shrinks over iterations; however, it is less helpful. The reduction by our technique is impressive; only a small number of BFSes were required in most iterations after the first iteration.



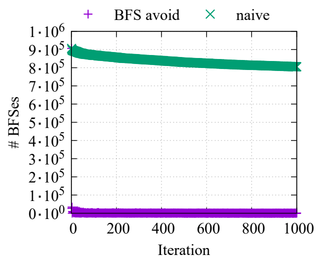
(a) soc-LiveJournal1 ($\text{UC}_{0.1}$)



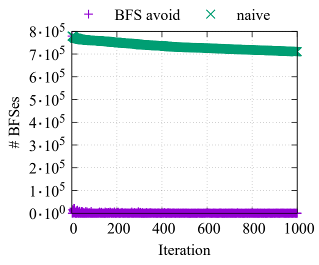
(b) soc-LiveJournal1 ($\text{UC}_{0.01}$)



(c) soc-LiveJournal1 (TRI)



(d) soc-LiveJournal1 (IWC)



(e) soc-LiveJournal1 (OWC)

Figure 5.24: Transitions of the number of BFSes for each iteration.

Table 5.3: The total number of BFSes performed when $k = 1,000$.

		total number of BFSes in 1,000 iterations		
network	**model**	naive	BFS avoidance	$\frac{\text{BFS avoidance}}{\text{naive}}$ %
com-Youtube	UC$_{0.1}$	245,903	9,394,239	2.62 %
	UC$_{0.01}$	36,679	2,977,448	1.23 %
	TRI	156,038	5,742,274	2.72 %
	IWC	221,285	93,457,266	0.24 %
	OWC	174,213	62,791,614	0.28 %
wiki-Talk	UC$_{0.1}$	166,279	1,906,348	8.72 %
	UC$_{0.01}$	75,405	2,243,130	3.36 %
	TRI	171,791	2,400,999	7.15 %
	IWC	201,023	12,646,070	1.59 %
	OWC	23,778	11,664,959	0.20 %
web-Google	UC$_{0.1}$	105,477	44,599,935	0.24 %
	UC$_{0.01}$	5,365	4,583,337	0.12 %
	TRI	33,087	24,384,211	0.14 %
	IWC	125,325	68,738,337	0.18 %
	OWC	103,194	88,216,511	0.12 %
web-BerkStan	UC$_{0.1}$	70,172	19,265,366	0.36 %
	UC$_{0.01}$	10,480	5,109,137	0.21 %
	TRI	37,574	13,916,296	0.27 %
	IWC	81,086	35,345,530	0.23 %
	OWC	103,257	79,048,400	0.13 %
higgs-twitter	UC$_{0.1}$	444,862	3,303,399	13.47 %
	UC$_{0.01}$	160,570	10,050,437	1.60 %
	TRI	729,776	6,023,075	12.12 %
	IWC	225,065	52,230,808	0.43 %
	OWC	346,251	51,317,419	0.67 %
soc-Pokec	UC$_{0.1}$	2,054,351	19,634,805	10.46 %
	UC$_{0.01}$	56,776	46,019,579	0.12 %
	TRI	3,726,617	54,578,508	6.83 %
	IWC	975,567	274,014,245	0.36 %
	OWC	769,826	287,174,601	0.27 %
soc-LiveJournal1	UC$_{0.1}$	2,560,612	59,615,657	4.30 %
	UC$_{0.01}$	295,079	73,645,716	0.40 %
	TRI	4,287,233	115,534,268	3.71 %
	IWC	1,717,407	835,384,868	0.21 %
	OWC	1,368,856	733,839,019	0.19 %
com-Orkut	UC$_{0.1}$	53,465	727,733	7.35 %
	UC$_{0.01}$	5,878,463	130,286,233	4.51 %
	TRI	1,207,661	13,601,604	8.88 %
	IWC	2,756,350	586,598,513	0.47 %
	OWC	2,312,479	689,721,865	0.34 %

Chapter 6

Dynamic Indexing Algorithm for Real-time Influence Analysis

In this chapter, we propose a dynamic indexing algorithm for real-time influence analysis on evolving networks. Figure 6.1 shows an overview of our overall approach. We first design the index structure, present query algorithms for influence estimation and influence maximization, and provide update algorithms (Section 6.1). We then provide a theoretical analysis, which guarantees the non-degeneracy of the update algorithms and the solution quality of our query algorithm (Section 6.2). Further, we introduce several techniques for improving the performance of our indexing method (Section 6.3). We finally report experimental evaluations of the proposed indexing algorithm and the comparison between our index and existing static algorithms (Section 6.4).

6.1 Proposed Indexing Algorithm

In this section, we present our indexing algorithm for influence analysis in evolving influence graphs. First, we explain what we store in our index, and how it is constructed from a static influence graph. Then, we demonstrate how to answer queries of influence estimation and influence maximization using our index. Finally, we explain how the index is dynamically updated.

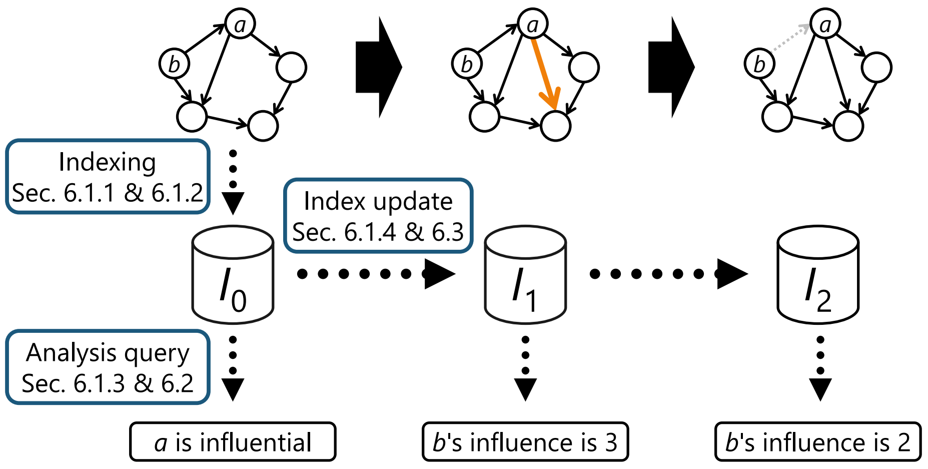
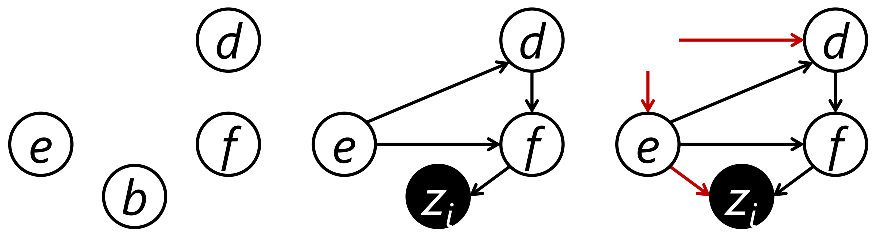


Figure 6.1: Overview of our dynamic index for real-time influence analysis in Chapter 6.



(a) RR sets (Section 3.3) (b) Our sketch structure (c) Complete information

Figure 6.2: Structural comparison among RR sets, our index, and full information. Our index is a sweet spot between memory consumption and correct updates.

6.1.1 Index Structure

Let us begin with our index structure by extending RR sets [29]. For a static influence graph $\mathcal{G} = (V, E, p)$, The proposed index consists of a set of triplets $I = \{(z_i, x_i, H_i)\}_i$, where $z_i \in V$ is a *target* vertex, $x_i : E \to [0, 1]$ is an *activation function*, and H_i is a subgraph of (V, E) consisting of live edges with respect to x_i. Recall from Section 3.3.5 that an edge (u, v) is called *live* with respect to x_i if $x_i(u, v) < p(u, v)$ and *blocked* otherwise, Here, every vertex in $V(H_i)$ is able to reach z_i in H_i. We call each (z_i, x_i, H_i) a *sketch*.

Since storing x_i requires $\mathcal{O}(|E|)$ space, we reduce the memory consumption by using a pseudorandom generator called Random123 [170]. Random123 is *counter-based*, that is, it maps an integer to a (pseudo)random value. Instead of explicitly storing $x_i(u, v)$ for each $i \in [|I|]$ and $(u, v) \in E$, for each time that we need $x_i(u, v)$, we call Random123 with an integer representation of the pair $(i, (u, v))$, and we use the obtained value after normalizing it to be in $[0, 1]$.

The number of sketches in I is determined as follows: The *weight* of a subgraph H, denoted by $w(H)$, is defined as $|V(H)| + \sum_{v \in V(H)} |\mathcal{N}^-_{\mathcal{G}}(v)|$, which is an upper bound of the space required to store H. Then, we will always keep the condition

$$\sum_{i \in [|I|-1]} w(H_i) < W \text{ and } \sum_{i \in [|I|]} w(H_i) \geq W. \tag{6.1}$$

Here, W is a parameter for tuning the trade-off between efficiency and accuracy, and we discuss how to set the value of W in Section 6.2. Note that the space complexity of our index is roughly bounded by $\mathcal{O}(W)$.

For a vertex $v \in V$, let I_v denote the set of indices $i \in [|I|]$ with $v \in V(H_i)$, and for a vertex set $S \subseteq V$, let I_S denote the set of indices $i \in [|I|]$ with $S \cap V(H_i) \neq \emptyset$, i.e., $I_S = \bigcup_{v \in S} I_v$. Our index stores I_v for every vertex v, so that we can quickly fetch the sketches that include a particular vertex.

Comparison against RIS [29, 30]. Here, we compare our sketches with RR sets and complete information (Figure 6.2). First, RR sets cannot support dynamic updates efficiently, because these correspond to $V(H_i)$'s in our language. On the other hand, Observation 4.3 tells us that storing both live and blocked edges consumes a significant amount of memory. Therefore, our index is a sweet spot.

6.1.2 Index Construction

Here, we describe how we construct our index from a static influence graph. Given an influence graph $\mathcal{G} = (V, E, p)$, we begin with an empty index $I = \emptyset$, and we

Algorithm 6.1 Influence queries.

```
1:  procedure EstimateInfluence($I, S$)
2:      if $S = \{v\}$ then
3:          return $|V| \cdot |I_v|/|I|$.
4:      else
5:          return $|V| \cdot \left|\bigcup_{v \in S} I_v\right| / |I|$.

6:  procedure MaximizeInfluence($I, k$)
7:      $S \leftarrow \emptyset$.
8:      for $i = 1$ to $k$ do
9:          $s_\ell \leftarrow \mathrm{argmax}_{v \in V \setminus S}\, d_{I-S}(S \cup \{v\})$.
10:         $S \leftarrow S \cup \{s_\ell\}$.
11:     return $S$.
```

repeatedly create new sketches and add them into I for as long as the total weight of the current index is less than W. A sketch is constructed using the following reverse-BFS-like method: First, we sample a target vertex $z_i \in V$ uniformly at random, and add it to an empty queue. If the queue is not empty, then we remove a vertex v from the queue. If this is the first time that v is visited, then for each live edge (u, v) entering v, we add u to the queue. This procedure continues until the queue gets empty. Finally, we set $V(H_i)$ to the visited vertices and $E(H_i)$ to the examined live edges. Building a single sketch (z_i, x_i, H_i) requires $\mathcal{O}(w(H_i))$ time, and thus the entire index construction requires $\mathcal{O}(W)$ time.

6.1.3 Supporting Queries

In this subsection, we will describe how we approach influence estimation and influence maximization queries using the index constructed in the previous subsection. Remark that our query algorithms are based on those of RIS and introduce techniques for boosting empirical efficiency.

Influence estimation.

We will start with influence estimation. EstimateInfluence in Algorithm 6.1 shows pseudocode of our query algorithm. In a similar way to RIS, we approximate the influence spread of a vertex set S as

$$\frac{|V| \cdot |I_S|}{|I|}. \tag{6.2}$$

Given a singleton v, we can obtain its influence estimation by computing $|V| \cdot |I_v|/|I|$ in constant time, as we have stored I_v in our index. For a vertex set S of size greater than one, by storing each $V(H_i)$ using a hash so that we can check $v \in V(H_i)$ in constant time, we can compute $|V| \cdot |I_S|/|I|$ in time $\mathcal{O}(|S| \cdot |I|)$. However, we can improve the running time by using I_v stored in our index. That is, we compute I_S as $I_S = \bigcup_{v \in S} I_v$. This technique reduces the running time to $\mathcal{O}(\sum_{v \in S} |I_v|)$, and our experimental results demonstrate it is much faster than a naive $\mathcal{O}(|S| \cdot |I|)$-time method.

Influence maximization.

We next proceed to influence maximization. MaximizeInfluence in Algorithm 6.1 shows pseudocode of our query algorithm. In fact, our algorithm is almost equivalent to the seed selection procedure in RIS-based algorithms [29],

Algorithm 6.2 Auxiliary functions.

```
1: procedure Expand($I, i, z$)
2:     $Q \leftarrow$ a queue with only one element $z$.
3:     $H_i \leftarrow H_i \cup \{z\}$.
4:     while $Q \neq \emptyset$ do
5:         dequeue $v$ from $Q$.
6:         for all $(u, v) \in E$ do
7:             if $x_i(u, v) < p(u, v)$ then
8:                 $E(H_i) \leftarrow E(H_i) \cup \{(u, v)\}$.
9:                 if $v \notin V(H_i)$ then
10:                    enqueue $u$ onto $Q$.
11:                    $V(H_i) \leftarrow V(H_i) \cup \{u\}$.

12: procedure Shrink($I, i$)
13:     $H_i \leftarrow$ the subgraph consisting of vertices that can reach $z_i$ by passing through
    live edges.

14: procedure Adjust($I$)
15:     while $\sum_{1 \leq i \leq |I|} w(H_i) < W$ do
16:         sample target vertex $z_{|I|+1}$ uniformly at random.
17:         Expand($I, |I| + 1, z_{|I|+1}$).
18:     while $\sum_{1 \leq i \leq |I|-1} w(H_i) \geq W$ do
19:         discard the last element from $I$.
```

which solves the maximum coverage on a collection of RR sets as shown in Algorithm 3.3. For the sake of completeness, we will describe that procedure in our language. For a vertex v, define the *degree* $d_I(v)$ of v in I as the number of $i \in [|I|]$ with $v \in V(H_i)$. In other words, $d_I(v) = |I_v|$ initially. We choose the vertex with the maximum degree in I and denote this by v_1. Then, examining each sketch (z_i, x_i, H_i), we remove it if $v_1 \in V(H_i)$. Next, we choose the vertex with the maximum degree in the resulting index and denote this by v_2. Then, examining each sketch (z_i, x_i, H_i), we remove it if $v_2 \in V(H_i)$. We repeat this process k times, and then output the vertex set $\{v_1, v_2, \ldots, v_k\}$.

Now, we consider the computation time. When we add a vertex v_ℓ to the output, we need to decrement the degrees of vertices $v \in V(H_i)$ for each $i \in I_v$. However, this decrementation occurs only once for each sketch. Hence, the total time complexity is $\mathcal{O}(\sum_{i \in [|I|]} |V(H_i)|)$. To boost the empirical performance, we further employ the lazy evaluation technique [140].

6.1.4 Supporting Dynamic Update Operations

In this subsection, we explain how we update our index. We consider five operations: vertex additions, vertex deletions, edge additions, edge deletions, and influence probability updates. First, we present three subroutines used in our update operations, and then we describe how we update the index when the graph changes. The details are provided in Algorithms 6.2, 6.3, and 6.4.

Auxiliary subroutines.

Expand(I, i, z). Suppose that we have added an edge (z, w) or increased the influence probability of an edge (z, w). Then, for each $i \in [|I|]$ with $w \in V(H_i)$, we want to add vertices from which we can newly reach the vertex z_i to H_i. To this end, we perform a reverse BFS from z, and add the traversed vertices to H_i.

Note that all of the newly added vertices can reach z.

Shrink(I, i)**.** Suppose that we have removed an edge (u, v) or decreased the influence probability of an edge (u, v). Then, for $i \in [|I|]$ with $v \in V(H_i)$, we want to remove the vertices in H_i from which we can no longer reach z_i anymore. To this end, we recompute the set of vertices that can reach z_i by conducting a reverse BFS from z_i.

Adjust(I)**.** While we are processing edge and vertex updates, the total weight of the index may violate the condition on the total weight (Eq. (6.1)). In such a case, we create new sketches or remove current sketches as follows. If the total weight is smaller than the threshold W, then we create a new sketch (z, x, H) by sampling $z \in V$, and calling Expand on z to make H. On the other hand, if the total weight of sketches, excluding the last one, is larger than or equal to W, then we remove the last sketch from the index.

Dynamic update routines.

Now, we explain how we update our index when the graph changes.

AddVertex(I, v)**.** Suppose that we have added a new vertex v to the current graph. In such a case, we must update the target vertices in the index to preserve the property that each vertex in the graph is chosen uniformly at random as a target vertex.

Let V and V' denote the vertex set of a graph before and after we add a new vertex v, respectively, that is, $V' = V \cup \{v\}$. Suppose that we construct an index from scratch after inserting v. Obviously, for each time we choose a sketch, the probability that the target vertex is chosen from V is $\frac{|V|}{|V|+1}$, and the probability that the target vertex is v is $\frac{1}{|V|+1}$. In order to ensure that this property holds, we update the target vertex in the current index to v with probability $\frac{1}{|V|+1}$.

DeleteVertex(I, v)**.** Suppose that we have removed a vertex v from the current graph. Then, for each $i \in [|I|]$, we check whether v is contained in H_i. If this is the case, then we update the triplet (z_i, x_i, H_i) as follows: If $z_i = v$, we sample z_i from $V \setminus \{v\}$ uniformly at random and reconstruct H_i. Otherwise, we remove v and the edges leaving or entering v from H_i, and then we call Shrink(I, i) to shrink H_i.

Change$(I, (u, v), p)$**.** Suppose that we have changed the influence probability of an edge (u, v) from p' to p. If the state of (u, v) with respect to x_i changes, then we need to update subgraphs in the index I. More specifically, we carry out the following for each $i \in [|I|]$. If $p' < x_i(u, v) \leq p$, then we expand H_i by calling Expand(I, i, u). If $p < x_i(u, v) \leq p'$, then we shrink H_i by calling Shrink(I, i).

AddEdge$(I, (u, v), p)$**.** Suppose that we have added an edge (u, v) with influence probability p to the current graph. First, we add (u, v) to the current edge set E and set $p(u, v) = 0$. Then, we update the influence probability $p(u, v)$ to p, by calling Change$(I, (u, v), p)$.

Algorithm 6.3 Vertex operations.

```
 1: procedure AddVertex($I, v$)
 2:     for $i = 1$ to $|I|$ do
 3:         continue with probability $1 - \frac{1}{|V|+1}$.
 4:         $H_i \leftarrow \emptyset$, $z_i \leftarrow v$.
 5:         Expand($I, i, z_i$).
 6:     Adjust($I$).

 7: procedure DeleteVertex($I, v$)
 8:     remove the edges leaving or entering $V$ from each $E(H_i)$.
 9:     for all $i \in I_v$ do
10:         if $z_i = v$ then
11:             $H_i \leftarrow \emptyset$.
12:             sample target vertex $z_i$ uniformly at random.
13:             Expand($I, i, z_i$).
14:         else
15:             Shrink($I, i$).
16:     Adjust($I$).
```

Algorithm 6.4 Edge operations.

```
 1: procedure Change($I, (u, v), p$)
 2:     $p(u, v) \leftarrow p$.
 3:     for all $i \in I_v$ do
 4:         $P \leftarrow [\![(u, v) \in E(H_i)]\!]$, $Q \leftarrow [\![x_i(u, v) < p]\!]$.
 5:         if $\neg P \wedge Q$ then                          ▷ blocked → live
 6:             $E(H_i) \leftarrow E(H_i) \cup \{(u, v)\}$.
 7:             Expand($I, i, u$).
 8:         if $P \wedge \neg Q$ then                          ▷ live → blocked
 9:             $E(H_i) \leftarrow E(H_i) \setminus \{(u, v)\}$.
10:             Shrink($I, i$).
11:     Adjust($I$).

12: procedure AddEdge($I, (u, v), p$)
13:     $p(u, v) \leftarrow 0$.
14:     Change($I, (u, v), p$).

15: procedure DeleteEdge($I, (u, v)$)
16:     Change($I, (u, v), 0$).
```

DeleteEdge$(I, (u, v))$. Suppose that we have deleted an edge (u, v) from the current graph. Then, we first update the influence probability $p(u, v)$ to zero, by calling Change$(I, (u, v), 0)$, and then remove (u, v) from E.

6.2 Theoretical Analysis

In this section, we will show the correctness of our indexing method and then proceed to analyze its time complexity.

6.2.1 Correctness

In this subsection, we consider the correctness of our indexing method. The proof in this subsection has been derived by Yuichi Yoshida, a co-author of the extended abstract [157] published in PVLDB 2016. In this thesis, we revise Theorems 6.9 and 6.10, for the quality of our query algorithms so that a hidden constant is clear.

Note that our method is randomized. We first define $\mathcal{I}_W^{\mathrm{sta}}(\mathcal{G})$ and $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G})$ as the distribution of indices in the case that we apply our method to a static influence graph $\mathcal{G}$ and the sequence of dynamic updates that results in $\mathcal{G}$, respectively. Our goal is to show that $\mathcal{I}_W^{\mathrm{sta}}(\mathcal{G}) = \mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G})$. If this is the case, then queries on the index following any dynamic updates will inherit the same guarantees to the ones in [30].

Given an influence graph $\mathcal{G} = (V, E, p)$, consider the following random process that generates a sequence of pairs. For each step, we sample a target vertex $z \in V$ and an activation function $x : E \to [0, 1]$ uniformly at random, and add the pair (z, x) to the sequence. Let $\mathcal{X}_\infty(\mathcal{G})$ denote the distribution of infinite sequences of pairs obtained in this way. Let $H(z, x)$ be the subgraph consisting of vertices that can reach z under x. Furthermore, let $\mathcal{I}_\infty(\mathcal{G})$ denote the distribution of infinite sequences of triplets obtained from $\mathcal{X}_\infty(\mathcal{G})$ by replacing each pair (z, x) by $(z, x, H(z, x))$.

We say that a distribution $\mathcal{X}$ of pair sequences is *valid* for $\mathcal{G}$ if it can be obtained by sampling a random sequence from $\mathcal{X}_\infty(\mathcal{G})$ uniformly at random and taking a prefix of it (of arbitrary length). Similarly, we say that a distribution $\mathcal{I}$ of triplet sequences is *valid* for $\mathcal{G}$ if it can be obtained by sampling a random sequence from $\mathcal{I}_\infty(\mathcal{G})$ uniformly at random and taking a prefix of it.

For a positive integer W, we define $\mathcal{I}_W(\mathcal{G})$ as the distribution over prefixes of triplet sequences in $\mathcal{I}_\infty(\mathcal{G})$ that are obtained as follows: We sample a triplet sequence (z_1, x_1, H_1), (z_2, x_2, H_2), $\ldots$ from $\mathcal{I}_\infty(\mathcal{G})$ and take the minimum prefix of it such that the total weight of the subgraphs is at least W. It is easy to see that $\mathcal{I}_W(\mathcal{G}) = \mathcal{I}_W^{\mathrm{sta}}(\mathcal{G})$. We will establish that $\mathcal{I}_W^{\mathrm{sta}}(\mathcal{G}) = \mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G})$ by showing that $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G}) = \mathcal{I}_W(\mathcal{G})$.

For the empty influence graph $\mathcal{G} = (\emptyset, \emptyset, p)$, we clearly have $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G}) = \mathcal{I}_W(\mathcal{G})$. We will show that for any influence graph $\mathcal{G}$ with $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G}) = \mathcal{I}_W(\mathcal{G})$ and an influence graph $\mathcal{G}'$ obtained from $\mathcal{G}$ by a dynamic update, we again have $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G}') = \mathcal{I}_W(\mathcal{G}')$. Then, we are done by induction on the number of updates.

The following auxiliary lemma states that we can change the length of a valid distribution to W using $\textsc{Adjust}(\cdot)$.

Lemma 6.1. *Let $\mathcal{G}$ be an influence graph, and let $\mathcal{I}$ be a valid distribution of finite pair sequences for $\mathcal{G}$. Then, $\mathcal{I}' = \textsc{Adjust}(\mathcal{I})$ is equal to $\mathcal{I}_W(\mathcal{G})$, where W is a parameter used in* $\textsc{Adjust}$.

Remark 6.2. *By $\textsc{Adjust}(\mathcal{I})$, we mean the distribution of sequences obtained by applying* $\textsc{Adjust}$ *to a sequence I sampled from $\mathcal{I}$. We will use similar conventions for other procedures in the following.*

Proof. We obtain $\mathcal{I}_W(\mathcal{G})$ from $\mathcal{I}$ as follows. Let $I = (z_1, x_1, H_1), \ldots, (z_{|I|}, x_{|I|}, H_{|I|})$ be a triplet sequence sampled from $\mathcal{I}$. Then, we repeat the following process. If $\sum_{1 \leq i \leq |I|} w(H_i) < W$, then we sample a target vertex z and an activation function $x : E \to [0, 1]$ uniformly at random. Then, we compute the corresponding subgraph H, and add the triplet (z, x, H) to I. If $\sum_{1 \leq i \leq |I|-1} w(H_i) \geq W$, then we remove the last element from I. This exactly corresponds to what is carried out in $\textsc{Adjust}$, and it follows that $\mathcal{I}' = \mathcal{I}_W(\mathcal{G})$. $\square$

In the following, we will show that our update routines in Algorithms 6.3 and 6.4 transform a valid distribution for the original influence graph to a valid distribution for the new influence graph.

Lemma 6.3. *Let $\mathcal{G}$ be an influence graph and $\mathcal{G}'$ be the influence graph obtained from $\mathcal{G}$ by changing the influence probability of an edge $e \in E(\mathcal{G})$ to $p \in (0,1]$). If $\mathcal{I}$ is a valid distribution of triplet sequences for $\mathcal{G}$, then the distribution $\mathcal{I}' =$ $\textsc{Change}(\mathcal{I}, (u,v), p)$ is a valid distribution of triplet sequences for $\mathcal{G}'$.*

Proof. Let $\mathcal{X}$ and $\mathcal{X}'$ be the distributions of the pair sequences corresponding to $\mathcal{I}$ and $\mathcal{I}'$, respectively. Then, it is clear that $\mathcal{X} = \mathcal{X}'$. Because $\mathcal{X}$ is valid for $\mathcal{G}$, it follows that $\mathcal{X}'$ is also valid for $\mathcal{G}'$. Hence, $\mathcal{I}'$ is valid for $\mathcal{G}'$. □

Lemma 6.4. *Let $\mathcal{G}$ be an influence graph and $\mathcal{G}'$ be the influence graph obtained from $\mathcal{G}$ by adding a new vertex $v \notin V(\mathcal{G})$. If $\mathcal{I}$ is a valid distribution for $\mathcal{G}$, then $\mathcal{I}' = \textsc{AddVertex}(\mathcal{I}, v)$ is a valid distribution for $\mathcal{G}'$.*

Proof. Let $\mathcal{X}$ and $\mathcal{X}'$ be the distributions of the pair sequences corresponding to $\mathcal{I}$ and $\mathcal{I}'$, respectively. Then, we can obtain $\mathcal{X}'$ from $\mathcal{X}$ as follows. Let $(z_1, x_1), (z_2, x_2), \ldots$ be a sequence sampled from $\mathcal{X}$. Then, we replace each of z_i by v with probability $1 - 1/|V(\mathcal{G}')|$. We can observe that $\mathcal{X}'$ is valid for $\mathcal{G}'$, and it follows that $\mathcal{I}'$ is valid for $\mathcal{G}'$. □

Lemma 6.5. *Let $\mathcal{G}$ be an influence graph and $\mathcal{G}'$ be the influence graph obtained from $\mathcal{G}$ by removing a vertex $v \in V(\mathcal{G})$. If $\mathcal{I}$ is a valid distribution for $\mathcal{G}$, then $\mathcal{I}' = \textsc{DeleteVertex}(\mathcal{I}, v)$ is a valid distribution for $\mathcal{G}'$.*

Proof. Let $\mathcal{X}$ and $\mathcal{X}'$ be the distributions of the pair sequences corresponding to $\mathcal{I}$ and $\mathcal{I}'$, respectively. Then, we can obtain $\mathcal{X}'$ from $\mathcal{X}$ as follows. Let $(z_1, x_1), (z_2, x_2), \ldots$ be a sequence sampled from $\mathcal{X}$. If $z_i = v$, then we again replace z_i from $V(\mathcal{G}) \setminus \{v\}$ uniformly at random again. We can observe that $\mathcal{X}'$ is valid for $\mathcal{G}'$, and it follows that $\mathcal{I}'$ is valid for $\mathcal{G}'$. □

Lemma 6.6. *Let $\mathcal{G}$ be an influence graph and $\mathcal{G}'$ be the influence graph obtained from $\mathcal{G}$ by adding a new edge $(u,v) \notin E(\mathcal{G})$ with influence probability $p \in (0,1]$. If $\mathcal{I}$ is a valid distribution for $\mathcal{G}$, then $\mathcal{I}' = \textsc{AddEdge}(\mathcal{I}, (u,v), p)$ is a valid distribution for $\mathcal{G}'$.*

Proof. Let $\mathcal{X}$ and $\mathcal{X}'$ be the distribution of pair sequences corresponding to $\mathcal{I}$ and $\mathcal{I}'$, respectively. We can obtain $\mathcal{X}'$ from $\mathcal{X}$ as follows. Let $(z_1, x_1), (z_2, x_2), \ldots$ be a sequence sampled from $\mathcal{X}$. Then, we replace (z_i, x_i) by (z_i, x_i'), where x_i' is obtained from x_i by choosing $x_i'(u,v)$ from $[0,1]$ uniformly at random. We can observe that $\mathcal{X}'$ is valid for $\mathcal{G}'$, and it follows that $\mathcal{I}'$ is valid for $\mathcal{G}'$. □

Lemma 6.7. *Let $\mathcal{G}$ be an influence graph and $\mathcal{G}'$ be the influence graph obtained from $\mathcal{G}$ by removing an edge $(u,v) \in E(\mathcal{G})$. If $\mathcal{I}$ is a valid distribution for $\mathcal{G}$, then $\mathcal{I}' = \textsc{DeleteEdge}(\mathcal{I}, (u,v))$ is a valid distribution for $\mathcal{G}'$.*

Proof. Let $\mathcal{X}$ and $\mathcal{X}'$ be the distribution of pair sequences corresponding to $\mathcal{I}$ and $\mathcal{I}'$, respectively. We can obtain $\mathcal{X}'$ from $\mathcal{X}$ as follows. Let $(z_1, x_1), (z_2, x_2), \ldots$ be a sequence sampled from $\mathcal{X}$. Then, we replace (z_i, x_i) by (z_i, x_i'), where x_i' is the restriction of x to $E \setminus \{(u,v)\}$. We can observe that $\mathcal{X}'$ is valid for $\mathcal{G}'$, and it follows that $\mathcal{I}'$ is valid for $\mathcal{G}'$. □

Theorem 6.8. $\mathcal{I}_W^{\mathrm{dyn}}(\mathcal{G}) = \mathcal{I}_W(\mathcal{G})$.

Proof. From Lemmas 6.3–6.7, we have that the distribution of indices obtained by our dynamic update procedures is always a valid distribution of triplet sequences for the current influence graph. Because we apply $\textsc{Adjust}$ at the end of each update, the distribution of indices is exactly $\mathcal{I}_W(\mathcal{G})$ by Lemma 6.1. □

By Theorems 6.8 and 3.3, we obtain the following guarantee for influence estimation.

Theorem 6.9. *Let I be the index obtained by a sequence of dynamic updates, and assume that I contains θ sketches. Then, for any $\epsilon > 0$, an estimate returned by* ESTIMATEINFLUENCE*(I, S) in Algorithm 6.1 approximates the influence* $\mathsf{Inf}(S)$ *with an additive error of $\epsilon|V|$ with probability* $\exp(2\theta\epsilon^2)$ *over the choice of z_i's and x_i's.*

By Theorems 6.8 and 3.4, we obtain the following guarantee for influence maximization.

Theorem 6.10. *Let $W = 12k\epsilon^{-2}(|V| + |E|)\log|V|$, and let I be the index obtained by a sequence of dynamic updates. Then,* MAXIMIZEINFLUENCE*(I, k) in Algorithm 6.1 returns a set S of size k, such that* $\mathsf{Inf}(S) \geq (1 - \mathrm{e}^{-1} - \epsilon)\max_{S^* \in \binom{V}{k}} \mathsf{Inf}(S^*)$ *with probability at least* $3/5$ *over the choice of z_i's and x_i's.*

6.2.2 Time Complexity

Now, we turn our focus to analyzing the time complexity of our indexing method. We note that it is difficult to precisely bound the time complexity of dynamic update operations because it depends on the sizes of cascades and hence the structure of the input graph. Instead, we will analyze the number of sketches examined in each update operation.

To this end, we will apply the following lemma.

Lemma 6.11. *For a vertex set $S \subseteq V(\mathcal{G})$ and a randomly sampled sketch (z_i, x_i, H_i), it holds that*

$$\mathbf{Pr}[V(H_i) \cap S \neq \emptyset] = \frac{\mathsf{Inf}(S)}{|V(\mathcal{G})|}, \tag{6.3}$$

where the probability is over the choice of z_i and x_i.

Proof. From the construction of H_i, the proof is a direct consequence of [29, Observation 3.2]. □

Theorem 6.12. ADDVERTEX *examines $|I|$ sketches, and on average,* DELETEVERTEX, CHANGE, ADDEDGE, *and* DELETEEDGE *examine* $\frac{\mathsf{Inf}(v)}{|V|}|I|$ *sketches.*

Proof. The first claim is obvious as the number of sketches is $|I|$. Suppose that DELETEVERTEX(I, v), CHANGE$(I, (u, v), p)$, ADDEDGE$(I, (u, v), p)$, or DELETEEDGE$(I, (u, v), p)$ was called. Then, we only examine the sketches containing v and thus the expected number of examined sketches is $\frac{\mathsf{Inf}(v)}{|V|}|I|$, following from Lemma 6.11. □

6.3 Scaling-up Practical Performance

In this section, we introduce several techniques to improve the performance of the index update algorithms without deteriorating the quality of influence estimation and influence maximization.

6.3.1 Reachability-tree-based Pruning Techniques

Although we have demonstrated that the number of updated sketches resulting from vertex deletions and update operations on edges is small, naive implementations of these are computationally expensive, because a whole subgraph is scanned for each relevant sketch in the SHRINK procedure. Here, we will address this issue.

Observation 4.4 tells us that the removal of a single edge or vertex rarely effects the reachability among the vertices in a sketch, To exploit this, for each sketch we will store a *directed reachability tree* T_i in H_i rooted at z_i. Therefore, each sketch is now a quadruplet (z_i, x_i, H_i, T_i).

Definition of reachability trees.

Let us begin with the definition of the reachability tree (Figure 6.3a). The *reachability tree* T_i for a sketch (z_i, x_i, H_i) is a subgraph of H_i with $V(T_i) = V(H_i)$ and $E(T_i) \subseteq E(H_i)$ which satisfies that (1) it contains exactly $|V(H_i)| - 1$ edges and (2) each vertex in $V(H_i)$ is able to reach z_i along with T_i. For a vertex $v \in V(T_i)$, we define the *subtree* $T_i(v)$ rooted at v as the subgraph of T_i induced by the vertices that can reach v.

Speeding up the SHRINK procedure.

Next, we will explain how we use the reachability tree to efficiently prune the SHRINK procedure without scanning the whole subgraph.

Suppose that an edge (u, v) becomes blocked in the CHANGE procedure. Then, we need to update each sketch (z_i, x_i, H_i, T_i) with $(u, v) \in E(H_i)$ by calling the procedure SHRINK(I, i) (Algorithm 6.4, line 10). Previously, we updated H_i by performing a reverse BFS from z_i. Because we now have T_i, we can update H_i more efficiently. We present the details as follows.

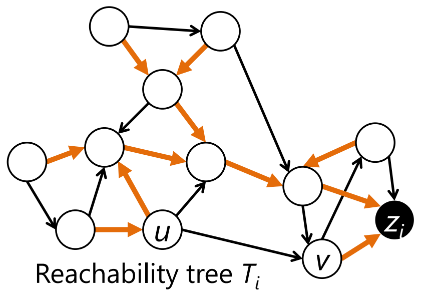
If $(u, v) \notin E(T_i)$, then u can still reach z_i along with T_i after the removal of (u, v), as we know there exists a "detour" from u to z_i avoiding (u, v) (Figure 6.3b). Thus, we do nothing; we call this the *technique of detour existence checking*. Otherwise, the vertices in $V(T_i(u))$ are the only candidates that may no longer reach z_i. Thus, we check whether the vertices in $V(T_i(u))$ can still reach $V(H_i) \setminus V(T_i(u))$. With this aim, we first explicitly compute the set $V(T_i(u))$ (Figure 6.3c). Then, we compute the set R of vertices in $V(T_i(u))$ connected to $V(H_i) \setminus V(T_i(u))$. Finally, we compute the set of vertices in $V(T_i(u))$ that can reach R, using a reverse BFS from R (Figure 6.3d). We call this the *technique of limiting the search range*. Then, we remove the vertices that can no longer reach R from $V(H_i)$ and update H_i and T_i. The pseudocode is presented as SHRINK-AFTER-EDGE-REMOVAL in Algorithm 6.5.

In addition, we can speed up SHRINK(I, i) at line 15 in the DELETEVERTEX procedure in a similar manner. The pseudocode is presented as SHRINK-AFTER-VERTEX-REMOVAL in Algorithm 6.5.

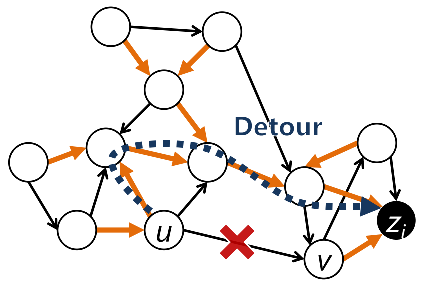
Maintaining reachability trees.

Now, we discuss how we maintain reachability trees. First, when creating a new sketch (z_i, x_i, H_i, T_i), T_i is a tree consisting of a single vertex z_i. When updating T_i in an existing sketch (z_i, x_i, H_i, T_i), we have the following four cases.

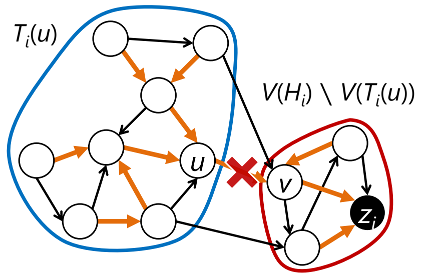
1. Suppose that an edge $(u, v) \in E(H_i)$ becomes blocked. Let H_i' be the new subgraph computed by calling SHRINK-AFTER-EDGE-REMOVAL$(I, i, (u, v))$.



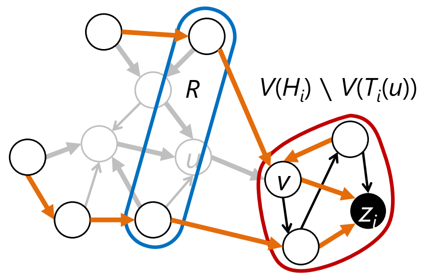
(a) Reachability tree



(b) Detour existence checking



(c) Limiting the search range (before deletion)



(d) Limiting the search range (after deletion)

Figure 6.3: Reachability-tree-based techniques for fast edge deletion. Orange edges are the edges of the reachability tree.

Note that the vertices in $R' = V(H_i') \cap V(T_i(u))$ are obtained by performing a reverse BFS from R (see SHRINK-AFTER-EDGE-REMOVAL for the definitions of R and R'). Hence, by removing the subtree $V(T_i(u))$ from T_i and concatenating the tree formed by this reverse BFS, we obtain a new reachability tree for H_i'.

2. Suppose that an edge (u, v) with $v \in V(H_i)$ becomes live. Then, we perform a reverse BFS from u, and add the edge (u, v) and the obtained subtree rooted at u to T_i.

3. When deleting a vertex v with $v \in V(H_i)$, we can update T_i, in a similar manner as in the case where an edge becomes blocked.

4. When adding a vertex, we are not required to do anything.

6.3.2 A Skipping Method for Vertex Addition

When adding a vertex, we are required to change the target vertex of each sketch with probability $\frac{1}{|V|+1}$. However, running through all of the sketches in I is a costly procedure. We can avoid this issue by applying the following technique. Let k be the first index for which we change the target vertex z_k. Then, for each positive integer t, we have $\mathbf{Pr}[k = t] = (1-\alpha)^{t-1}\alpha$, where $\alpha = \frac{1}{|V|+1}$. Hence, we can sample k by first sampling $y \in [0, 1]$ uniformly at random, and then taking the minimum k such that $\sum_{t \in [k]} (1-\alpha)^{t-1}\alpha \geq y$. This is equivalent to $k \geq \log \frac{1}{1-y} / \log \frac{1}{1-\alpha}$. We choose minimum such k with these properties and change the target vertex of the k^{th} sketch to v. Then, we repeat the same procedure for the remainder of the index. See ADDVERTEX' in Algorithm 6.5 for complete details.

Algorithm 6.5 Improved operations.

```
1: procedure Shrink-after-Edge-Removal($I, i, (u, v)$)
2:     if $(u, v) \notin E(T_i)$ then
3:         return
4:     $T' \leftarrow$ the subtree rooted at $u$.
5:     $R \leftarrow$ vertices in $T'$ connected to $V(H_i) \setminus T'$.
6:     $R' \leftarrow$ the set of vertices in $T'$ that can reach $R$.
7:     for $w \in T' \setminus R'$ do
8:         remove $w$ and the edges entering $w$ from $H_i$ and $T_i$, resp.
```

```
9: procedure Shrink-after-Vertex-Removal($I, i, v$)
10:    $T' \leftarrow$ the subtree rooted at $v$.
11:    $R \leftarrow$ vertices in $T'$ connected to $V(H_i) \setminus T'$.
12:    $R' \leftarrow$ the set of vertices in $T'$ that can reach $R$.
13:    for $w \in T' \setminus R'$ do
14:        remove $w$ and the edges entering $w$ from $H_i$ and $T_i$, resp.
```

```
15: procedure AddVertex'($I, v$)
16:    $\alpha \leftarrow \frac{1}{|V|+1}$, $i \leftarrow 0$.
17:    while $i < |I|$ do
18:        sample $y$ from $[0, 1]$ uniformly at random.
19:        $k \leftarrow \lceil \log \frac{1}{1-y} / \log \frac{1}{1-\alpha} \rceil$.
20:        $i \leftarrow i + k$.
21:        if $i \leq |I|$ then
22:            $H_i \leftarrow \emptyset$, $z_i \leftarrow v$.
23:            Expand($I, i, z_i$).
24:    Adjust($I$).
```

Because we are only required to seeing sketches for which we change the target vertex, the expected number of sketches we look at is $\frac{|I|}{|V|+1}$, which is much smaller than $|I|$.

6.4 Experiments

In this section, we will demonstrate the efficiency and effectiveness of our dynamic indexing method by performing experiments on real-world networks.

6.4.1 Setup

Datasets.

We selected seven real-world dynamic networks, where each edge has the *timestamp* at which it was created, from the Koblenz Network Collection [111, 112]. We ordered edges in ascending time order. The basic information regarding each dataset is presented in Table 6.1 and the degree distribution of each dataset is plotted in Figure 6.4.

Influence probability settings.

These networks do not contain information concerning influence probabilities. Therefore, we adopt $\textsc{uc}_{0.1}$, $\textsc{uc}_{0.01}$, TRI, and IWC settings described in Chapter 4.

Table 6.1: Datasets used in Chapter 6.

network	$\lvert V\rvert$	$\lvert E\rvert$	Type
Digg	30,398	85,247	communication (directed)
Enron	87,273	320,154	communication (directed)
Epinions	131,828	840,799	social (directed)
Facebook	63,731	1,634,070	social (undirected)
DBLP	1,314,050	10,724,828	collaboration (undirected)
YouTube	3,223,585	18,750,748	social (undirected)
Flickr	2,302,925	33,140,017	social (directed)

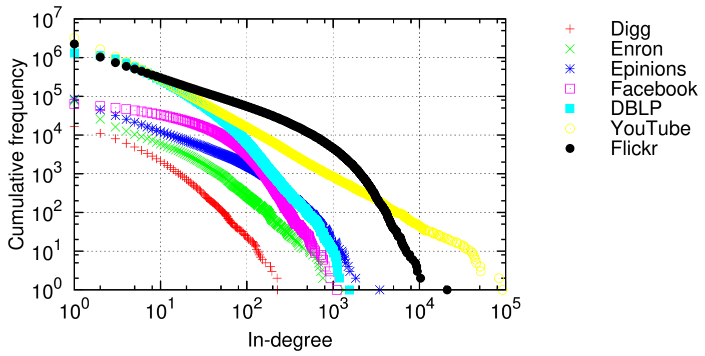


Figure 6.4: Degree distribution of each dataset.

Algorithms.

Our method is parameterized by W, which represents the threshold of the total weight. In order to apply the same parameter for various networks, we introduce the parameter β, which determines W in a manner such that $W = \beta(|V| + |E|)\log|V|$. Unless otherwise specified, we set $\beta = 32$. This choice will be justified in Section 6.4.4 empirically in terms of accuracy.

For the purpose of our comparison, we use the following algorithms in our experiments, which have been reviewed in Chapter 3. Recall that all the algorithms handle static graphs only.

- *RIS* [29]: A reverse influence sampling method on which our method is based. We set the total number of traversed edges to be $32(|V|+|E|)\log|V|$.
- *TIM*$^+$ [176]: A sketch-based method with the two-phase strategy. We use the implementation of [91]. We set the precision parameters as $\ell = 1$ and $\epsilon = 0.5$.
- *IMM* [177]: A sketch-based method that uses martingales. We use the implementation of [90]. We set the precision parameters as $\ell = 1$ and $\epsilon = 0.5$.
- *PMC* [156]: The pruned Monte-Carlo simulation algorithm proposed in Chapter 5. The number of subgraphs is set to 200.
- *IRIE* [96]: A heuristic-based method that uses a linear system. We use the code provided by Kyomin Jung, an author of [96]. We set the parameters as $\alpha = 0.7$ and $\theta = 1/320$.
- *Degree* [98]: A baseline method that chooses the topmost k vertices in decreasing order of out-degrees.

- *MC* [98]: A simulation-based influence estimation method that simulates the diffusion process 10,000 times, and calculates the average number of activated vertices.

Environments.

We conducted the experiments on a Linux server, with Intel Xeon E5-2690 2.90GHz CPU and 256GB memory. All algorithms were implemented in C++ and compiled using g++v4.6.3 with the -O2 option.

6.4.2 Index Construction

We first examine the scalability of our indexing algorithm. For each network, we constructed our index using the entire network. The indexing times and index sizes are presented in Table 6.2. From this, we can observe the scalability and efficiency of our method. For the larger three networks, which incorporate tens of millions of edges, it only requires a few hours to construct the index. However, we note that without our dynamic method, this amount of time is required to estimate or maximize the influence spread. We here note that the difference in the index size under the IWC model comes from the differences in the degree distribution. For example, YouTube has more vertices with high in-degree than DBLP. Thus, for each sketch construction, the total weight of the index for YouTube increases faster than DBLP. As a result, the obtained index for YouTube has a small number of sketches (1 million), which requires only 3.9GB, while the index for DBLP has a larger number of sketches (37 million) and consumes 34.8GB.

Table 6.3 reports the size of two types of index, where one is the proposed index that stores only live edges and the other one stores both the live edges and blocked edges. By discarding blocked edges from the sketches, the space consumption is reduced by a factor from 4 to 50. Figure 6.5 depicts the change of the indexing time and index size with the value of β from 2 to 2,048. Both of these values are scaled to β.

6.4.3 Dynamic Updates

We then evaluate the efficiency of our methods in terms of dynamic updates. Specifically, we have measured the running time of each operation as follows.

- Vertex additions: The average running time for adding 1,000 new isolated vertices to the index constructed using the whole network.
- Vertex deletions: The average running time for deleting 1,000 uniformly chosen vertices from the index constructed using the whole network.
- Influence probability updates: The average running time for updating the influence probabilities of 1,000 uniformly chosen edges of the index constructed using the whole network. The update is conducted as follows. Suppose that an edge e with influence probability $p(e)$ is chosen. When using the TRI setting, we randomly choose a probability from $\{0.1, 0.01, 0.001\} \setminus \{p(e)\}$ as the new influence probability of e. When using other settings, we randomly assign $p(e) \times 2$ or $p(e)/2$ to e.
- Edge additions: The average running time for adding the final 1,000 edges to the index constructed using all of the edges except for the final 1,000 edges.

- Edge deletions: The average running time for deleting 1,000 edges in the reverse of the order that they were added, from the index constructed using the whole network.

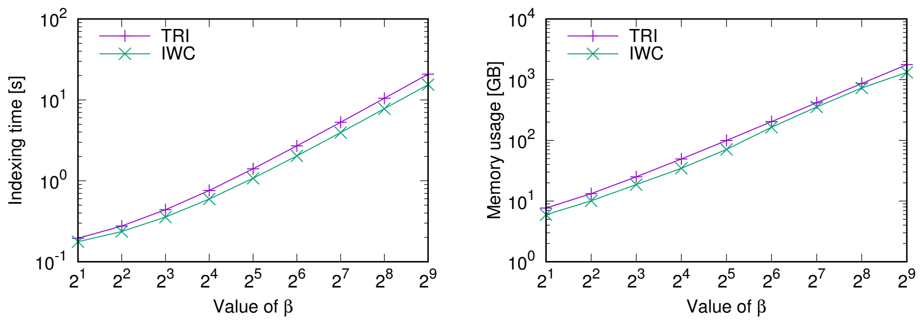
Table 6.2 presents the average running times of the dynamic update operations. Each vertex addition and edge deletion is processed within a few milliseconds. For the three largest networks, an edge addition requires several hundred milliseconds under the $\textsc{uc}_{0.1}$ setting, several ten milliseconds under the TRI setting, and a few milliseconds under the IWC setting. The reason for this is that under unweighted settings, sketches are likely to expand more following the addition of edges in comparison with the IWC setting. Vertex deletion exhibits a similar tendency to edge addition, where it becomes slower for the probability setting of $\textsc{uc}_{0.1}$. In the slowest case, it took four seconds for deleting a single vertex on Flickr because vertex deletion causes a number of edge deletions. Figure 6.5 suggests that the average processing times of dynamic operations are roughly proportional to the value of β, because the expected size of I is proportional to β.

Next, we analyze the effectiveness of the proposed speed-up techniques introduced in Sections 6.3.1 and 6.3.2. Table 6.3 presents the average running times of dynamic updates with or without the proposed pruning techniques. The proposed techniques improve the performances of edge deletion, vertex deletion, and vertex addition, making them several hundred times faster. Notice that our techniques for deletion operations are more effective for larger networks.

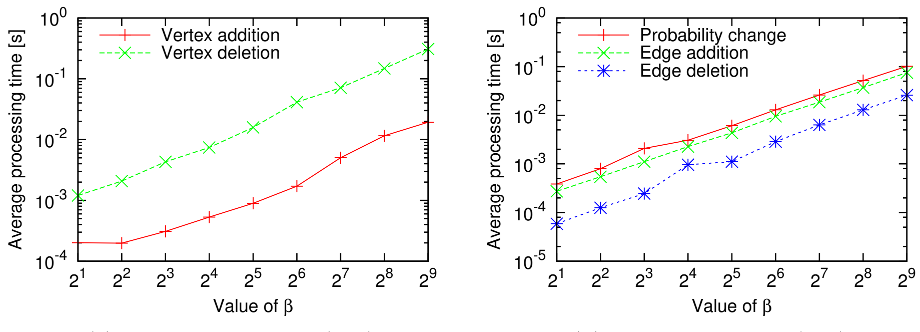
Here, we discuss how our technique improves the efficiency of edge deletions. Table 6.4 reports the detailed information of the reachability-tree-based technique on edge deletion. In this table, the third column shows the number of sketch updates examined during 1,000 edge deletions, the fourth column shows the number of sketch updates that were skipped by our technique of detour existence checking, the fifth column shows the average size of the sketches that were not skipped, and the sixth column shows the average size of the subtrees that our technique of limiting the search range computed. Firstly, our detour existence checking skipped a significant fraction of sketches without any BFS, e.g., 24% have been skipped for Flickr ($\textsc{uc}_{0.1}$). Then, for sketches that have not been skipped, we perform a BFS on subtrees. Those subtrees contain a few vertices on average, thus we were able to cut down the computation cost significantly.

Table 6.2: Indexing time, index size, and average processing times of dynamic updates.

network	model	indexing time	index size	vertex addition	vertex deletion	probability change	edge addition	edge deletion
Digg	$UC_{0.01}$	42.0 s	3.2 GB	3.1 ms	2.8 ms	0.16 ms	0.16 ms	0.20 ms
	$UC_{0.1}$	19.4 s	1.3 GB	1.3 ms	5.9 ms	3.1 ms	1.2 ms	1.6 ms
	TRI	36.3 s	3.0 GB	2.6 ms	2.9 ms	0.59 ms	0.32 ms	0.39 ms
	IWC	20.3 s	1.4 GB	1.6 ms	4.0 ms	0.92 ms	0.81 ms	1.3 ms
Enron	$UC_{0.01}$	95.3 s	5.9 GB	2.3 ms	5.4 ms	0.81 ms	0.40 ms	0.54 ms
	$UC_{0.1}$	41.3 s	1.0 GB	1.1 ms	11.7 ms	6.4 ms	5.6 ms	2.0 ms
	TRI	27.3 s	0.6 GB	0.46 ms	8.1 ms	3.4 ms	1.2 ms	1.2 ms
	IWC	16.5 s	0.6 GB	0.33 ms	4.6 ms	1.4 ms	0.39 ms	0.94 ms
Epinions	$UC_{0.01}$	65.2 s	0.8 GB	0.68 ms	13.1 ms	2.9 ms	1.2 ms	1.8 ms
	$UC_{0.1}$	119.1 s	2.3 GB	1.7 ms	57.4 ms	27.5 ms	15.8 ms	3.3 ms
	TRI	89.1 s	1.4 GB	0.80 ms	14.8 ms	5.8 ms	4.1 ms	1.0 ms
	IWC	62.2 s	1.1 GB	0.69 ms	8.3 ms	1.7 ms	1.0 ms	1.8 ms
Facebook	$UC_{0.01}$	127.3 s	1.1 GB	2.6 ms	80.4 ms	3.0 ms	0.32 ms	0.74 ms
	$UC_{0.1}$	200.0 s	3.6 GB	2.4 ms	360.7 ms	35.8 ms	47.6 ms	3.1 ms
	TRI	165.0 s	2.1 GB	2.7 ms	95.8 ms	11.1 ms	6.9 ms	0.75 ms
	IWC	135.2 s	1.6 GB	2.7 ms	61.0 ms	3.6 ms	1.2 ms	2.3 ms
DBLP	$UC_{0.01}$	5,684.9 s	106.2 GB	8.3 ms	12.2 ms	1.3 ms	0.44 ms	0.39 ms
	$UC_{0.1}$	4,982.3 s	53.0 GB	3.5 ms	930.4 ms	895.4 ms	735.2 ms	24.2 ms
	TRI	2,965.9 s	27.7 GB	5.2 ms	124.0 ms	85.7 ms	42.5 ms	2.1 ms
	IWC	3,395.7 s	34.8 GB	4.0 ms	18.1 ms	2.7 ms	0.87 ms	1.3 ms
YouTube	$UC_{0.01}$	3,238.7 s	18.3 GB	1.2 ms	14.1 ms	26.8 ms	0.54 ms	0.16 ms
	$UC_{0.1}$	6,593.6 s	92.8 GB	7.9 ms	722.0 ms	1,777.9 ms	388.6 ms	0.68 ms
	TRI	5,000.4 s	44.6 GB	0.01 ms	92.2 ms	236.2 ms	31.8 ms	0.26 ms
	IWC	1,985.5 s	3.9 GB	0.65 ms	5.7 ms	1.5 ms	0.14 ms	0.04 ms
Flickr	$UC_{0.01}$	4,984.1 s	17.0 GB	0.00 ms	108.5 ms	17.0 ms	5.2 ms	0.29 ms
	$UC_{0.1}$	6,758.6 s	65.7 GB	0.00 ms	3,820.4 ms	610.4 ms	700.1 ms	19.3 ms
	TRI	5,467.6 s	31.3 GB	0.00 ms	459.0 ms	125.2 ms	89.6 ms	2.4 ms
	IWC	4,253.7 s	12.2 GB	2.1 ms	53.8 ms	4.8 ms	0.19 ms	0.08 ms


(a) Indexing time

(b) Index size


(c) Vertex operations (TRI)

(d) Edge operations (TRI)

Figure 6.5: The change of indexing time, index size, and average processing times of dynamic updates with the increase of β on Epinions.

Table 6.3: Effectiveness of the proposed techniques compared to naive implementations.

		edge deletion		vertex deletion		vertex addition		index size	
network	**model**	Sec. 6.3.1	naive	Sec. 6.3.1	naive	Sec. 6.3.2	naive	live edges	all edges
Epinions	TRI	1.0 ms	163.1 ms	14.8 ms	575.3 ms	0.80 ms	1.6 ms	1.4 GB	5.5 GB
	IWC	1.8 ms	1.4 ms	8.3 ms	7.4 ms	0.69 ms	6.4 ms	1.1 GB	7.0 GB
DBLP	TRI	2.1 ms	914.5 ms	124.0 ms	$>$ 10,000.0 ms	5.2 ms	2.4 ms	27.7 GB	130.7 GB
	IWC	1.3 ms	0.94 ms	18.1 ms	23.4 ms	4.0 ms	178.9 ms	34.8 GB	120.3 GB
Flickr	TRI	2.4 ms	1,705.5 ms	459.0 ms	$>$ 10,000.0 ms	0.00 ms	6.5 ms	31.3 GB	* $\approx$ 282.0 GB
	IWC	0.08 ms	0.10 ms	53.8 ms	41.2 ms	2.1 ms	33.6 ms	12.2 GB	* $\approx$ 292.1 GB

* We report twice the index size for $\beta = 16$ as an approximation of that for $\beta = 32$.

Table 6.4: Impact of reachability-tree-based technique in Sec. 6.3.1 on edge deletion.

network	**model**	**# examined sketches**	**# skipped sketches**	**average size of remained sketches**	**average size of remained subtrees**
Epinions	TRI	24,684	14,159	8,735.2	3.2
	IWC	8,875	386	58.8	5.8
DBLP	TRI	14,450	308	65,827.4	1.5
	IWC	12,751	1,178	28.1	3.4
Flickr	TRI	5,003	620	184,275.0	1.0
	IWC	3,277	359	21.7	1.1

Table 6.5: Average running time for estimating the influence spread of a single vertex.

		this work		static methods	
network	**model**	**indexing**	**query**	*MC*	*RIS*
Epinions	$\mathrm{UC}_{0.01}$	65.2 s	1.43 µs	0.1 s	8.9 s
	$\mathrm{UC}_{0.1}$	119.1 s	0.99 µs	13.0 s	8.1 s
	TRI	89.1 s	0.97 µs	6.3 s	8.7 s
	IWC	62.2 s	0.96 µs	0.01 s	9.3 s
DBLP	$\mathrm{UC}_{0.01}$	5,684.9 s	1.60 µs	0.006 s	504.4 s
	$\mathrm{UC}_{0.1}$	4,982.3 s	1.63 µs	> 100.0 s	271.1 s
	TRI	2,965.9 s	1.62 µs	48.0 s	267.1 s
	IWC	3,395.7 s	1.41 µs	0.02 s	298.1 s

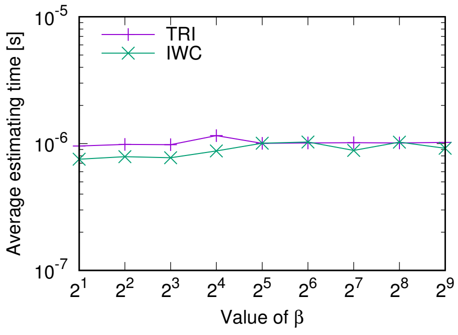


Figure 6.6: Average times for estimating influence of a single vertex with β on Epinions.

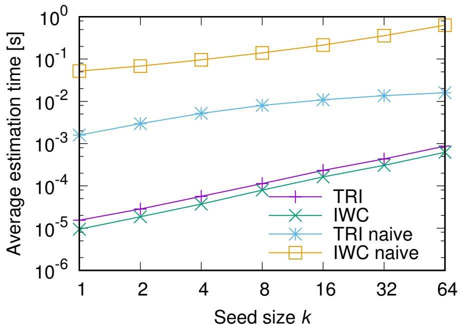


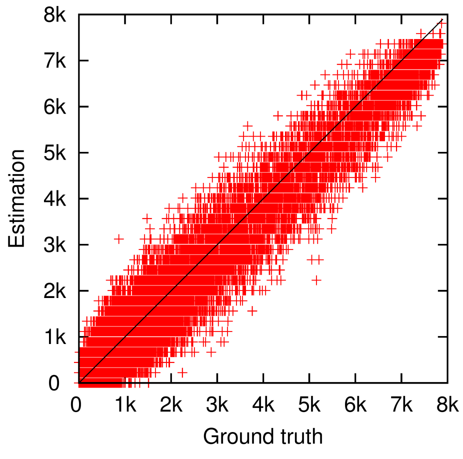
Figure 6.7: Average times for estimating influence of a vertex set of various sizes on Epinions.

6.4.4 Influence Estimation Queries

Now, we will show that our method efficiently and accurately estimates the influence spread using the index constructed from a given graph. As the exact computation of the influence spread is #P-hard, we regard the estimate given using *MC* as the ground truth of the influence spread.

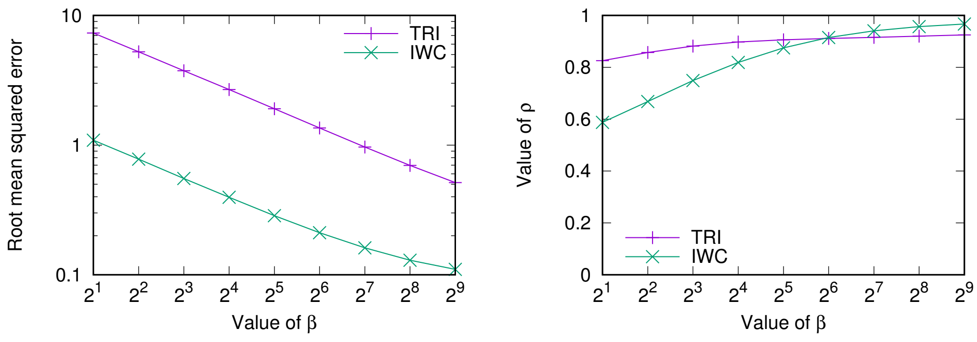
First, we focus on the influence estimation for a single vertex. For each network, we randomly sampled 1,000 vertices, and then estimated the influence spread for each vertex. Table 6.5 presents the average estimation time for each method. The average query time for our method is of the order of a few microseconds, which is several orders of magnitude faster than both *RIS* and *MC*. Therefore, once we construct our index, we can perform efficient tracking of influential vertices. Figure 6.6 indicates that the average estimation time is robust against changes in β, because it requires only constant time.

Figure 6.8 illustrates the accuracy of our method, where each point corresponds to a seed set consisting of a single vertex, and the x and y coordinates represent the influence spreads computed by *MC* and our method, respectively. It can be seen that as β increases, our method becomes more accurate. Even when $\beta = 1$, our method is stable, in the sense that all the points are close to the diagonal (i.e., the line of $y = x$) and we do not have any outliers. This property is desirable, because such outliers would result in huge errors when we want to maximize the influence spread. We here justify our choice of parameter β. We



(a) Epinions (TR, $\beta = 1$) (b) Epinions (TR, $\beta = 32$)

Figure 6.8: Correlation between the ground truth and influence estimation calculated by our method.



(a) Root mean squared error (b) Rank correlation coefficient

Figure 6.9: Accuracy improvements of influence estimation with the increase of β on Epinions.

plotted the root mean squared error, which is calculated as

$$\sum_{v \in V} \left(\frac{\widehat{\mathsf{Inf}}(v) - \mathsf{Inf}(v)}{\mathsf{Inf}(v)} \right)^2, \tag{6.4}$$

where $\widehat{\mathsf{Inf}}(v)$ is an estimation by our method and Spearman's rank correlation coefficient between the influence estimations of our method with various values of β and those from MC, as presented in Figure 6.9. Higher values of β yield more accurate influence estimations, but the improvements are limited when $\beta > 32$. Because the efficiency of index construction and dynamic updates depends on the value of β, we adopted $\beta = 32$ as a sweet spot between accuracy and efficiency.

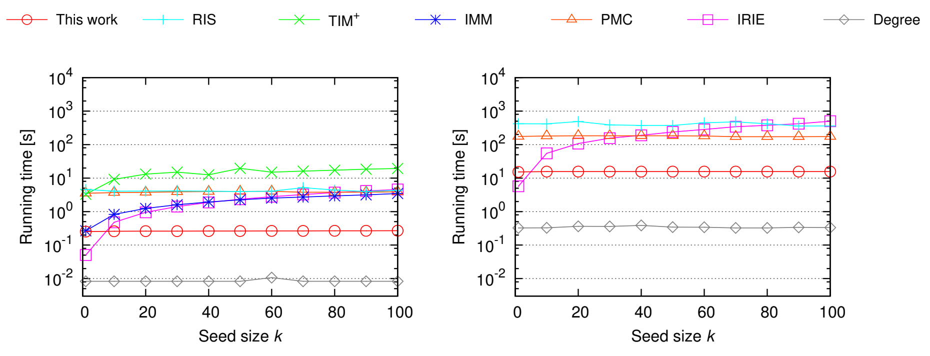
Next, we evaluate the efficiency of the influence estimation for a set of multiple vertices. We randomly generated 1,000 vertex sets of a specific size, and then estimated the influence spread for each vertex set. Figure 6.7 presents the average estimation times for a vertex set of sizes ranging from 1 to 64. The estimation times are scaled to the seed set size, and under one millisecond is required for a seed set of size 64, which is 10–100 times faster than the required times without our speed-up technique.

6.4.5 Influence Maximization Queries

Finally, we will demonstrate that our method processes influence maximization queries efficiently and accurately using the index constructed from a given graph.

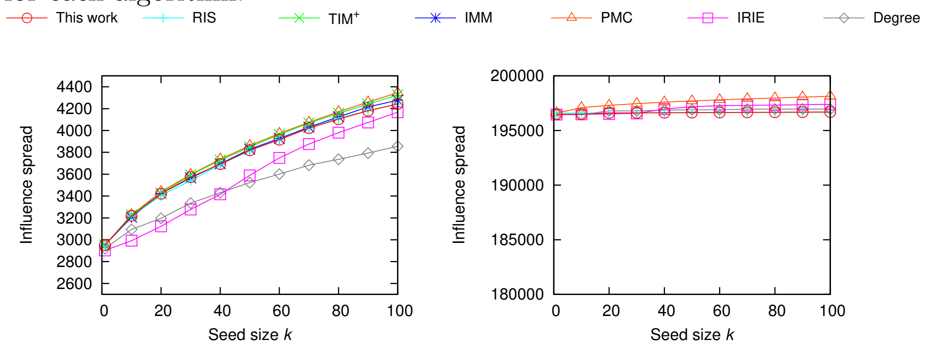
Figure 6.10 summarizes the running times required to compute seed sets of sizes $1, 10, 20, \ldots, 100$ after reflecting all of the edges in each network. Note that the running times do not include the times needed to read the input graph from a secondary storage location. Both *TIM*$^+$ and *IMM* did not finish within two hours on Flickr (TRI, $k \geq 1$). Our method returns a seed set within 20 seconds for all of the settings, whereas other static methods require a time of at least one order of magnitude longer. It should be noted that finding a seed set of the same quality that our method delivers from scratch requires ten times longer, as the performance of *RIS* demonstrates. We can also observe the robustness of our method against the seed size k, while *TIM*$^+$, *IMM*, and *IRIE* become slower as k increases.

Figure 6.11 presents the influence spread for seed sets of sizes $1, 10, 20, \ldots, 100$, as computed using each method. As we can see, our method and *RIS* deliver almost the same quality. This is not a coincidence because we have a theoretical guarantee that both our method and *RIS* generate indices sampled from the same distribution. *TIM*$^+$, *IMM*, and *PMC* also gave seed sets of a similar quality to our method, and this is because they also have accuracy guarantees. *IRIE* and *Degree* perform comparatively badly on Enron (TRI).



(a) Enron (TRI) (b) Flickr (TRI)

Figure 6.10: Running times for extracting a seed set of size from 1 to 100 for each algorithm.



(a) Enron (TRI) (b) Flickr (TRI)

Figure 6.11: The influence spreads of a seed set of size from 1 to 100 computed by each algorithm.

6.4.6 Case Study on Flixster Social Network

Here, we finally apply our dynamic index to track the important individuals in real-world networks. To that end, we downloaded the dataset of a movie review site Flixster provided by Mohsen Jamali [93]. This dataset contains the social network of Flixster, where vertices correspond to users of Flixster, and undirected edges correspond to the friendship between a pair of users, and the log of movie ratings, each of which is a triplet consisting of a user id, a movie id, and a rating with timestamps between February 2006 and November 2009. We then learned the influence probability of edges at the beginning of each month by applying a method in [79].

First, we show the influence spread for individuals. Figure 6.12 illustrates the transition of the influences of selected vertices in the Flixster network. We observe that several vertices have critical times at which their influences rapidly grow or decline. This type of query is therefore useful for evaluating, comparing, and identifying influential people or groups on the latest snapshot of a dynamic network.

Then, we issue a number of queries of influence maximization on this evolving network. Figure 6.13 illustrates the transition of the (approximated) maximum influence spread of a seed vertex set of size 100 computed by our method. Remarkable difference can be observed between the static setting (i.e., the outdated solution) and the dynamic setting (i.e., the updated solution), which suggests the usefulness of dynamic methods.

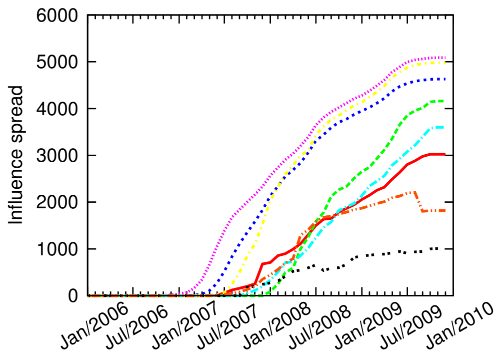


Figure 6.12: Transition of the influence spread of popular vertices in a real-world network.

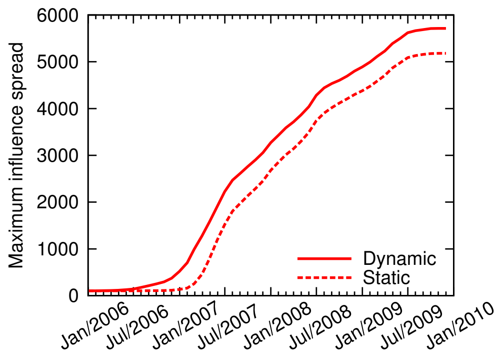


Figure 6.13: Transition of the approximated maximum influence spread of a seed set of size 100.

Chapter 7

Reduction Algorithms of Massive Influence Graphs

In this chapter, we propose a new algorithm for influence graph reduction. Figure 7.1 shows an overview of our overall approach. We first introduce the central reduction strategy called *coarsening* followed by a theoretical analysis of its impact on the graph size and the influence function (Section 7.1). We then provide two implementations according to the available space and their extensions to distributed and parallel systems (Section 7.2). We further present general frameworks using our coarsened graphs that accelerate existing algorithms for both influence estimation and influence maximization (Section 7.3). We end this chapter with intensive experimental evaluations on real-world massive networks with up to billions of edges (Section 7.4).

7.1 Reduction Strategy

7.1.1 Definition of Coarsening

We begin with the proposed influence graph reduction strategy. The overall idea is to shrink the giant component in advance. The central strategy for this purpose is *coarsening*, which merges a certain vertex set into a single weighted vertex, in the intention of making no distinction among vertices in the set. Our coarsening procedure is formally defined as follows.

Definition 7.1 (Coarsened influence graph)**.** *Let $\mathcal{G} = (V, E, p)$ be an influence graph and $\mathcal{P} = \{C_j\}_{j \in [\ell]}$ be a partition of V, where each C_j is strongly connected. Then, a* coarsened influence graph *obtained from $\mathcal{G}$ by coarsening $\mathcal{P}$ is defined as*

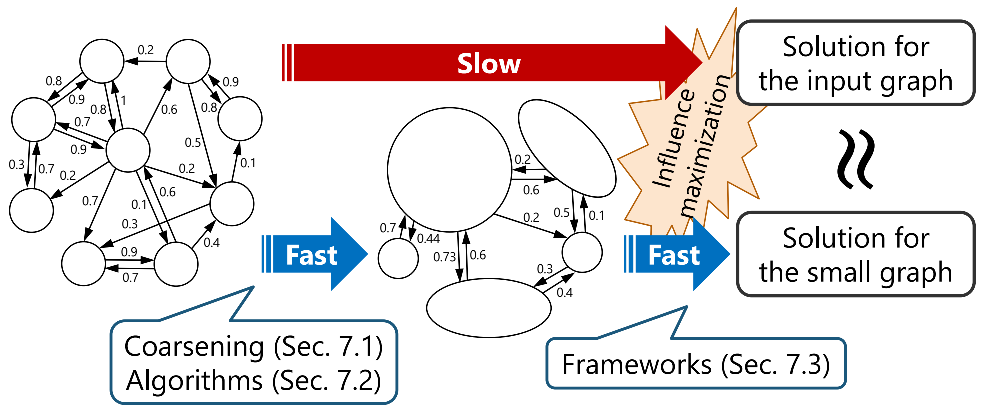


Figure 7.1: Overview of our approach for influence graph reduction in Chapter 7.

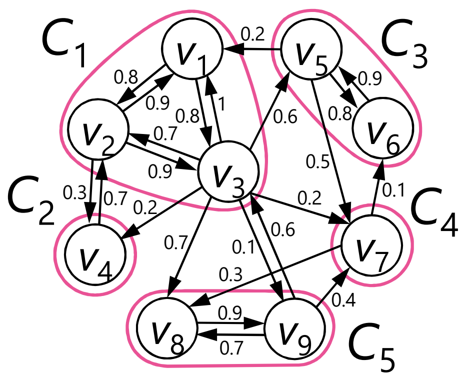


Figure 7.2: Influence graph $\mathcal{G}$.

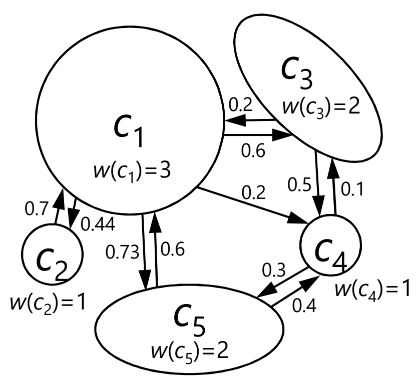


Figure 7.3: Coarsened graph $\mathcal{H}$.

an influence graph $\mathcal{H} = (W, F, q, \mathsf{w})$*, where*

$$W = \{c_j \mid j \in [\ell]\}, \tag{7.1}$$

$$F = \{(c_x, c_y) \mid c_x \neq c_y, \exists (u, v) \in E, u \in C_x, v \in C_y\}, \tag{7.2}$$

$$q(c_x, c_y) = 1 - \prod_{\substack{(u,v)\in E \\ u \in C_x, v \in C_y}} (1 - p(u, v)) \qquad (\forall (c_x, c_y) \in F), \tag{7.3}$$

$$\mathsf{w}(c_j) = |C_j| \qquad (\forall c_j \in W). \tag{7.4}$$

The correspondence mapping $\pi : V \to W$ *is defined as* $\pi(v) = c_j$ such that $v \in C_j$.

We abuse the notation to let π act on subsets by writing $\pi(S) = \{\pi(v) \mid v \in S\}$. It should also be noted that $F = \{(\pi(u), \pi(v)) \mid \pi(u) \neq \pi(v), (u, v) \in E\}$.

Example 7.2. *Let us take an example to give an intuition of the definition. Figures 7.2 and 7.3 show an influence graph* $\mathcal{G}$ *and a coarsened influence graph* $\mathcal{H}$*, respectively. Here, a vertex partition to be coarsened is* $\mathcal{P} = \{C_1, C_2, C_3, C_4, C_5\} = \{\{v_1, v_2, v_3\}, \{v_4\}, \{v_5, v_6\}, \{v_7\}, \{v_8, v_9\}\}$*. There is a one-to-one correspondence between a vertex set* C_j *in* $\mathcal{P}$ *and a vertex* c_j *in* $\mathcal{H}$*, i.e.,* $\mathcal{P}$ *consists of five vertex sets, and thus,* $\mathcal{H}$ *consists of five vertices. Each vertex in* $\mathcal{H}$ *has a weight that is the size of the corresponding component in* $\mathcal{G}$*, e.g.,* $|C_1| = 3$*; thus,* $\mathsf{w}(c_1) = 3$*. When there is an edge connecting from* C_x *to* C_y *with* $x \neq y$ *in* $\mathcal{G}$*, then there is a corresponding edge from* c_x *to* c_y *in* $\mathcal{H}$*, e.g.,* (v_2, v_4) *is in* $\mathcal{G}$*, and thus* $(\pi(v_2), \pi(v_4)) = (c_1, c_2)$ *is in* $\mathcal{H}$*. An activation through edge* (c_x, c_y) *in* $\mathcal{H}$ *corresponds to some activation among edges from* C_x *to* C_y *in* $\mathcal{G}$*, whose event probability is given by Eq. (7.3), e.g., influence probabilities of two edges connecting from* C_1 *to* C_2 *are* 0.3 *and* 0.2*, and thus, the influence probability of edge* (c_1, c_2) *is calculated as* $1 - (1 - 0.3)(1 - 0.2) = 0.44$*.*

Intuitively, for a seed set $S \subseteq V$, the diffusion process on $\mathcal{H}$ starting from $\pi(S)$ "emulates" the diffusion process on $\mathcal{G}$ starting from S. We use $\mathsf{Inf}_{(W,F,q),\mathsf{w}}(\pi(S))$ as an approximation of $\mathsf{Inf}_{\mathcal{G}}(S)$. Hereafter, we use $\mathsf{Inf}_{\mathcal{H}}(\cdot)$ to denote $\mathsf{Inf}_{(W,F,q),\mathsf{w}}(\cdot)$. Hence, a coarsened influence graph $\mathcal{H}$ is preferable if (1) $\mathcal{H}$ is much smaller than $\mathcal{G}$ and (2) $\mathsf{Inf}_{\mathcal{H}}(\pi(\cdot))$ is close to $\mathsf{Inf}_{\mathcal{G}}(\cdot)$. Of course, there are exponentially many candidates for the vertex partition to be coarsened, and we cannot evaluate all the candidates. To resolve this issue, in subsequent sections, we will investigate what type of vertex partition is desired and discuss how to create such a partition.

7.1.2 Theoretical Properties of Coarsening

We investigate the impact of coarsening on the influence function and the graph size. For an influence graph $\mathcal{G} = (V, E, p)$ and a partition $\mathcal{P} = \{C_j\}_{j\in[\ell]}$ of V, let $\mathcal{H} = (W, F, q, \mathsf{w})$ be an influence graph and $\pi : V \to W$ be the correspondence mapping obtained from $\mathcal{G}$ by coarsening $\mathcal{P}$. Then, an *intermediate influence graph* between $\mathcal{G}$ and $\mathcal{H}$ is defined as $\mathcal{I} = (V, E, p')$, where $p'(u, v) = 1$ if $u, v \in C_j$ for some $j \in [\ell]$, otherwise $p'(u, v) = p(u, v)$.

Here, we show the equivalence between $\mathcal{I}$ and $\mathcal{H}$ in terms of the influence function. Therefore, it is sufficient to examine the influence on $\mathcal{I}$, which is of the same structure as $\mathcal{G}$, rather than $\mathcal{H}$.

Lemma 7.3. *For any $S \subseteq V$, $\mathsf{Inf}_{\mathcal{I}}(S) = \mathsf{Inf}_{\mathcal{H}}(\pi(S))$.*

Proof. For each $j \in [\ell]$, we define $E_j = \{(u, v) \in E \mid u, v \in C_j\}$ and $E_{\mathrm{o}} = E \setminus (\bigcup_{j\in[\ell]} E_j)$. Note that a family consisting of $E_1, \dots, E_\ell$ and E_{o} forms a partition of E. For $X_j \subseteq E_j$, $p'(X_j \mid E_j)$ is 1 if $X_j = E_j$, otherwise 0. Thus, by Eq. (2.10),

$$\begin{aligned}
\mathsf{Inf}_{\mathcal{I}}(S) &= \sum_{X\subseteq E} p'(X \mid E) \cdot \mathsf{r}_{(V,X)}(S) \\
&= \sum_{X_1\subseteq E_1} p'(X_1 \mid E_1) \cdots \sum_{X_\ell\subseteq E_\ell} p'(X_\ell \mid E_\ell) \sum_{X_{\mathrm{o}}\subseteq E_{\mathrm{o}}} p'(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V,X_{\mathrm{o}}\cup\bigcup_j X_j)}(S) \\
&= \sum_{X_{\mathrm{o}}\subseteq E_{\mathrm{o}}} p'(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V,X_{\mathrm{o}}\cup\bigcup_j E_j)}(S). \qquad (7.5)
\end{aligned}$$

For each $Y \subseteq F$, we define a family $\mathcal{X}_Y$ of edge sets as $\mathcal{X}_Y = \{X_{\mathrm{o}} \subseteq E_{\mathrm{o}} \mid \{(\pi(u), \pi(v)) \mid \pi(u) \neq \pi(v), (u, v) \in X_{\mathrm{o}}\} = Y\}$. Note that a collection of $\{\mathcal{X}_Y\}_{Y\subseteq F}$ forms a partition of the power set $2^{E_{\mathrm{o}}}$. Recall that $q(Y \mid F) = \prod_{e\in Y} q(e) \prod_{e\in F\setminus Y}(1 - q(e))$. From the construction of q (Eq. (7.3)), it turns out that

$$q(Y \mid F) = \sum_{X_{\mathrm{o}}\in\mathcal{X}_Y} p'(X_{\mathrm{o}} \mid E_{\mathrm{o}}). \qquad (7.6)$$

For any s and t in V and $X_{\mathrm{o}} \in \mathcal{X}_Y$, "$s$ can reach t by passing through the edges in $X_{\mathrm{o}} \cup \bigcup_j E_j$" if and only if "$\pi(s)$ can reach $\pi(t)$ by passing through the edges in Y" since every subgraph (C_j, E_j) for $j \in [\ell]$ is strongly connected; therefore, it holds that $\mathsf{r}_{(V,X_{\mathrm{o}}\cup\bigcup_j E_j)}(S) = \mathsf{r}_{(W,Y),\mathsf{w}}(\pi(S))$ for all $X_{\mathrm{o}} \in \mathcal{X}_Y$. Thus,

$$\begin{aligned}
\mathsf{Inf}_{\mathcal{I}}(S) &= \sum_{Y\subseteq F} \sum_{X_{\mathrm{o}}\in\mathcal{X}_Y} p'(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V,X_{\mathrm{o}}\cup\bigcup_j E_j)}(S) \\
&= \sum_{Y\subseteq F} q(Y \mid F) \cdot \mathsf{r}_{(W,Y),\mathsf{w}}(\pi(S)) = \mathsf{Inf}_{\mathcal{H}}(\pi(S)). \qquad (7.7)
\end{aligned}$$

□

The gap of influence between $\mathcal{G}$ and $\mathcal{H}$.

We first give a lower bound of the influence on $\mathcal{I}$. From the following lemma, it turns out that $\mathsf{Inf}_{\mathcal{I}}(S) \geq \mathsf{Inf}_{\mathcal{G}}(S)$ for any $S \subseteq V$.

Lemma 7.4. *Let $\mathcal{G} = (V, E, p)$ and $\mathcal{G}' = (V, E, p')$ be two influence graphs with the same structure, where $p(e) \leq p'(e)$ for every edge e. Then, $\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{G}'}(S)$ for any $S \subseteq V$.*

Proof. We can construct a sequence of $|E|+1$ influence probability functions $p = p^0, p^1, \dots, p^{|E|-1}, p^{|E|} = p'$ such that p^{i-1} and p^i differ in only at most one element, say e^*, and $p^{i-1}(e^*) \leq p^i(e^*)$ for $i \in [|E|]$. Then, $\mathsf{Inf}_{(V,E,p^{i-1})}(S) \leq \mathsf{Inf}_{(V,E,p^i)}(S)$ for each $i \in [|E|]$ follows from Eq. (2.10), and thus we obtain that $\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{G}'}(S)$. □

For an influence graph $\mathcal{G} = (V, E, p)$, the *strongly connected reliability* of $\mathcal{G}$, denoted $\mathsf{Rel}(\mathcal{G})$, is defined as the probability that the random graph sampled from $\mathcal{G}$ is strongly connected, i.e.,

$$\mathsf{Rel}(\mathcal{G}) = \sum_{X \subseteq E} p(X \mid E) \cdot \big[(V, X) \text{ is SC}\big], \tag{7.8}$$

where $\big[\cdot\big]$ returns 1 if the given statement is true, and 0 otherwise.

We now show an upper bound of the influence on $\mathcal{I}$ by using the strongly connected reliability of the subgraph of $\mathcal{G}$.

Lemma 7.5. *For any $S \subseteq V$,*

$$\mathsf{Inf}_{\mathcal{I}}(S) \leq \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1} \cdot \mathsf{Inf}_{\mathcal{G}}(S). \tag{7.9}$$

Proof. For each $j \in [\ell]$, we define $E_j = \{(u,v) \in E \mid u, v \in C_j\}$ and $E_{\mathrm{o}} = E \setminus (\bigcup_{j \in [\ell]} E_j)$. For $X_j \subseteq E_j$ ($j \in [\ell]$) and $X_{\mathrm{o}} \subseteq E_{\mathrm{o}}$, $\mathsf{r}_{(V, X_{\mathrm{o}} \cup \bigcup_j X_j)}(S) = \mathsf{r}_{(V, X_{\mathrm{o}} \cup \bigcup_j E_j)}(S)$ holds if every subgraph (C_j, X_j) for $j \in [\ell]$ is strongly connected. Thus,

$$\begin{aligned}
\mathsf{Inf}_{\mathcal{G}}(S) &= \sum_{X_1 \subseteq E_1} p(X_1 \mid E_1) \cdots \sum_{X_\ell \subseteq E_\ell} p(X_\ell \mid E_\ell) \\
&\qquad \sum_{X_{\mathrm{o}} \subseteq E_{\mathrm{o}}} p(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V, X_{\mathrm{o}} \cup \bigcup_j X_j)}(S) \\
&\geq \sum_{\substack{X_1 \subseteq E_1 \\ (C_1, X_1) \text{ is SC}}} p(X_1 \mid E_1) \cdots \sum_{\substack{X_\ell \subseteq E_\ell \\ (C_\ell, X_\ell) \text{ is SC}}} p(X_\ell \mid E_\ell) \\
&\qquad \sum_{X_{\mathrm{o}} \subseteq E_{\mathrm{o}}} p(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V, X_{\mathrm{o}} \cup \bigcup_j X_j)}(S) \\
&= \Bigg(\sum_{\substack{X_1 \subseteq E_1 \\ (C_1, X_1) \text{ is SC}}} p(X_1 \mid E_1) \Bigg) \cdots \Bigg(\sum_{\substack{X_\ell \subseteq E_\ell \\ (C_\ell, X_\ell) \text{ is SC}}} p(X_\ell \mid E_\ell) \Bigg) \\
&\qquad \cdot \Bigg(\sum_{X_{\mathrm{o}} \subseteq E_{\mathrm{o}}} p(X_{\mathrm{o}} \mid E_{\mathrm{o}}) \cdot \mathsf{r}_{(V, X_{\mathrm{o}} \cup \bigcup_j E_j)}(S) \Bigg) \\
&= \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j]) \cdot \mathsf{Inf}_{\mathcal{I}}(S).
\end{aligned} \tag{7.10}$$

□

By Lemmas 7.3, 7.4, and 7.5, we have the following:

Theorem 7.6. *For any $S \subseteq V$,*

$$\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{H}}(\pi(S)) \leq \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1} \cdot \mathsf{Inf}_{\mathcal{G}}(S). \tag{7.11}$$

In summary, $\mathsf{Inf}_{\mathcal{H}}(\pi(\cdot))$ well approximates $\mathsf{Inf}_{\mathcal{G}}(\cdot)$ when $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1}$ is small, i.e., $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ is large. The factor $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ takes a value from zero to one, and it approaches one if, for every $j \in [\ell]$, the random graph sampled from $\mathcal{G}[C_j]$ is strongly connected with a high probability.

Monotonicity.

Apart from the above results, we here show the monotonicity of the size and the influence function of coarsened graphs with respect to the refinement relation. Let $\mathcal{G} = (V, E, p)$ be an influence graph and $\mathcal{P}_1$ and $\mathcal{P}_2$ be two partitions of V. Then, for $i = 1, 2$, let $\mathcal{H}_i = (W_i, F_i, q_i, \mathsf{w}_i)$ and $\pi_i : V \to W_i$ be a vertex-weighted influence graph and the correspondence mapping obtained from $\mathcal{G}$ by coarsening P_i, respectively. We obtain the following as a consequence of Definition 7.1 and Lemmas 7.3 and 7.4.

Theorem 7.7. *If $\mathcal{P}_1$ is a refinement of $\mathcal{P}_2$, then, $|W_1| \geq |W_2|$ and $|F_1| \geq |F_2|$.*

Proof. The proof is a direct consequence of Definition 7.1. □

In particular, coarsening $\mathcal{P} = \bigcup_{v \in V} \{\{v\}\}$, which is a refinement of any partition, yields the original $\mathcal{G}$; therefore, coarsening definitely does not increase the graph size.

Theorem 7.8. *If $\mathcal{P}_1$ is a refinement of $\mathcal{P}_2$, then, $\mathsf{Inf}_{\mathcal{H}_1}(\pi_1(S)) \leq \mathsf{Inf}_{\mathcal{H}_2}(\pi_2(S))$ for any $S \subseteq V$.*

Proof. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two intermediate influence graphs for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. It is easy to see that every influence probability of $\mathcal{I}_1$ is at least that of $\mathcal{I}_2$. Hence, the proof is a direct consequence of Lemmas 7.3 and 7.4. □

To sum up, it is desirable if the fineness of a vertex partition is tunable.

7.1.3 Creating a Partition to be Coarsened

Definition of r-robust SCCs.

We consider how to create a vertex partition which is desired to be coarsened. In the previous section, we demonstrated that

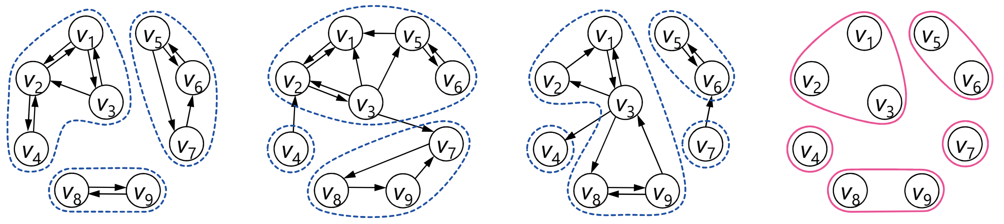
- coarsening $\{C_j\}_{j \in [\ell]}$ with large $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ preserves the influence on $\mathcal{G}$ (Theorem 7.6),
- the coarser a vertex partition is, the smaller a coarsened graph is (Theorem 7.7), and
- the finer a vertex partition is, the closer an influence function is to that of $\mathcal{G}$ (Theorem 7.8).

Unfortunately, computing the strongly connected reliability exactly is proven to be #P-hard [18, 181], and even approximate computation through sampling requires a number of random graph generations. Thus, we rely on the following intuition:

> if a vertex set is strongly connected in a small number of random graphs, then it is likely to be strongly connected in other random graphs.

Now, we formally define the notion of r-robust SCCs.

Definition 7.9 (r-robust SCC)**.** *Let $\mathcal{G} = (V, E, p)$ be an influence graph and $G_1 = (V, E_1), \ldots, G_r = (V, E_r)$ be r (fixed) random graphs sampled from $\mathcal{G}$. Then, a vertex set $C \subseteq V$ (or an induced influence subgraph $\mathcal{G}[C]$) is identified as an* r-robust SCC *with regard to $G_1, \ldots, G_r$ if*



(a) Subgraph G_1. (b) Subgraph G_2. (c) Subgraph G_3. (d) r-robust SCCs.

Figure 7.4: Example of r-robust SCCs with regard to three subgraphs. Each blue dotted curve corresponds an SCC and each red solid curve corresponds an r-robust SCC.

1. *for all $i \in [r]$, C is strongly connected in G_i, i.e., vertices in C are mutually reachable in G_i,*
2. *C is maximal.*

Example 7.10. *Figure 7.4 illustrates three subgraphs G_1, G_2, and G_3 with the same vertex set and r-robust SCCs with regard to G_1, G_2, and G_3. Each r-robust SCC is strongly connected in all the three subgraphs and maximal, e.g., adding a vertex v_4 to an r-robust SCC $\{v_1, v_2, v_3\}$ violates the former condition.*

r-robust SCCs have the following convenient characterization, whose proof is straightforward from Definition 7.9.

Theorem 7.11. *Let $\mathcal{C}_i$ be a partition consisting of all SCCs in G_i for each $i \in [r]$, and let $\mathcal{P}_r$ be a collection of all r-robust SCCs with regard to $G_1, \ldots, G_r$. Then, $\mathcal{P}_r = \bigwedge_{i\in[r]} \mathcal{C}_i$.*

Thus, a collection of r-robust SCCs forms a vertex partition, and it is easy to find, i.e., a partition consisting of all i-robust SCCs with regard to $G_1, \ldots, G_i$ can be computed incrementally as the meet of a partition consisting of all $(i-1)$-robust SCCs with regard to $G_1, \ldots, G_{i-1}$ and a partition consisting of SCCs in G_i. Note that an isolated vertex forms an r-robust SCC for any $r \geq 1$ since it is an SCC by itself. Furthermore, the value of r controls the fineness; a partition consisting of r-robust SCCs becomes finer as r increases.

Robustness.

Here, we give a justification of r-robust SCCs and discuss the choice of r from a theoretical point of view. We first claim that coarsening r-robust SCCs will not significantly affect the influence function. Rather than aiming to bound the strongly connected reliability of r-robust SCCs, we show the following:

Theorem 7.12. *For an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, let $\mathcal{P}_r$ be a (random) vertex partition consisting of r-robust SCCs with regard to r random graphs $G_1, \ldots, G_r$ sampled from $\mathcal{G}$. Then, for any vertex set $V' \subseteq V$, the probability that V' is contained inside some element in $\mathcal{P}_r$ is at least $\mathsf{Rel}(\mathcal{G}[V'])^r$.*

Proof. This event is equivalent to the event that V' is contained inside some SCC in G_i for all $i \in [r]$, which occurs with probability at least $\mathsf{Rel}(\mathcal{G}[V'])^r$. $\square$

The above fact implies that the "most part" of an r-robust SCC has a large strongly connected reliability. In our experiments, we set $r = 16$ so that

r-robust SCCs contain vertex sets with a large strongly connected reliability. For example, in Figure 7.2, a simple calculation according to Eq. (7.8) yields $\mathsf{Rel}(\mathcal{G}[C_1]) = 0.88848$; thus, some 16-robust SCC contains C_1 with probability at least $0.88848^{16} \approx 0.15$. Our experimental results in Section 7.4.5 verify this implication.

Density.

We then claim that coarsening r-robust SCCs leads to a powerful edge reduction in the sense that r-robust SCCs are dense in practice. Our spotlight for this purpose is on the core-fringe structure. An undirected graph is called *k-edge-connected* if it remains weakly connected after removing fewer than k edges. Akiba, Iwata, and Yoshida [9] observed that complex networks contain a large k-edge-connected subgraph for high k, e.g., $k = 100$. Then, the following theorem suggests that r-robust SCCs contain k-edge-connected subgraphs with high k.

Theorem 7.13. *Assume that $\mathcal{G} = (V, E, p)$ is undirected and the influence probability function p is a constant α, i.e., $p(e) = \alpha$ for every edge e. Let $\mathcal{P}_r$ be a (random) vertex partition consisting of r-robust SCCs with regard to r random graphs sampled from $\mathcal{G}$. If there exists a vertex set $V' \subseteq V$ such that the subgraph of (V, E) induced by V' is k-edge-connected and it holds that $\alpha^k = |V'|^{-(2+\delta)}$ for some $\delta > 0$, then, the probability that V' is contained inside some element in $\mathcal{P}_r$ is at least $(1 - |V'|^{-\delta}(1 + 2/\delta))^r$.*

Proof. By [97, Theorem 2.9], the random graph sampled from $\mathcal{G}[V']$ is connected with probability at least $1 - |V'|^{-\delta}(1 + 2/\delta)$. Thus, by combining with Theorem 7.12, we obtain the desired bound. □

Note that every vertex in a k-edge-connected subgraph is of degree at least k, and thus, r-robust SCCs are expected to contain such a dense vertex set. For example, suppose that there exists a 100-edge-connected subgraph of size $|V'| = 10^6$, $\alpha = 0.1$, and $r = 16$. Then, $\delta = 44/3$ satisfies $\alpha^k = |V'|^{-(2+\delta)}$, and thus, the desired probability is at least $(1 - 10^{-88}(1 + 3/22))^{16}$, which is close to one. Note that the above discussion holds for undirected graphs. However, in Section 7.4.5, we will experimentally verify that the directed subgraph induced by the largest r-robust SCC is much denser than the whole graph.

Monotonicity.

We finally show that the value of r controls the trade-off between size reduction and estimation accuracy. For an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, let $\mathcal{H}_r = (W_r, F_r, q_r, \mathsf{w}_r)$ and $\pi_r : V \to W_r$ be random variables representing a vertex-weighted influence graph and the correspondence mapping obtained from $\mathcal{G}$ by coarsening $\mathcal{P}_r$, respectively. As a consequence of Theorems 7.7, 7.8, and 7.11, we have the following:

Theorem 7.14. *The expected size of $\mathcal{H}_r$ increases as r increases, i.e.,*

$$\begin{aligned} \mathbf{E}[|W_1|] \leq \mathbf{E}[|W_2|] \leq \cdots \leq |V|, \\ \mathbf{E}[|F_1|] \leq \mathbf{E}[|F_2|] \leq \cdots \leq |E|, \end{aligned} \tag{7.12}$$

where the expectation is taken over the choice of random graphs.

Theorem 7.15. *The expected influence on $\mathcal{H}_r$ decreases as r increases, i.e., for any $S \subseteq V$,*

$$\mathbf{E}[\mathsf{Inf}_{\mathcal{H}_1}(\pi_1(S))] \geq \mathbf{E}[\mathsf{Inf}_{\mathcal{H}_2}(\pi_2(S))] \geq \cdots \geq \mathsf{Inf}_{\mathcal{G}}(S), \tag{7.13}$$

where the expectation is taken over the choice of random graphs.

7.2 Algorithm Implementations

In this section, we present a scalable algorithm for coarsening an influence graph. We provide an overview of our algorithm followed by its implementations according to the available space and analyze their time, space, and I/O complexities. We further describe how to parallelize those implementations.

7.2.1 Overview

Here, we give an overview of the proposed method. Given an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, the proposed method produces a (smaller) vertex-weighted influence graph $\mathcal{H} = (W, F, q, \mathsf{w})$. At a high level, the proposed method executes the following two stages.

- **First stage:** Create a partition $\mathcal{P}_r$ of V consisting of all r-robust SCCs with respect to r random graphs sampled from $\mathcal{G}$.
- **Second stage:** Construct an influence graph $\mathcal{H}$ obtained from $\mathcal{G}$ by coarsening $\mathcal{P}_r$.

In the following, we provide detailed implementations according to the available space, i.e., a speed-oriented implementation with linear space and a scalability-oriented implementation with sublinear space.

7.2.2 Linear-space Implementation

We first consider a situation wherein the entire input $\mathcal{G}$ can be stored in memory. Assume that $\mathcal{G}$ is already stored in memory, which actually requires $\mathcal{O}(|V| + |E|)$ I/O cost. Our implementation with linear space is presented in Algorithm 7.1, and the two stages are performed as follows.

First stage.

In the first stage (Algorithm 7.1, lines 1–5), we create a partition consisting of all r-robust SCCs with regard to r random graphs. Since generating r random graphs at once consumes $\mathcal{O}(r(|V| + |E|))$ space, we reduce the memory consumption to $\mathcal{O}(|V| + |E|)$ by sequential generation of random graphs and incremental updates.

More precisely, beginning with $\mathcal{P}_0 = \{V\}$, which is actually the partition consisting of the (single) 0-robust SCC, we repeat the following process to obtain $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$, r times. In the i^{th} process, we sample the i^{th} random graph G_i from $\mathcal{G}$, compute all of its SCCs $\mathcal{C}_i$, and identify a partition $\mathcal{P}_i$ consisting of all i-robust SCCs by computing the meet $\mathcal{P}_{i-1} \wedge \mathcal{C}_i$. Eventually, we obtain a partition $\mathcal{P}_r$ consisting of all r-robust SCCs with regard to $G_1, \ldots, G_r$.

Algorithm 7.1 Proposed algorithm with linear space.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$ and an integer $r$.
Output: a vertex-weighted influence graph $\mathcal{H} = (W, F, q, \mathsf{w})$ and the correspondence
    mapping $\pi : V \to W$.
1: $\mathcal{P}_0 \leftarrow \{V\}$.
2: for $i = 1$ to $r$ do
3:     $G_i \leftarrow$ a random graph sampled from $\mathcal{G}$.
4:     compute a partition $\mathcal{C}_i$ consisting of all SCCs in $G_i$.
5:     $\mathcal{P}_i \leftarrow \mathcal{P}_{i-1} \wedge \mathcal{C}_i$.
6: build $W, F, \pi, \mathsf{w}$ from $\mathcal{G}$ and $\mathcal{P}_r$ according to Definition 7.1.
7: $\mathsf{q}[c_x, c_y] \leftarrow 1$ for all $(c_x, c_y) \in F$.
8: for all $(u, v) \in E$ do
9:     if $(\pi(u), \pi(v)) \in F$ then
10:        $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p(u, v))$.
11: $q(e) \leftarrow 1 - \mathsf{q}[e]$ for all $e \in F$.
12: return $\mathcal{H} = (W, F, q, \mathsf{w})$ and $\pi$.
```

Second stage.

In the second stage (Algorithm 7.1, lines 6–12), we construct a coarsened influence graph $\mathcal{H}$.

Given an influence graph $\mathcal{G}$ and a partition $\mathcal{P}_r = \{C_j\}_{j \in [\ell]}$ of V, we naively construct the vertex set W, the edge set F, the vertex weights w, and the correspondence mapping π according to Definition 7.1. Here, the only non-trivial procedure is the calculation of the influence probability $q(e)$ for each e in F according to Eq. (7.3). We perform this with a single scan of (u, v)'s and $p(u, v)$'s using a hash table. Let q be a hash table storing influence probabilities for all edges in F with initial values of one. For each scanned edge (u, v) in E, we translate it onto $W \times W$, i.e., we compute $(\pi(u), \pi(v))$. If $(\pi(u), \pi(v))$ is not a self-loop, we update the entry of key $(\pi(u), \pi(v))$ in q as $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p(u, v))$. After scanning all edges, $q(e) = 1 - \mathsf{q}[e]$ holds for every edge e in F, and we finally output an influence graph $\mathcal{H} = (W, F, q, \mathsf{w})$.

Efficiency analysis.

Theorem 7.16. *Algorithm 7.1 requires $\mathcal{O}(r(|V| + |E|))$ time, $\mathcal{O}(|V| + |E|)$ space, and $\mathcal{O}(|V| + |E|)$ I/O cost.*

Proof. In the first stage, $\mathcal{O}(|V| + |E|)$ time is sufficient to generate a single random graph, find all of its SCCs, and compute the meet of two partitions. Thus, the whole process completes in $\mathcal{O}(r(|V| + |E|))$ time. During the i^{th} process, we maintain only the i^{th} random graph G_i, all of its SCCs $\mathcal{C}_i$, the $(i-1)^{\text{th}}$ partition $\mathcal{P}_{i-1}$, and the i^{th} partition $\mathcal{P}_i$; thus, $\mathcal{O}(|V| + |E|)$ space is sufficient.

In the second stage, $\mathcal{O}(|V| + |E|)$ time is required to construct $\mathcal{H}$ because each edge and influence probability are scanned once, and $\mathcal{H}$ consumes $\mathcal{O}(|W| + |F|)$ space, which is dominated by $\mathcal{O}(|V| + |E|)$.

Note that reading $\mathcal{G}$ from and writing $\mathcal{H}$ to secondary storage can be completed with $\mathcal{O}(|V| + |E|)$ I/O cost. $\square$

7.2.3 Sublinear-space Implementation

Next, we consider a situation wherein we cannot store the input graph $\mathcal{G}$ in memory. This situation happens frequently because simply storing all edges into

Algorithm 7.2 Proposed algorithm with sublinear space.

```
Input: disk $\mathbb{D}_\mathcal{G}$ storing $\mathcal{G} = (V, E, p)$ and an integer $r$.
Output: disk $\mathbb{D}_\mathcal{H}$ storing $\mathcal{H} = (W, F, q, \mathsf{w})$ and $\pi : V \to W$.
 1: $\mathcal{P}_0 \leftarrow \{V\}$.
 2: for $i = 1$ to $r$ do
 3:     for all $\langle u, v, p(u, v)\rangle$ read from disk $\mathbb{D}_\mathcal{G}$ do
 4:         write $(u, v)$ to disk $\mathbb{D}_{G_i}$ with probability $p(u, v)$.
 5:     run a disk-based SCC algorithm on $G_i$ stored in disk $\mathbb{D}_{G_i}$.
 6:     $\mathcal{C}_i \leftarrow$ a partition of $V$ consisting of all SCCs in $G_i$.
 7:     $\mathcal{P}_i \leftarrow \mathcal{P}_{i-1} \wedge \mathcal{C}_i$.
 8: build $W, \mathsf{w}, \pi$ from $\mathcal{G}$ and $\mathcal{P}_r$ according to Definition 7.1.
 9: write $W, \mathsf{w}, \pi$ to disk $\mathbb{D}_\mathcal{H}$.
10: $F' \leftarrow \{(c_x, c_y) \in F \mid w(c_x) > 1 \vee w(c_y) > 1\}$.
11: $\mathsf{q}[c_x, c_y] \leftarrow 1$ for all $(c_x, c_y) \in F'$.
12: for all $\langle u, v, p(u, v)\rangle$ read from disk $\mathbb{D}_\mathcal{G}$ do
13:     if $(\pi(u), \pi(v))$ is not a self-loop then
14:         if $(\pi(u), \pi(v)) \in F'$ then
15:             $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p(u, v))$.
16:         else                                         ▷ $(\pi(u), \pi(v)) \in F \setminus F'$.
17:             write $\langle \pi(u), \pi(v), p(u, v)\rangle$ to disk $\mathbb{D}_\mathcal{H}$.
18: for all $(c_x, c_y) \in F'$ do
19:     $q(c_x, c_y) \leftarrow 1 - \mathsf{q}[c_x, c_y]$.
20:     write $\langle c_x, c_y, q(c_x, c_y)\rangle$ to disk $\mathbb{D}_\mathcal{H}$.
21: return disk $\mathbb{D}_\mathcal{H}$.
```

memory consumes $8|E|$ bytes when a single edge is represented by a pair of 4-byte integers. To address this, we can reasonably assume that $\mathcal{G}$ is stored on a disk $\mathbb{D}_\mathcal{G}$ in the form of a sequence of triplets $\langle u, v, p(u, v)\rangle$, i.e., we can read $\langle u, v, p(u, v)\rangle$ sequentially in a certain order. Hereafter, $\mathbb{D}_I$ denotes a disk storing some information I (e.g., a graph).

Our implementation with sublinear space, presented in Algorithm 7.2, reduces its space complexity from $\mathcal{O}(|V| + |E|)$ to $\mathcal{O}(|V| + |F'|)$, where F' is defined as $F' = \{(c_x, c_y) \in F \mid \mathsf{w}(c_x) > 1 \vee \mathsf{w}(c_y) > 1\}$ and $|F'| \ll |F|$ in practice.

First stage.

In the first stage (Algorithm 7.2, lines 1–7), we construct a partition consisting of all r-robust SCCs using only $\mathcal{O}(|V|)$ space, rather than $\mathcal{O}(|V| + |E|)$ space. To achieve this space requirement, we write each random graph to a disk and run a disk-based SCC algorithm with $\mathcal{O}(|V|)$ space, e.g., [114].

Specifically, beginning with $P_0 = \{V\}$, we repeat the following process r times, similar to the linear-space implementation. In the i^{th} process, consuming constant space, we read each triplet $\langle u, v, p(u, v)\rangle$ from $\mathbb{D}_\mathcal{G}$ and write (u, v) on $\mathbb{D}_{G_i}$ with probability $p(u, v)$ one by one. Eventually, $\mathbb{D}_{G_i}$ stores all edges in G_i. Next, we apply a disk-based SCC algorithm to $\mathbb{D}_{G_i}$ and obtain a partition $\mathcal{C}_i$ consisting of all SCCs in G_i. We then compute $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$ and $\mathcal{C}_i$.

Second stage.

In the second stage (Algorithm 7.2, lines 8–21), we construct a coarsened influence graph $\mathcal{H}$ given $\mathcal{G}$ and $\mathcal{P}_r$ using $\mathcal{O}(|V|+|F'|)$ space, rather than $\mathcal{O}(|V|+|E|)$ space. Recall that $F' = \{(c_x, c_y) \in F \mid \mathsf{w}(c_x) > 1 \vee \mathsf{w}(c_y) > 1\}$. A key factor of this space reduction is that $q(\pi(u), \pi(v)) = p(u, v)$ holds if $(\pi(u), \pi(v)) \in F \setminus F'$. Thus, we

Algorithm 7.3 Parallel implementation.

```
Input: $\mathcal{G}$, $r$, and the number of threads $T$.
Output: $\mathcal{H}$ and $\pi$.
1: for $t = 1$ to $T$ do
2:     $r_t \leftarrow \lfloor \frac{r+t-1}{T} \rfloor$.                    ▷ $\sum_{1 \le t \le T} r_t = r$
3:     launch the $t^{\text{th}}$ thread to execute CreatePartition$(t, r_t)$.
4: wait until all the threads have completed.
5: $\mathcal{P}_r \leftarrow \bigwedge_{t \in [T]} \mathcal{P}(t)$.
6: construct $\mathcal{H}$ and $\pi$ for $\mathcal{G}$ and $\mathcal{P}_r$ according to Def. 7.1.
7: return $\mathcal{H}$ and $\pi$.

8: procedure CreatePartition$(t, r_t)$
9:     $\mathcal{P}(t) \leftarrow$ a partition consisting of all $r_t$-robust SCCs.
```

do not need to store an entry with key $(\pi(u), \pi(v))$ in $F \setminus F'$ in a hash table.

More precisely, sequentially scanning each triplet $\langle u, v, p(u, v)\rangle$ from $\mathbb{D}_{\mathcal{G}}$, we update the entry $\mathsf{q}[\pi(u), \pi(v)]$ in the same manner as the linear-space implementation if $(\pi(u), \pi(v)) \in F'$; otherwise, we immediately write $\langle \pi(u), \pi(v), p(u, v)\rangle$ to $\mathbb{D}_{\mathcal{H}}$. After the scan is complete, we write $\langle c_x, c_y, q(c_x, c_y)\rangle$ to $\mathbb{D}_{\mathcal{H}}$ for each edge (c_x, c_y) in F'.

Our space reduction technique is effective if $\mathsf{w}(c_j) = 1$ holds for most $c_j \in W$. This is the case in reality because vertices in the *tree-like* fringe part are rarely strongly connected. In fact, our experimental results demonstrate 90% reduction of memory usage compared to the linear-space implementation.

Efficiency analysis.

Theorem 7.17. *Algorithm 7.2 requires $\mathcal{O}(r(|V| + |E| + \mathrm{t}_{\mathcal{A}}(V, E)))$ time, $\mathcal{O}(|V| + |F'|)$ space, and $\mathcal{O}(r(|V| + |E| + \mathrm{io}_{\mathcal{A}}(V, E)))$ I/O cost, where $\mathrm{t}_{\mathcal{A}}(V, E)$ and $\mathrm{io}_{\mathcal{A}}(V, E)$ are the time and I/O complexities of a disk-based SCC algorithm $\mathcal{A}$ given a graph (V, E), respectively.*

Proof. In the first stage, the whole process obviously completes in $\mathcal{O}(r(|V| + |E| + \mathrm{t}_{\mathcal{A}}(V, E)))$ time and $\mathcal{O}(r(|V| + |E| + \mathrm{io}_{\mathcal{A}}(V, E)))$ I/O cost. During the i^{th} process, we maintain only $\mathcal{C}_i$, $\mathcal{P}_{i-1}$, and $\mathcal{P}_i$, which require $\mathcal{O}(|V|)$ space, and a disk-based SCC algorithm $\mathcal{A}$ is assumed to consume $\mathcal{O}(|V|)$ space. Thus, $\mathcal{O}(|V|)$ space is sufficient.

In the second stage, $\mathcal{O}(|V| + |E|)$ time is required to construct $\mathcal{H}$ because $\mathbb{D}_{\mathcal{G}}$ is scanned once. W, w, and π consume $\mathcal{O}(|W|) = \mathcal{O}(|V|)$ space, F' and q consume $\mathcal{O}(|F'|)$ space, and writing all information involving $\mathcal{H}$ requires $\mathcal{O}(|W| + |F|)$ I/O cost, which is dominated by $\mathcal{O}(|V| + |E|)$. □

7.2.4 Parallelization

Thanks to its flexibility, we can easily extend the proposed algorithm so as to support *parallel processing*.

Let us begin with an overview of our parallel algorithm, which is applicable to both *shared-memory* and *distributed-memory* systems. Essentially, we simply parallelize the first stage. This significantly gains the scalability of the sublinear-space implementation (Algorithm 7.2), since the first stage runs a disk-based SCC algorithm many times, which is quite expensive (though depends on which method is employed).

Our parallel algorithm is presented in Algorithm 7.3. In addition to the ordinary input $\mathcal{G}$ and r, we are given the number T of threads to be created.

Algorithm 7.4 Proposed influence estimation framework.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a vertex set $S$, a coarsened graph $\mathcal{H} = (W, F, q, \mathsf{w})$, the correspondence mapping $\pi : V \to W$, an influence estimation algorithm $\mathcal{A}$.
1: $T \leftarrow \pi(S)$.
2: $\mathsf{Inf}_{\mathrm{out}} \leftarrow$ (approximately) compute $\mathsf{Inf}_{\mathcal{H}}(T)$ using $\mathcal{A}$.
3: return $\mathsf{Inf}_{\mathrm{out}}$.
```

For each t in $[T]$, we assign the number r_t of random graphs that the t^{th} thread will create in a balanced way so that $\sum_{i \in [T]} r_t = r$ and $|r_{t_1} - r_{t_2}| \leq 1$ for any $t_1, t_2 \in [T]$. Then, we launch the t^{th} thread and execute CreatePartition(t, r_t), which creates a partition $\mathcal{P}(t)$ that consists of all r_t-robust SCCs with regard to r_t random graphs. When all the T threads have been completed, we have T partitions $\mathcal{P}(1), \ldots, \mathcal{P}(T)$. By computing the meet of them, we obtain a partition $\mathcal{P}_r$ consisting of all r-robust SCCs. We finally construct $\mathcal{H}$ and π according to Definition 7.1.

7.3 Frameworks for Scaling-up Influence Analysis

Here, we present general frameworks that exploit our coarsened graphs for accelerating existing algorithms for influence estimation and influence maximization. Throughout this section, $\mathcal{G} = (V, E, p)$ is an input influence graph, $\{C_j\}_{j \in [\ell]}$ is a partition of V, $\mathcal{H} = (W, F, q, \mathsf{w})$ is a vertex-weighted influence graph obtained from $\mathcal{G}$ by coarsening $\{C_j\}_{j \in [\ell]}$, and $\pi : V \to W$ is a correspondence mapping. Note that $\mathcal{H}$ is not limited to be an output of our algorithm but a coarsened graph obtained by coarsening any partition.

7.3.1 Framework for Influence Estimation

Recall that the influence estimation problem requires the computation of $\mathsf{Inf}_{\mathcal{G}}(S)$ given a seed set $S \subseteq V$. To improve efficiency, our influence estimation framework runs an existing algorithm on $\mathcal{H}$ rather than directly running on $\mathcal{G}$. Specifically, given $\mathcal{H}$, π, S, and an influence estimation algorithm $\mathcal{A}$, our framework shown in Algorithm 7.4 translates S onto W via π, i.e., it computes $T = \pi(S)$, (approximately) computes $\mathsf{Inf}_{\mathcal{H}}(T)$ using $\mathcal{A}$, and returns an obtained estimation $\mathsf{Inf}_{\mathrm{out}}$. Since $\mathcal{H}$ is smaller than $\mathcal{G}$, processing on $\mathcal{H}$ is expected to be more efficient than on $\mathcal{G}$. Moreover, we bound the relative error of our framework's estimation as follows.

Theorem 7.18. *For a non-empty seed set $S \subseteq V$, let $\mathsf{Inf}_{\mathrm{out}}$ be an output of Algorithm 7.4, i.e., an estimation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$. If $\mathsf{Inf}_{\mathrm{out}}$ is a $(1 \pm \epsilon)$-approximation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$, then, the relative error between $\mathsf{Inf}_{\mathcal{G}}(S)$ and $\mathsf{Inf}_{\mathrm{out}}$ is bounded by*

$$-\epsilon \leq \frac{\mathsf{Inf}_{\mathrm{out}} - \mathsf{Inf}_{\mathcal{G}}(S)}{\mathsf{Inf}_{\mathcal{G}}(S)} \leq \frac{1 + \epsilon}{\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])} - 1. \tag{7.14}$$

Proof. Remark that $\mathsf{Inf}_{\mathcal{G}}(S) > 0$ since $S \neq \emptyset$. Hence, The proof is a direct consequence of Lemmas 7.3, 7.4, and 7.5. □

For example, consider applying our framework to a naive simulation method. In our framework, the naive method simulates the diffusion process from $\pi(S)$ on $\mathcal{H}$ and takes the average weight of active vertices. Since $\mathcal{H}$ is smaller than $\mathcal{G}$, the diffusion process in $\mathcal{H}$ terminates earlier than in $\mathcal{G}$. Furthermore, $\mathsf{Inf}_{\mathrm{out}}$

Algorithm 7.5 Proposed influence maximization framework.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$, a seed size $k$, a coarsened graph $\mathcal{H} = (W, F, q, \mathsf{w})$, the correspondence mapping $\pi : V \to W$, an influence maximization algorithm $\mathcal{A}$.
1: compute a solution $T$ of size $k$ that (approximately) maximizes $\mathsf{Inf}_{\mathcal{H}}(\cdot)$ using $\mathcal{A}$.
2: $S_{\mathrm{out}} \leftarrow$ select random $S$ such that $\pi(S) = T$.
3: return $S_{\mathrm{out}}$.
```

is a $(1 \pm \epsilon)$-approximation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$ with high probability for a sufficiently large number of simulations; thus, Theorem 7.18 can be applied. Remark that when an output of our algorithm with r is used as $\mathcal{H}$, we can expect that the estimation accuracy improves as r increases due to Theorem 7.15.

7.3.2 Framework for Influence Maximization

Recall that the influence maximization problem seeks to select a seed set S of size k with the maximum influence on $\mathcal{G}$. Similar to the influence estimation framework described above, our influence maximization framework applies an existing method to $\mathcal{H}$. More precisely, given $\mathcal{H}$, π, a seed size k, and an influence maximization algorithm $\mathcal{A}$, our framework shown in Algorithm 7.5 first (approximately) solves the influence maximization problem for $\mathcal{H}$ and k using $\mathcal{A}$. Let $T \subseteq W$ be an obtained solution of size k. Then, in contrast to influence estimation, we translate T onto $\mathcal{G}$. To this end, we simply convert each vertex w in T to a random vertex v such that $\pi(v) = w$ and return an obtained set S_{out}. Processing on $\mathcal{H}$ is faster than on $\mathcal{G}$, and the following theorem gives an approximation ratio of Algorithm 7.5.

Theorem 7.19. *For an integer k, let S_{out} be an output of Algorithm 7.5. If T is an α-approximate solution of size k for $\mathcal{H}$, then, S_{out} is an $(\alpha \cdot \prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j]))$-approximate solution of size k for $\mathcal{G}$.*

Proof. Let S^* and T^* be the optimal solutions of size k for $\mathcal{G}$ and $\mathcal{H}$, respectively. Since Lemmas 7.3 and 7.4 ensure $\mathsf{Inf}_{\mathcal{H}}(T^*) \geq \mathsf{Inf}_{\mathcal{G}}(S^*)$, we have

$$\mathsf{Inf}_{\mathcal{H}}(\pi(S_{\mathrm{out}})) = \mathsf{Inf}_{\mathcal{H}}(T) \geq \alpha \cdot \mathsf{Inf}_{\mathcal{H}}(T^*) \geq \alpha \cdot \mathsf{Inf}_{\mathcal{G}}(S^*).$$

Then, applying Lemmas 7.3 and 7.5 yields the following:

$$\mathsf{Inf}_{\mathcal{G}}(S_{\mathrm{out}}) \geq \left(\prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j]) \right) \cdot \alpha \cdot \mathsf{Inf}_{\mathcal{G}}(S^*). \tag{7.15}$$

□

For example, consider applying our framework to RIS-based algorithms. In our framework, to build sketches, sketching algorithms repeatedly perform reverse simulations on $\mathcal{H}$ starting from a vertex selected from W with probability proportional to its weight. Then, they greedily select a vertex that intersects the maximum number of sketches and construct a seed set T of size k for $\mathcal{H}$. They finally return a random S such that $\pi(S) = T$. Here, T is a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution to the optimal solution of size k for $\mathcal{H}$ with high probability for some parameter ϵ; thus, S_{out} is a $((1-\mathrm{e}^{-1}-\epsilon)\cdot\prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j]))$-approximate solution of size k for $\mathcal{G}$.

Table 7.1: Datasets used in Chapter 7. (d) and (u) denote "directed" and "undirected," respectively.

name	type	$\|V\|$	$\|E\|$
ca-GrQc	collab.(u)	5,242	28,968
ca-HepTh	collab.(u)	9,877	51,946
wiki-Vote	social(d)	7,115	103,689
ca-HepPh	collab.(u)	12,008	236,978
soc-Epinions1	social(d)	75,879	508,837
soc-Slashdot0922	social(d)	82,168	870,161
web-NotreDame	web(d)	325,729	1,469,679
ego-Twitter	social(d)	81,306	1,768,135
loc-Gowalla	social(u)	196,591	1,900,654
web-Stanford	web(d)	281,903	2,312,497
wiki-Talk	commu.(d)	2,394,385	5,021,410
web-Google	web(d)	875,713	5,105,039
com-Youtube	social(u)	1,134,890	5,975,248
web-BerkStan	web(d)	685,230	7,600,595
higgs-twitter	social(d)	456,626	14,855,819
soc-Pokec	social(d)	1,632,803	30,622,564
soc-LiveJournal1	social(d)	4,847,571	68,475,391
com-Orkut	social(u)	3,072,441	234,370,166
twitter-2010	social(d)	41,652,230	1,468,364,884
com-Friendster	social(u)	65,608,366	3,612,134,270
uk-2007-05	web(d)	105,218,569	3,717,169,969
ameblo	web(d)	272,687,914	6,910,266,107

7.4 Experiments

We conducted experiments using real-world networks to demonstrate the effectiveness, efficiency, and scalability of our algorithm and frameworks.

7.4.1 Setup

Datasets.

We used real-world social networks and web graphs. Table 7.1 summarizes the basic statistics of each network. twitter-2010 and uk-2007-05 were downloaded from Laboratory for Web Algorithmics (LAW) [25, 26], which is maintained by Paolo Boldi, Andrea Marino, Corrado Monti, Massimo Santini, and Sebastiano Vigna. ameblo is a crawled web graph of the `ameblo.jp` domain provided by Yahoo Japan Corporation, which was used to demonstrate the scalability of our algorithm. The other networks were downloaded from Stanford Network Analysis Project Datasets (SNAP) [116]. We have applied the preprocessing to each network described in Section 4.2.

Influence probability settings.

To investigate the behavior of the algorithms under different probability settings, we employed the $\textsc{exp}_{0.1}$, $\textsc{tri}$, $\textsc{uc}_{0.1}$, and $\textsc{iwc}$ settings introduced in Chapter 4. Since we have similar trends on $\textsc{uc}_{0.1}$ and $\textsc{exp}_{0.1}$, the results for $\textsc{uc}_{0.1}$ are deferred to Appendix C. The results for the $\textsc{iwc}$ are also deferred to Appendix C.

Parameter settings.

Our algorithm has a parameter r that controls the trade-off between size reduction and estimation accuracy. Unless otherwise specified, we set $r = 16$. This choice will be justified in Section 7.4.6 in terms of accuracy. Note that the sublinear-space implementation (Algorithm 7.2) uses an existing disk-based SCC algorithm proposed by Laura and Santaroni [114].

Environments.

Unless otherwise specified, the experiments were conducted on a Linux server with an Intel Xeon E5-2690 2.90GHz CPU and 256GB memory. Experiments for ameblo were conducted on a Linux server with an Intel Xeon E5-2630L 2.00GHz CPU and 256GB memory because this dataset is only available on this machine at the moment. The proposed algorithm and frameworks were implemented in C++ and compiled using g++v4.6.3 with the -O2 option.

7.4.2 Scalability Evaluation

First, we examined the scalability of the proposed algorithm (Algorithms 7.1 and 7.2). Table 7.2 reports the run times and memory usages of the two implementations for each setting.

Note that the run time of the linear-space implementation does not include the time required to read the input graph from and write the coarsened graph to secondary storage. Obviously, both the run time and memory usage scale linearly to the graph size and are robust against the probability setting. The linear-space implementation took approximately one hour and required roughly one hundred gigabytes for billion-edge graphs, but it ran out-of-memory for ameblo because the input and output graphs cannot fit in memory at the same time. Compared to the linear-space implementation, the sublinear-space implementation reduced the memory usage by 90% and ran on ameblo, while it is ten times slower.

Figures 7.5 and 7.6 plot the run time and memory usage of the two implementations against the value of r, respectively. As expected from the efficiency analyses (Theorems 7.16 and 7.17), the run time of both implementations scales linearly to r and the memory usage of linear-space implementation is not affected by r. Note that the memory usage of the sublinear-space implementation is also robust against the value of r.

Table 7.2: Run time and memory usage of the proposed algorithm under $\mathrm{EXP}_{0.1}$ and TRI.

	exponential ($\mathrm{EXP}_{0.1}$)				**trivalency** (TRI)			
	linear space (Alg. 7.1)		sublinear space (Alg. 7.2)		linear space (Alg. 7.1)		sublinear space (Alg. 7.2)	
dataset	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**
ca-GrQc	0.02 s	4 MB	0.19 s	2 MB	0.02 s	4 MB	0.18 s	2 MB
ca-HepTh	0.04 s	6 MB	0.36 s	2 MB	0.03 s	6 MB	0.31 s	2 MB
wiki-Vote	0.04 s	8 MB	0.69 s	2 MB	0.03 s	9 MB	0.63 s	2 MB
ca-HepPh	0.09 s	14 MB	1.46 s	3 MB	0.07 s	16 MB	1.34 s	3 MB
soc-Epinions1	0.38 s	38 MB	3.42 s	10 MB	0.33 s	41 MB	3.26 s	8 MB
soc-Slashdot0922	0.54 s	57 MB	5.57 s	13 MB	0.44 s	61 MB	5.28 s	11 MB
web-NotreDame	1.77 s	122 MB	10.86 s	28 MB	1.67 s	125 MB	10.57 s	26 MB
ego-Twitter	0.80 s	103 MB	11.66 s	14 MB	0.60 s	111 MB	10.68 s	11 MB
loc-Gowalla	1.29 s	138 MB	12.67 s	26 MB	1.06 s	132 MB	11.90 s	22 MB
web-Stanford	1.86 s	157 MB	18.06 s	26 MB	1.51 s	157 MB	14.92 s	24 MB
wiki-Talk	42.31 s	603 MB	57.37 s	270 MB	25.67 s	588 MB	55.47 s	249 MB
web-Google	7.93 s	344 MB	43.36 s	75 MB	7.00 s	343 MB	35.64 s	69 MB
com-Youtube	14.59 s	452 MB	44.89 s	147 MB	12.22 s	472 MB	44.25 s	121 MB
web-BerkStan	6.44 s	398 MB	56.23 s	70 MB	5.71 s	406 MB	49.47 s	57 MB
higgs-twitter	7.64 s	530 MB	100.35 s	85 MB	5.00 s	686 MB	94.96 s	63 MB
soc-Pokec	35.20 s	1,280 MB	224.19 s	237 MB	30.49 s	1,489 MB	216.44 s	164 MB
soc-LiveJournal1	94.58 s	2,966 MB	507.54 s	677 MB	83.45 s	3,329 MB	473.83 s	480 MB
com-Orkut	122.34 s	6,288 MB	1,527.21 s	523 MB	103.55 s	7,183 MB	1,452.29 s	576 MB
twitter-2010	1,762.93 s	50,801 MB	11,522.38 s	5,635 MB	1,209.89 s	58,467 MB	10,354.66 s	5,325 MB
com-Friendster	3,964.05 s	101,398 MB	26,423.50 s	7,683 MB	3,120.62 s	113,337 MB	23,929.75 s	8,315 MB
uk-2007-05	3,105.88 s	136,659 MB	29,540.45 s	10,841 MB	2,619.00 s	166,670 MB	25,345.31 s	8,087 MB
ameblo	OOM	OOM	35,761.37 s	27,502 MB	OOM	OOM	16,926.93 s	20,684 MB

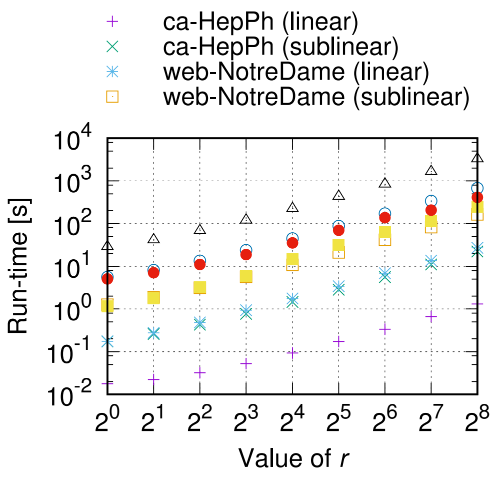


Figure 7.5: Run time with varying r ($\textsc{exp}_{0.1}$).

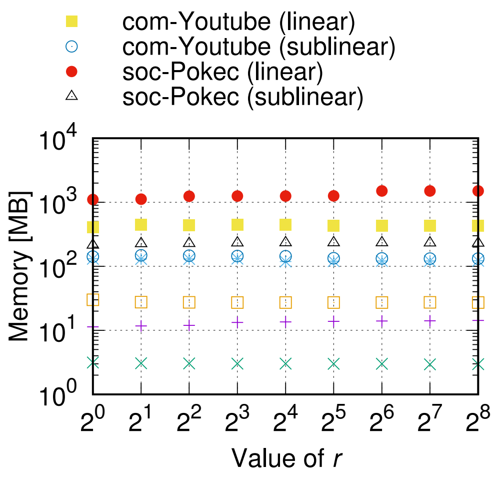


Figure 7.6: Memory usage with varying r ($\textsc{exp}_{0.1}$).

7.4.3 Power of Parallelization

We here demonstrate the effectiveness of our parallelization (Algorithm 7.3). The implementation and environmental setup required in this subsection have been performed by Tomohiro Sonobe, a co-author of the extended abstract [158] published in SIGMOD 2017. Our parallelization works for two systems below. We first adopt a *shared-memory* system, i.e., there exists a global address space, and parallel threads read from it and write to it concurrently. We used OpenMP for parallelization and ran the OpenMP implementation on the same environment as that described in Section 7.4. Table 7.3 shows the run times of the shared-memory parallel implementation, with linear and sublinear space, and with a different number of threads (1, 4, and 16). Table 7.3 indicates that our parallel algorithm exhibited good scalability on both linear- and sublinear-space implementations, around 3–4 times speed-up with 16 threads.

We then consider a *distributed-memory* system, i.e., multiple processes that run on multiple machines send and receive data through Message Passing Interface (MPI). One master process constructs r random graphs and sends them to slave processes through MPI. Each slave, given a number r_t, creates a partition that consists of all r_t-robust SCCs and sends it back to the master. Finally, the master process computes $\mathcal{P}_r$ and constructs a coarsened influence graph. We used OpenMPI for message passing and ran the OpenMPI implementation on three Linux machines with the following specifications: (1) Intel Core i7-6850K 3.6GHz and 128GB RAM, (2) Intel Core i7-5960X 3.0GHz and 64GB RAM, and (3) Intel Xeon E5-1603 2.8GHz and 64GB RAM. All the machines were placed on the same LAN over 1000BASE-T Ethernet. Table 7.3 shows the run time of the distributed-memory parallel algorithm with 16 slaves. Note that we were not able to use this distributed environment for ameblo dataset due to its availability. For the linear-space implementation, communication overheads by MPI dominate the processing time, resulting in the same or even worse performance than the sequential algorithm (Algorithm 7.1). However, thanks to the decentralization of disk access to multiple computing nodes, the distributed-memory algorithm with sublinear space achieved better performance than the shared-memory one, up to six times speed-up.

Table 7.3: Run time of our parallel implementations for $\text{EXP}_{0.1}$.

	linear space (Alg. 7.3 with impl. of Alg. 7.1)				sublinear space (Alg. 7.3 with impl. of Alg. 7.2)			
	shared			**distributed**	**shared**			**distributed**
dataset	1 thread	4 threads	16 threads	16 threads	1 thread	4 threads	16 threads	16 threads
ca-GrQc	0.02 s	0.05 s	0.03 s	0.05 s	0.19 s	0.28 s	2.46 s	0.04 s
ca-HepTh	0.04 s	0.11 s	0.06 s	0.09 s	0.36 s	0.29 s	0.18 s	0.07 s
wiki-Vote	0.04 s	0.05 s	0.03 s	0.10 s	0.69 s	0.50 s	0.29 s	0.11 s
ca-HepPh	0.09 s	0.19 s	0.11 s	0.21 s	1.46 s	1.20 s	0.77 s	0.23 s
soc-Epinions1	0.38 s	0.69 s	0.35 s	0.63 s	3.42 s	2.19 s	1.16 s	0.66 s
soc-Slashdot0922	0.54 s	1.01 s	0.63 s	1.28 s	5.57 s	3.88 s	1.79 s	1.01 s
web-NotreDame	1.77 s	3.43 s	1.73 s	2.27 s	10.86 s	8.24 s	3.85 s	2.31 s
ego-Twitter	0.80 s	1.24 s	0.65 s	1.74 s	11.66 s	6.81 s	2.97 s	2.09 s
loc-Gowalla	1.29 s	2.30 s	0.87 s	2.23 s	12.67 s	6.93 s	4.11 s	2.69 s
web-Stanford	1.86 s	3.60 s	1.38 s	3.74 s	18.06 s	11.51 s	4.24 s	3.05 s
wiki-Talk	42.31 s	25.70 s	12.39 s	24.26 s	57.37 s	42.09 s	22.08 s	18.35 s
web-Google	7.93 s	8.95 s	5.15 s	8.76 s	43.36 s	22.59 s	10.51 s	9.12 s
com-Youtube	14.59 s	11.99 s	6.95 s	14.60 s	44.89 s	29.03 s	13.72 s	13.37 s
web-BerkStan	6.44 s	7.87 s	4.31 s	12.14 s	56.23 s	29.75 s	13.78 s	9.91 s
higgs-twitter	7.64 s	7.76 s	3.58 s	18.38 s	100.35 s	47.15 s	23.43 s	15.26 s
soc-Pokec	35.20 s	21.74 s	10.75 s	37.10 s	224.19 s	102.73 s	52.09 s	40.85 s
soc-LiveJournal1	94.58 s	60.97 s	33.45 s	102.64 s	507.54 s	244.71 s	127.36 s	106.69 s
com-Orkut	122.34 s	59.82 s	38.29 s	209.90 s	1,527.21 s	657.50 s	349.71 s	217.32 s
twitter-2010	1,762.93 s	1,017.31 s	611.98 s	1,751.02 s	11,522.38 s	4,923.69 s	2,619.67 s	1,928.31 s
com-Friendster	3,964.05 s	2,171.24 s	1,142.70 s	4,381.44 s	26,423.50 s	11,333.60 s	5,843.53 s	4,083.38 s
uk-2007-05	3,105.88 s	2,164.21 s	1,035.69 s	4,425.94 s	29,540.45 s	12,856.15 s	6,328.57 s	4,494.19 s
ameblo	OOM	OOM	OOM	–	35,761.37 s	23,074.51 s	7,612.53 s	–

7.4.4 Graph Size Reduction

Next, we investigated the effect of the proposed algorithm on graph size. Table 7.4 shows the numbers of vertices and edges and the corresponding reduction ratio of each coarsened graph.

The proposed method reduced the number of edges to 3.6%–72.4% (EXP$_{0.1}$) and 15.4%–95.6% (TRI). Note that the edge reduction ratio is higher than the vertex reduction ratio because we extracted dense r-robust SCCs, as will be discussed in Section 7.4.5. Figure 7.7 shows the edge reduction ratio with various values of r. We can observe that the number of edges in the coarsened graphs logarithmically increases with r.

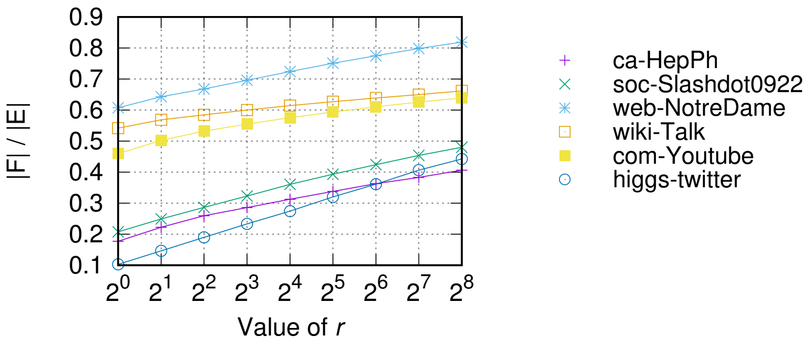


Figure 7.7: Edge reduction ratio with varying r (EXP$_{0.1}$).

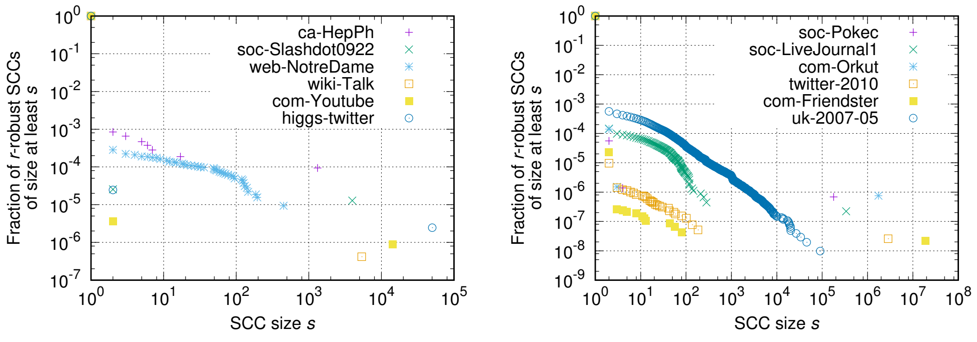
Table 7.4: Effect of the proposed algorithm on graph size under EXP$_{0.1}$ and TRI. V and E denote vertex and edge sets of an input graph, and W and F denote vertex and edge sets of a coarsened graph, respectively.

	exponential (EXP$_{0.1}$)				**trivalency** (TRI)			
dataset	$\|W\|$	$\|W\|/\|V\|$	$\|F\|$	$\|F\|/\|E\|$	$\|W\|$	$\|W\|/\|V\|$	$\|F\|$	$\|F\|/\|E\|$
ca-GrQc	5,174	98.7%	24,778	85.5%	5,242	100.0%	28,968	100.0%
ca-HepTh	9,866	99.9%	51,370	98.9%	9,877	100.0%	51,946	100.0%
wiki-Vote	6,956	97.8%	72,034	69.5%	7,115	100.0%	103,689	100.0%
ca-HepPh	10,656	88.7%	74,036	31.2%	11,592	96.5%	127,618	53.9%
soc-Epinions1	73,884	97.4%	220,511	43.3%	75,609	99.6%	410,436	80.7%
soc-Slashdot0922	78,202	95.2%	313,526	36.0%	81,450	99.1%	609,007	70.0%
web-NotreDame	321,196	98.6%	1,064,305	72.4%	324,525	99.6%	1,262,801	85.9%
ego-Twitter	72,748	89.5%	705,268	39.9%	80,431	98.9%	1,472,832	83.3%
loc-Gowalla	189,586	96.4%	1,028,844	54.1%	195,686	99.5%	1,507,150	79.3%
web-Stanford	281,638	99.9%	2,214,140	95.7%	281,903	100.0%	2,312,497	100.0%
wiki-Talk	2,389,034	99.8%	3,084,856	61.4%	2,392,612	99.9%	3,677,722	73.2%
web-Google	875,680	100.0%	5,099,721	99.9%	875,713	100.0%	5,105,039	100.0%
com-Youtube	1,120,463	98.7%	3,437,134	57.5%	1,132,305	99.8%	4,471,436	74.8%
web-BerkStan	679,666	99.2%	6,359,491	83.7%	684,321	99.9%	7,389,147	97.2%
higgs-twitter	406,481	89.0%	4,077,314	27.4%	446,601	97.8%	9,891,098	66.6%
soc-Pokec	1,453,548	89.0%	13,292,116	43.4%	1,626,472	99.6%	29,277,656	95.6%
soc-LiveJournal1	4,498,127	92.8%	28,925,742	42.2%	4,800,660	99.0%	53,523,063	78.2%
com-Orkut	1,330,379	43.3%	8,446,726	3.6%	2,474,681	80.5%	64,035,002	27.3%
twitter-2010	38,805,787	93.2%	345,073,261	23.5%	40,746,064	97.8%	592,434,240	40.3%
com-Friendster	46,709,541	71.2%	171,230,294	4.7%	56,750,334	86.5%	557,214,040	15.4%
uk-2007-05	102,402,599	97.3%	1,553,084,998	41.8%	104,335,244	99.2%	2,578,741,487	69.4%
ameblo	271,042,159	99.4%	5,478,111,790	79.3%	272,624,016	99.9%	6,836,460,512	98.9%

7.4.5 Analyzing Extracted r-robust SCCs

Size distribution and density.

Third, we analyzed the structural properties of the extracted r-robust SCCs. Figure 7.8 illustrates the size distribution of the extracted r-robust SCCs ($\textsc{exp}_{0.1}$). As can be seen, a giant r-robust SCC exists, e.g., for soc-LiveJournal1 and com-Friendster, the largest r-robust SCCs are of size 3,370,90 and 18,897,527, respectively, while the second largest is of size 281 and 80, respectively. Furthermore, as discussed in Section 7.1.3, these components are dense; for soc-LiveJournal1 and com-Friendster, the average degree of the subgraph induced by the largest r-robust SCC is 57.9 and 154.8, while the average degree of the whole graph is 14.1 and 55.1, respectively. Thus, r-robust SCCs provide a vertex set whose coarsening leads to powerful edge reduction. We also observe that 99.9% of r-robust SCCs are of size one, which indicates that $|F'| \ll |F|$.



(a) Smaller graphs (b) Larger graphs

Figure 7.8: Size distribution of r-robust SCCs ($\textsc{exp}_{0.1}$).

Robustness.

Here, we investigated the robustness of the extracted r-robust SCCs. For a graph G, we define the *maximum SCC rate* as the size of the largest SCC in G divided by the number of vertices in G. This rate takes one if G is strongly connected in whole. Then, we consider the distribution of the maximum SCC rate of $\mathcal{G}[C_{\max}]$, where $C_{\max}$ is the largest r-robust SCC. To this end, we evaluate the probability that "the maximum SCC rate of the random graph sampled from $\mathcal{G}[C_{\max}]$ is more than a threshold θ." We estimated this value by sampling 10,000 random graphs from $\mathcal{G}[C_{\max}]$.

Figure 7.9 shows the cumulative distribution of the maximum SCC rate. We can see that the most part of $C_{\max}$ is strongly connected with a high probability. For example, 93% of the vertices in $C_{\max}$ of soc-Slashdot0922 are strongly connected in the random graph with probability 0.9. This coincides with the discussion in Section 7.1.3.

7.4.6 Evaluating Influence Estimation Framework

Fourth, we evaluated our influence estimation framework (Algorithm 7.4). We applied our framework to the naive Monte-Carlo simulation method *MC*, which repeatedly simulates the diffusion process and takes the average number (weight) of active vertices.

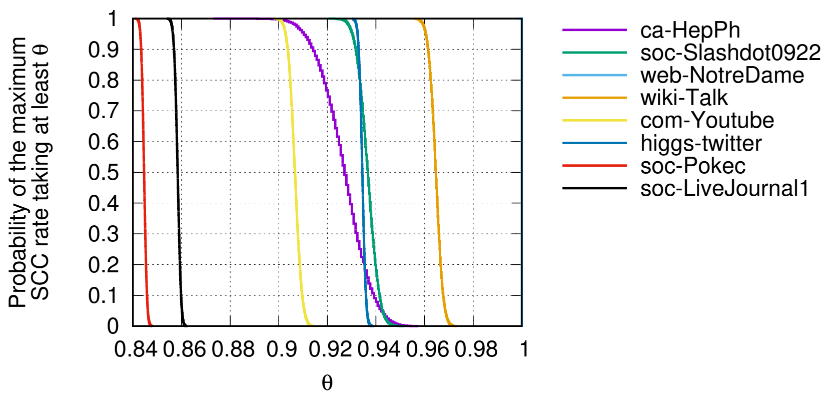


Figure 7.9: Cumulative distribution of the maximum SCC rate of the subgraph induced by the largest r-robust SCC ($\textsc{exp}_{0.1}$).

Efficiency improvement.

We first observe the efficiency improvement. To this end, we randomly sampled 10,000 vertices from the input graph and ran plain *MC* and our framework with *MC* to estimate the influence of each sampled vertex. Table 7.5 shows the total run time of each method. Our framework drastically reduced the computation time to up to 3.5%. As the simulation cost is dominated by edge traversal costs, the time reduction ratio is roughly equal to the edge reduction ratio.

Estimation accuracy.

We now examine the estimation accuracy of our framework. For each vertex v in the input graph, our framework ran *MC* on the coarsened graph, i.e., it simulated the diffusion process 100,000 times on the *coarsened graph*. We approximated the ground truth of the influence of each vertex v by simulating the diffusion process 100,000 times on the *input graph*. Since twitter-2010, com-Friendster, and uk-2007-05 are too large to compute the ground truth for every vertex, we were unable to obtain the results for these datasets. We denote the ground truth and our framework's estimation for v's influence as $\mathsf{Inf}_{\mathrm{gt}}(v)$ and $\mathsf{Inf}_{\mathrm{out}}(v)$, respectively. First, we quantitatively evaluated our estimation accuracy. Table 7.5 reports the mean absolute relative error (MARE) defined as $\frac{1}{|V|}\sum_{v\in V}\left|\frac{\mathsf{Inf}_{\mathrm{gt}}(v)-\mathsf{Inf}_{\mathrm{out}}(v)}{\mathsf{Inf}_{\mathrm{gt}}(v)}\right|$ and Spearman's rank correlation coefficient (RCC) between the ground truth and our estimation. The MAREs are at most 0.1025, i.e., our estimation errors are within 10% on average, and the RCCs are always greater than 0.8834. These results demonstrate both the accuracy and stability of our framework.

We then qualitatively analyzed the estimation accuracy. Figure 7.10 illustrates the relationship between the two estimations, where each point corresponds to a single vertex v, and the x and y coordinate $\mathsf{Inf}_{\mathrm{gt}}(v)$ and $\mathsf{Inf}_{\mathrm{out}}(v)$, respectively. The estimation with $r = 1$ was heavily biased to higher values because we accidentally merged a 1-robust SCC, which has a small strongly connected reliability. For $r = 16$, we avoided merging such "fragile" components. As a result, most of the estimations are close to the diagonal line, i.e., the ground truth.

We here justify our choice of parameter r in terms of accuracy. Figure 7.11 presents MAREs with various values of r. Higher values of r yield more accurate estimations, but the improvements are limited when $r \geq 16$. Because our algorithm with lower r produces smaller coarsened graphs, we adopted $r = 16$ as a sweet spot between accuracy and size reduction.

Table 7.5: Average estimation time of the influence of a single vertex for plain *MC* and our framework with *MC* under $\text{EXP}_{0.1}$ and TRI. MARE and RCC stand for "mean absolute relative error" and "rank correlation coefficient," respectively.

	exponential ($\text{EXP}_{0.1}$)					**trivalency** (TRI)				
	run time			**accuracy**		**run time**			**accuracy**	
dataset	*MC*	Alg.7.4(*MC*)	$\frac{\text{Alg.7.4}(MC)}{MC}$	MARE	RCC	*MC*	Alg.7.4(*MC*)	$\frac{\text{Alg.7.4}(MC)}{MC}$	MARE	RCC
ca-HepPh	9.0 s	1.8 s	20.1%	0.0163	0.9991	2.8 s	926.5 ms	32.9%	0.0284	0.9995
soc-Slashdot0922	31.6 s	8.0 s	25.4%	0.0156	0.9998	9.1 s	5.2 s	57.1%	0.0370	0.9996
web-NotreDame	229.8 ms	46.5 ms	20.2%	0.0090	0.9989	60.9 ms	16.6 ms	27.3%	0.0021	0.9942
com-Youtube	135.2 s	51.9 s	38.4%	0.0497	0.9999	28.0 s	14.1 s	50.4%	0.1025	0.9978
higgs-twitter	1,213.4 s	280.7 s	23.1%	0.0177	0.9942	443.0 s	248.8 s	56.2%	0.0246	0.9998
soc-Pokec	2,442.3 s	897.1 s	36.7%	0.0143	0.9969	606.0 s	481.7 s	79.5%	0.0422	1.0000
soc-LiveJournal1	5,348.8 s	1,782.6 s	33.3%	0.0240	0.9995	859.2 s	668.9 s	77.9%	0.0913	0.9999
com-Orkut	33,122.7 s	1,239.0 s	3.7%	0.0071	0.8834	26,080.0 s	6,696.2 s	25.7%	0.0114	0.9913
twitter-2010	106,428.0 s	24,212.4 s	22.8%	–	–	31,234.2 s	11,312.4 s	36.2%	–	–
com-Friendster	540,483.0 s	18,967.8 s	3.5%	–	–	279,137.0 s	32,467.3 s	11.6%	–	–
uk-2007-05	5,718.6 s	1,900.2 s	33.2%	–	–	20.2 s	1.6 s	8.0%	–	–

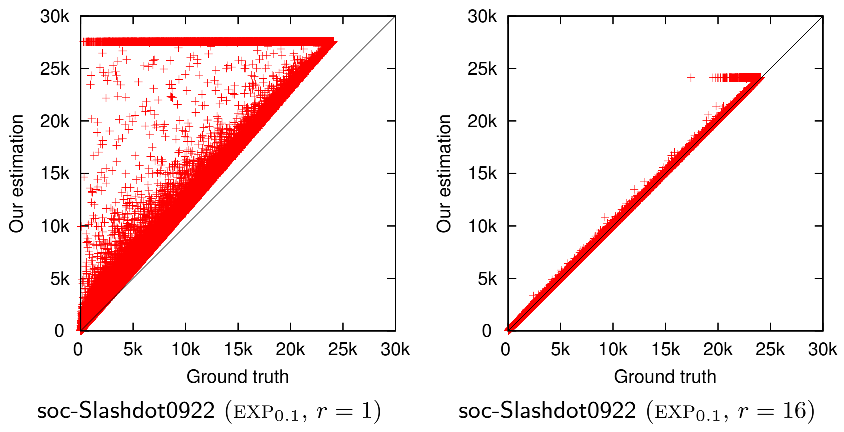


Figure 7.10: Influence correlation between the ground truth and our framework's estimation.

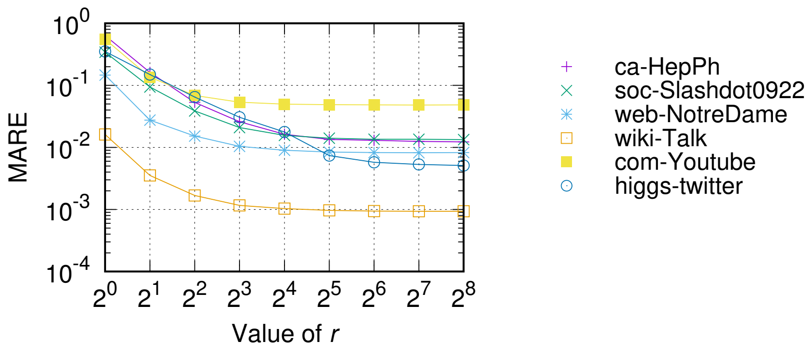


Figure 7.11: Estimation accuracy with varying r ($\text{EXP}_{0.1}$).

7.4.7 Evaluating Influence Maximization Framework

Fifth, we examined the effectiveness of our influence maximization framework (Algorithm 7.5). To this end, we applied our framework to *D-SSA* [148]. Recall *D-SSA* produces a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution with probability at least $1 - \delta$ for parameters $\epsilon \in [0, 1 - 2 \cdot \mathrm{e}^{-1}]$ and $\delta \in [0, 1]$. We set $\epsilon = 0.1$ and $\delta = 0.01$.

Efficiency improvement.

Table 7.6 summarizes the run time of plain *D-SSA* and our framework with *D-SSA* required to compute a solution of size 100. Our framework exhibited remarkable computation time reduction (up to 27.5%), which is approximately equal to the edge reduction ratio. This coincides with the *D-SSA* mechanism that iteratively performs RR set generation, whose computational cost is dominated by edge traversal costs.

Solution quality.

Table 7.6 reports the influence of a seed set extracted by plain *D-SSA* and our framework with *D-SSA*. To obtain reasonably accurate estimations, we conducted Monte-Carlo simulations 10,000 times. As expected from our framework's estimation accuracy, it provided solutions that were nearly the same quality as plain *D-SSA*.

Table 7.6: Run time for selecting a seed set of size 100 and solution quality for plain *D-SSA* and our framework with *D-SSA* under EXP$_{0.1}$ and TRI. OOM denotes "out of memory."

	exponential (EXP$_{0.1}$)					**trivalency** (TRI)				
	run time			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\lvert V\rvert$)		**run time**			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\lvert V\rvert$)	
dataset	*D-SSA*	Alg.7.5(*D-SSA*)	$\frac{\text{Alg.7.5}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.7.5(*D-SSA*)	*D-SSA*	Alg.7.5(*D-SSA*)	$\frac{\text{Alg.7.5}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.7.5(*D-SSA*)
ca-HepPh	26.1 s	14.3 s	54.8%	0.3574	0.3586	28.8 s	10.6 s	36.8%	0.1668	0.1671
soc-Slashdot0922	140.6 s	78.9 s	56.1%	0.2951	0.2955	271.4 s	158.2 s	58.3%	0.1359	0.1358
web-NotreDame	3.0 s	1.1 s	34.9%	0.0828	0.0830	1.8 s	1.7 s	93.9%	0.0329	0.0329
wiki-Talk	521.6 s	154.7 s	29.7%	0.1396	0.1397	221.8 s	84.4 s	38.1%	0.0521	0.0521
com-Youtube	2,808.7 s	772.8 s	27.5%	0.1511	0.1509	927.7 s	636.6 s	68.6%	0.0547	0.0548
higgs-twitter	3,873.8 s	1,228.1 s	31.7%	0.3510	0.3510	5,251.6 s	3,275.6 s	62.4%	0.1776	0.1776
soc-Pokec	18,349.5 s	6,215.7 s	33.9%	0.4739	0.4739	5,538.9 s	5,129.3 s	92.6%	0.1972	0.1971
soc-LiveJournal1	OOM	OOM	–%	–	–	7,806.3 s	11,771.8 s	150.8%	0.1307	0.1310
com-Orkut	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
twitter-2010	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
com-Friendster	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
uk-2007-05	OOM	OOM	–%	–	–	595.4 s	208.5 s	35.0%	0.0792	0.0790

7.4.8 Comparison with Existing Reduction Algorithms.

Finally, we compared the scalability of the proposed algorithm to the following existing algorithms for influence graph reduction: coarseNet [161] and Spine [138].

Settings.

For coarseNet, we used a C++ implementation [160] provided by the authors of [161] with GNU Octave [1] version 3.2 for eigenvalue calculation. We ran coarseNet with the same edge reduction rate as that shown in Table 7.2.

For Spine, we used a Java implementation [137] provided by the authors of [138]. Since Spine requires a log of cascades to produce a sparsified graph, we conducted Monte-Carlo simulations from $|V|$ randomly selected vertices for each graph. We ran Spine with the input graph and generated cascade logs so that the edge reduction ratio was the same as that of our algorithm's output.

Results.

Table 7.7 compares the run time of our linear-space implementation, coarseNet, and Spine for the exponential setting. Note that we observed the same tendency for the trivalency setting. The proposed method is several orders of magnitude faster than both existing algorithms for larger graphs. Moreover, due to out-of-memory error, coarseNet could handle only medium-sized graphs with tens of millions of edges and Spine could process only the smallest four graph. These results demonstrate the superiority of the proposed algorithm over existing algorithms in terms of both computation time and memory consumption.

Table 7.7: Comparison of the run time of each algorithm under $\text{EXP}_{0.1}$. OOM denotes "out of memory."

	exponential ($\text{EXP}_{0.1}$)		
dataset	This work (Alg. 7.1)	coarseNet [161]	Spine [138]
ca-GrQc	0.02 s	0.01 s	10.83 s
ca-HepTh	0.04 s	0.01 s	29.81 s
wiki-Vote	0.04 s	1.54 s	183.51 s
ca-HepPh	0.09 s	2.11 s	3,800.05 s
soc-Epinions1	0.38 s	57.54 s	OOM
soc-Slashdot0922	0.54 s	116.75 s	OOM
web-NotreDame	1.77 s	55.26 s	OOM
ego-Twitter	0.80 s	238.93 s	OOM
loc-Gowalla	1.29 s	69.26 s	OOM
web-Stanford	1.86 s	158.89 s	OOM
wiki-Talk	42.31 s	4,760.76 s	OOM
web-Google	7.93 s	1.30 s	OOM
com-Youtube	14.59 s	570.05 s	OOM
web-BerkStan	6.44 s	552.80 s	OOM
higgs-twitter	7.64 s	9,574.53 s	OOM
soc-Pokec	35.20 s	28,158.00 s	OOM
soc-LiveJournal1	94.58 s	OOM	OOM
com-Orkut	122.34 s	OOM	OOM
twitter-2010	1,762.93 s	OOM	OOM
com-Friendster	3,964.05 s	OOM	OOM
uk-2007-05	3,105.88 s	OOM	OOM

Chapter 8

Portfolio Optimization for Acquiring Low-risk Strategies

In this chapter, we will address the risk of having a small number of influenced vertices that expectation may not capture. Now that, we formally define our portfolio optimization problem for CVaR maximization. Let $\mathsf{r}_{\mathcal{G}}(S)$ denote the random variable representing $\mathsf{r}_G(S)$ for $G \sim \mathcal{G}$. For a finite set V and a positive integer k, let $\Delta_{V,k}$ be the $\binom{|V|}{k}$-dimensional simplex indexed by a set in $\binom{V}{k}$, i.e.,

$$\Delta_{V,k} = \{\boldsymbol{\pi} \in \mathbb{R}^{\binom{V}{k}} \mid \|\boldsymbol{\pi}\|_1 = 1\}. \tag{8.1}$$

We call each vector in $\Delta_{V,k}$ a $\binom{|V|}{k}$-dimensional *portfolio*. Our problem is defined as follows.

Problem 4 (Portfolio optimization for cascade CVaR maximization)**.** *Given an influence graph $\mathcal{G} = (V, E, p)$, an integer k, and a significance parameter $\alpha \in (0,1)$, find a $\binom{|V|}{k}$-dimensional portfolio $\boldsymbol{\pi} \in \Delta_{V,k}$ that maximizes the CVaR of the cascade size at significance level α, i.e.,* $\mathsf{CVaR}_\alpha[\sum_{S \in \binom{V}{k}} \pi_S \mathsf{r}_{\mathcal{G}}(S)]$.

In this chapter, we propose a polynomial-time algorithm with a guarantee of an additive error for Problem 4 (Section 8.1), and then we evaluate our algorithm and compare it with the original influence maximization (Section 8.2).

8.1 Proposed Algorithm

8.1.1 Overview

An overview of the proposed algorithm for Problem 4 is illustrated in Figure 8.1. First, we consider to optimize the empirical CVaR defined over the samples and write it as a linear program (Section 8.1.2). Next, we will explain a way to check the feasibility of the linear program using the MWU algorithm (Section 8.1.3). Here, the MWU algorithm assumes an oracle for a relaxation of the linear program. Then, we describe a greedy-based approximation oracle (Section 8.1.4). Finally, we put all together and analyze the total time complexity and accuracy guarantee (Section 8.1.5).

For ease of notation, for a vertex set $S \subseteq V$, we use X_S to denote a random variable $\mathsf{r}_{\mathcal{G}}(S)/|V|$ bounded in $[0,1]$. Let $\mathbf{X} = (X_S)_{S \in \binom{V}{k}}$ and $\mathcal{D}_{\mathbf{X}}$ denote the distribution of $\mathbf{X}$. Note that our problem is equivalent to find a portfolio $\boldsymbol{\pi} \in \Delta_{V,k}$ that maximizes $\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{X} \rangle]$.

The main result of this section is as follows.

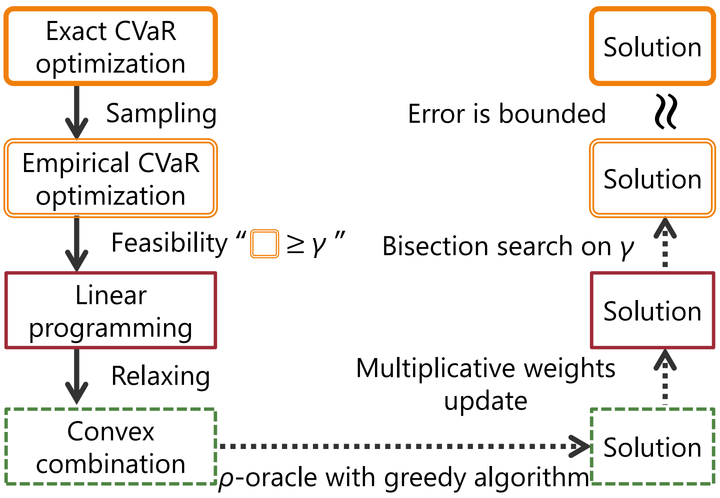


Figure 8.1: Overview of the proposed algorithm in Chapter 8.

Theorem 8.1. *Given an influence graph $\mathcal{G} = (V, E, p)$, an integer k, and $\alpha, \epsilon, \delta \in (0, 1)$, there exists an algorithm that returns a portfolio $\boldsymbol{\pi} \in \Delta_{V,k}$ such that*

$$\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{X} \rangle] \geq \left(\max_{\boldsymbol{\pi} \in \Delta_{V,k}} \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{X} \rangle] \right) - \mathrm{e}^{-1} - \epsilon \tag{8.2}$$

with probability at least $1 - \delta$. The running time is $\mathcal{O}(\frac{\log \epsilon^{-1} \cdot \log r}{\epsilon^2} k |V||E|r)$, where r satisfies $r = \mathcal{O}\left(\frac{1}{\epsilon^2} \left(\frac{k \log r}{\epsilon^2} \log \frac{|V|}{k} + \log \frac{1}{\delta} \right) \right)$.

Remark that for the problem of selecting a "single vertex set" that maximizes the CVaR of the cascade size, there is no polynomial time multiplicative approximation algorithm as proved by Maehara [134].

8.1.2 Empirical CVaR Maximization

Since it is hard to optimize CVaR under $\mathcal{D}_\mathbf{X}$ directly, we consider the empirical distribution of $\mathcal{D}_\mathbf{X}$. For a positive integer, we define $\hat{\mathcal{D}}_{\mathbf{X},r}$ as the uniform distribution over $\{\mathbf{X}^1, \ldots, \mathbf{X}^r\}$, where $\mathbf{X}^1, \ldots, \mathbf{X}^r$ are independent samples from $\mathcal{D}_\mathbf{X}$. Then, we optimize $\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{Y} \rangle]$, where $\mathbf{Y}$ is a vector of random variables sampled from $\hat{\mathcal{D}}_{\mathbf{X},r}$. The value of r is determined in Section 8.1.5. Note that $\mathbf{X}$ is of size $\binom{|V|}{k}$, and therefore we cannot *explicitly* sample $\mathbf{X}^1, \ldots, \mathbf{X}^r$ from $\mathcal{D}_\mathbf{X}$. We use a trick based on the random-graph interpretation and *implicitly* construct $\hat{\mathcal{D}}_{\mathbf{X},r}$ by sampling random graphs $G_1, \ldots, G_r$ from $\mathcal{G}$, and set $X^i_S = \mathsf{r}_{G_i}(S)/|V|$ for any vertex set $S \subseteq V$. In summary, we optimize CVaR under $\hat{\mathcal{D}}_{\mathbf{X},r}$. Recalling Eq. (2.20), we restate the problem as

$$\begin{array}{ll} \text{maximize} & \tau - \frac{1}{\alpha r} \sum_{i \in [r]} \max\left\{ \tau - \langle \boldsymbol{\pi}, \mathbf{X}^i \rangle, 0 \right\} \\ \text{subject to} & \tau \in [0, 1] \\ & \boldsymbol{\pi} \in \Delta_{V,k}. \end{array} \tag{8.3}$$

We can add the extra constraint $\tau \in [0, 1]$ without loss of generality because $X^i_S \in [0, 1]$ for every $i \in [r]$ and $S \in \binom{V}{k}$. By introducing auxiliary variables $\mathbf{y} =$

$(y_i)_{i\in[r]}$, this problem can be further rephrased as the following *linear program*:

$$\begin{array}{ll} \text{maximize} & \tau - \frac{1}{\alpha r}\sum_{i\in[r]} y_i \\ \text{subject to} & y_i \geq \tau - \langle \boldsymbol{\pi}, \mathbf{X}^i \rangle \ \forall i \in [r] \\ & \tau \in [0,1] \\ & \mathbf{y} \in [0,1]^r \\ & \boldsymbol{\pi} \in \Delta_{V,k}. \end{array} \tag{8.4}$$

We aim at solving Eq. (8.4) via bisection search on the objective value. However, we cannot solve the feasibility version of Eq. (8.4) because of exponentially many variables $\boldsymbol{\pi} \in \Delta_{V,k}$. To resolve this difficulty, we further relax the feasibility version by using the MWU algorithm.

8.1.3 Finding Approximate Feasible Solutions via MWU

Here, we discuss how to use the MWU algorithm to approximately check the feasibility of Eq. (8.4). Before that, we introduce several definitions to simplify the feasibility problem. Let $\gamma \in [0,1]$ be a midpoint for the bisection search. We define a set

$$\mathcal{P}_\gamma = \left\{(\tau, \mathbf{y}, \boldsymbol{\pi}) \mid \tau \in [0,1], \mathbf{y} \in [0,1]^r, \boldsymbol{\pi} \in \Delta_{V,k}, \tau - \frac{1}{\alpha r}\sum_{i\in[r]} y_i \geq \gamma\right\}. \tag{8.5}$$

If $\mathcal{P}_\gamma$ is not empty, then the maximum of Eq. (8.4) is at least γ. Given a triplet $(\tau, \mathbf{y}, \boldsymbol{\pi})$, we define a vector $\mathbf{v} := \mathbf{v}(\tau, \mathbf{y}, \boldsymbol{\pi})$, where $v_i = y_i + \langle \boldsymbol{\pi}, \mathbf{X}^i \rangle - \tau$ for each $i \in [r]$. Then, the feasibility version of Eq. (8.4) for a midpoint γ can be written as follows.

$$\exists ?\ (\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}_\gamma : \mathbf{v} \geq \mathbf{0}. \tag{8.6}$$

Here goes the main part. We will approximately solve Eq. (8.6) via the MWU algorithm. We adapt the feasibility checking algorithm presented in [15, Section 3.3]. The MWU algorithm repeatedly generates a distribution $\mathbf{d} \in \Delta_r$ and requires to check the feasibility of the following problem.

$$\exists ?\ (\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}_\gamma : \langle \mathbf{d}, \mathbf{v} \rangle \geq 0. \tag{8.7}$$

Note that the constraint "$\langle \mathbf{d}, \mathbf{v} \rangle \geq 0$" is a convex combination of "$\mathbf{v} \geq \mathbf{0}$". Thus, we may expect the existence of a solver that checks Eq. (8.7). To be precise, we define a notion of ρ-*oracle* for such a solver.

Definition 8.2 (ρ-oracle)**.** *For a parameter $\rho \in (0,1)$, a* ρ-oracle *for Eq.* (8.7) *is an algorithm which given a distribution $\mathbf{d} \in \Delta_r$, either finds a triplet $(\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}_\gamma$ such that $\langle \mathbf{d}, \mathbf{v} \rangle \geq -\rho$, or correctly declares that Eq.* (8.7) *is infeasible. Moreover, whenever the oracle returns a triplet $(\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}_\gamma$, $|v_i| \leq 1$ holds for every $i \in [r]$.*

We defer the implementation of a ρ-oracle to the next subsection.

In the following, we describe how to use a ρ-oracle in the MWU algorithm for approximate feasibility checking. Given a ρ-oracle, we consider Algorithm 8.1. In each round t, given a distribution $\mathbf{d}^{(t)}$ over the constraints, the algorithm runs the ρ-oracle with $\mathbf{d}^{(t)}$. It immediately declares *reject* if the oracle failed to find any feasible solution. Otherwise, it uses the feedback $(\tau^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\pi}^{(t)})$ of the oracle to adjust the distribution. After all rounds, it takes the average over the triplets $\{(\tau^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\pi}^{(t)})\}_{t\in[T]}$. We obtain the following guarantee:

Algorithm 8.1 Approximate checking of Eq. (8.6) via the MWU algorithm.

```
Input: random graphs $G_1, \ldots, G_r$, $\alpha$, $\epsilon_{\mathrm{mw}}$, $\gamma$.
 1: $T \leftarrow \frac{16 \log r}{\epsilon_{\mathrm{mw}}^2}$.
 2: $\eta \leftarrow \min\{\frac{\epsilon_{\mathrm{mw}}}{4}, \frac{1}{2}\}$.
 3: $\mathbf{w}^{(1)} \leftarrow \mathbf{1}$.
 4: for $t = 1$ to $T$ do
 5:     $\mathbf{d}^{(t)} \leftarrow \mathbf{w}^{(t)} / \|\mathbf{w}^{(t)}\|_1$.
 6:     call the $\rho$-oracle for Eq. (8.7) (Algorithm 8.2) with $\mathbf{d}^{(t)}$.
 7:     if the oracle declares no feasible solution then
 8:         reject.
 9:     $(\tau^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\pi}^{(t)}) \leftarrow$ the solution returned by the oracle.
10:     for all $i \in [r]$ do
11:         $\mathbf{v}^{(t)} \leftarrow \mathbf{v}(\tau^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\pi}^{(t)})$.
12:         $w_i^{(t+1)} \leftarrow w_i^{(t)}(1 - \eta v_i^{(t)})$.
13: $\bar{\tau} \leftarrow \frac{1}{T}\sum_{t \in [T]} \tau^{(t)}$, $\bar{\mathbf{y}} \leftarrow \frac{1}{T}\sum_{t \in [T]} \mathbf{y}^{(t)}$, $\bar{\boldsymbol{\pi}} \leftarrow \frac{1}{T}\sum_{t \in [T]} \boldsymbol{\pi}^{(t)}$.
14: return $(\bar{\tau}, \bar{\mathbf{y}}, \bar{\boldsymbol{\pi}})$.
```

Lemma 8.3. *Algorithm 8.1 either finds a triplet $(\bar{\tau}, \bar{\mathbf{y}}, \bar{\boldsymbol{\pi}}) \in \mathcal{P}_\gamma$ such that $\bar{\mathbf{v}} := \mathbf{v}(\bar{\tau}, \bar{\mathbf{y}}, \bar{\boldsymbol{\pi}})$ satisfies $\bar{v}_i \geq -\rho - \epsilon_{\mathrm{mw}}$ for every $i \in [r]$, or correctly declares that Eq. (8.6) is infeasible. The algorithm makes at most $T = \frac{16 \log r}{\epsilon_{\mathrm{mw}}^2}$ calls to the ρ-oracle.*

Proof. If the ρ-oracle declares that there is no $(\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}$ such that $\langle \mathbf{d}^{(t)}, \mathbf{v}^{(t)} \rangle \geq 0$, then we terminate because $\mathbf{d}^{(t)}$ is the proof that Eq. (8.6) is infeasible.

Now, let us assume that this does not occur, that is, in all rounds t, the ρ-oracle manages to find a solution $(\tau^{(t)}, \mathbf{y}^{(t)}, \boldsymbol{\pi}^{(t)}) \in \mathcal{P}_\gamma$ such that $\langle \mathbf{d}^{(t)}, \mathbf{v}^{(t)} \rangle \geq -\rho$. Note that $(\bar{\tau}, \bar{\mathbf{y}}, \bar{\boldsymbol{\pi}}) \in \mathcal{P}_\gamma$ since $\mathcal{P}_\gamma$ is a convex set.

Since the cost vector to the MWU algorithm is specified as $\mathbf{v}^{(t)}$, we conclude that the expected cost in each round is $\langle \mathbf{d}^{(t)}, \mathbf{v}^{(t)} \rangle \geq -\rho$. Hence, Theorem 2.45 tells us that after T rounds, for any $i \in [r]$, we obtain $\bar{v}_i = \frac{1}{T}\sum_{t \in [T]} v_i^{(t)} \geq -\rho - \epsilon_{\mathrm{mw}}$. Let $\bar{\tau} = \frac{1}{T}\sum_{t \in [T]} \tau^{(t)}$, $\bar{\mathbf{y}} = \frac{1}{T}\sum_{t \in [T]} \mathbf{y}^{(t)}$, and $\bar{\boldsymbol{\pi}} = \frac{1}{T}\sum_{t \in [T]} \boldsymbol{\pi}^{(t)}$. Setting $\bar{\mathbf{v}} = \frac{1}{T}\sum_{t \in [T]} \mathbf{v}^{(t)}$, we get $\bar{v}_i \geq -\rho - \epsilon_{\mathrm{mw}}$, which means that $\bar{\mathbf{v}}$ satisfies the feasibility problem Eq. (8.6) to within $\rho + \epsilon_{\mathrm{mw}}$.

The number of calls to the oracle that the algorithm makes is obvious. $\square$

8.1.4 Implementation of a ρ-oracle

In this section, we implement an e^{-1}-oracle for the feasibility problem Eq. (8.7). Note that it still contains exponentially many variables $\boldsymbol{\pi}$; however, it contains only a "single" constraint. Our crucial observation for the design of the oracle is the following:

> We do not need to construct a portfolio but just choose a single set. Furthermore, it suffices to solve an instance of influence maximization.

For a vertex set $S \in \binom{V}{k}$, let $\mathbf{1}^S \in \Delta_{V,k}$ be the portfolio where all entries are zero except the entry at S is one. We define $f_{\mathbf{d}} : 2^V \to \mathbb{R}$ as $f_{\mathbf{d}}(S) = \sum_{i \in [r]} d_i X_S^i$. Then, given a midpoint γ and a distribution $\mathbf{d}$, the objective of the oracle is maximizing $\langle \mathbf{d}, \mathbf{v} \rangle$ subject to $(\tau, \mathbf{y}, \boldsymbol{\pi}) \in \mathcal{P}_\gamma$. Note that expanding $\langle \mathbf{d}, \mathbf{v} \rangle$ yields

$$\begin{aligned}\langle \mathbf{d}, \mathbf{v} \rangle &= \sum_{i \in [r]} d_i (y_i - \tau) + \sum_{S \in \binom{V}{k}} \pi_S f_{\mathbf{d}}(S) \\ &= \sum_{i \in [r]} d_i y_i + \sum_{S \in \binom{V}{k}} \pi_S f_{\mathbf{d}}(S) - \tau. \end{aligned} \tag{8.8}$$

Let S_k^* be a maximizer of the problem $\max_{S \in \binom{V}{k}} f_{\mathbf{d}}(S)$. Then, we have the following lemma.

Lemma 8.4. *If $(\tau, \mathbf{y}, \boldsymbol{\pi})$ attains the maximum of $\langle \mathbf{d}, \mathbf{v} \rangle$, then we can assume that $\boldsymbol{\pi} = \mathbf{1}^{S_k^*}$.*

Proof. This is clear from Eq. (8.8); otherwise, we can increase the objective value by decreasing π_S for $S \neq S_k^*$ with $\pi_S > 0$ and by increasing $\pi_{S_k^*}$. $\square$

Although it is NP-hard to exactly compute S_k^*, we can compute $S_k \in \binom{V}{k}$ such that $f_{\mathbf{d}}(S_k) \geq (1 - \mathrm{e}^{-1}) f_{\mathbf{d}}(S_k^*)$ due to the monotonicity and submodularity of $f_{\mathbf{d}}$.

Lemma 8.5. *$f_{\mathbf{d}}(\cdot)$ is monotone and submodular.*

Proof. For each $i \in [r]$, $\mathsf{r}_{G_i}(S)$ is monotone and submodular in S and so is $X_S^i = \mathsf{r}_{G_i}(S)/|V|$. Note that a class of monotone submodular functions is closed under non-negative linear combinations. $f_{\mathbf{d}}(\cdot)$ is therefore monotone and submodular. $\square$

Lemma 8.6. *Assume $d_1 \geq d_2 \geq \cdots \geq d_r$. If $(\tau, \mathbf{y}, \boldsymbol{\pi})$ is a feasible solution to Eq. (8.7), then we can assume the following:*

- *For every $i \in [r]$, if $d_i \geq \frac{1}{\alpha r}$, then we have*

$$y_i = \min\Big\{\alpha r - \sum_{j \in [i-1]} y_j, 1\Big\}, \tag{8.9}$$

  and otherwise we have $y_i = 0$.

- *$\tau = 1 - \frac{1}{\alpha r} \sum_{i \in [r]} y_i$.*

Proof. Fixing $\boldsymbol{\pi}$, we want to maximize $\sum_{i \in [r]} d_i y_i - \tau$ subject to $\tau - \frac{1}{\alpha r} \sum_{i \in [r]} y_i \geq \gamma$.

If there exists $y_i < 1$ and $y_j > 0$ with $i < j$, then we can increase the objective value by increasing y_i and decreasing y_j.

If there exists $y_i > 0$ with $d_i < \frac{1}{\alpha r}$, we can increase the objective value decreasing y_i and increasing τ.

From these observations, we have the claim. $\square$

These lemmas motivate us to consider Algorithm 8.2.

Theorem 8.7. *Algorithm 8.2 is an e^{-1}-oracle for Eq. (8.7) and runs in time $O(k|V||E|r)$. Moreover, it returns a portfolio $\boldsymbol{\pi}$ consisting of a single vertex set.*

Proof. We have the following two cases.

- **Suppose that Algorithm 8.2 returned a triplet $(\tau, \mathbf{y}, \boldsymbol{\pi})$.** Then, we clearly have $\langle \mathbf{d}, \mathbf{v} \rangle \geq -\mathrm{e}^{-1}$. In addition, we have $|v_i| \leq 1$.

Algorithm 8.2 An e^{-1}-oracle for Eq. (8.7).

```
Input: random graphs $G_1, \dots, G_r$, $\alpha$, and $\mathbf{d}$.
 1: order $\mathbf{d}$ so that $d_1 \geq d_2 \geq \cdots \geq d_r$.
 2: for $i = 1$ to $r$ do
 3:     if $d_i \geq \frac{1}{\alpha r}$ then
 4:         $y_i \leftarrow \min\{\alpha s - \sum_{j \in [i-1]} y_j, 1\}$.
 5:     else
 6:         $y_i \leftarrow 0$.
 7: $\tau \leftarrow 1 - \frac{1}{\alpha r} \sum_{i \in [r]} y_i$.
 8: compute $S_k \in \binom{V}{k}$ by applying the greedy algorithm to $f_{\mathbf{d}}$.
 9: $\boldsymbol{\pi} \leftarrow \mathbf{1}^{S_k}$.
10: if $\langle \mathbf{d}, \mathbf{v} \rangle \geq -1/e$ then
11:     return $(\tau, \mathbf{y}, \boldsymbol{\pi})$.
12: else
13:     reject.
```

Algorithm 8.3 Generating random graphs from a given influence graph.

```
Input: $\mathcal{G} = (V, E, p)$, $\alpha$, $\epsilon$, $\delta$.
 1: $r \leftarrow$ the number of samples determined in Theorem 8.1.
 2: generate $r$ random graphs $G_1, \dots, G_r$ from $\mathcal{G}$,
 3: call Algorithm 8.4 with $G_1, \dots, G_r$, $\alpha$, and $\epsilon' = \epsilon/3$.
 4: return the obtained portfolio $\boldsymbol{\pi}$.
```

- **Suppose that Algorithm 8.2 rejected.**
  Then, the triplet $(\tau^*, \mathbf{y}^*, \boldsymbol{\pi}^*) := (\tau, \mathbf{y}, \mathbf{1}^{S_k^*})$ maximizes $\langle \mathbf{d}, \mathbf{v}^* \rangle$, which is bounded as

$$\begin{aligned}\langle \mathbf{d}, \mathbf{v}^* \rangle &= \sum_{i \in [r]} d_i y_i + f_{\mathbf{d}}(S_k^*) - \tau^* \leq \sum_{i \in [r]} d_i y_i + \frac{e}{e-1} f_{\mathbf{d}}(S_k) - \tau \\ &\leq \frac{e}{e-1}\Big(\sum_{i \in [r]} d_i y_i + f_{\mathbf{d}}(S_k) - \tau\Big) + \frac{1}{e-1} \qquad \text{(Since } \tau \leq 1\text{)} \\ &< \frac{e}{e-1}\Big(-e^{-1}\Big) + \frac{1}{e-1} = 0. \qquad (8.10)\end{aligned}$$

  This is the proof that Eq. (8.7) is infeasible.

Hence, Algorithm 8.2 is an e^{-1}-oracle for Eq. (8.7).

The total running time is dominated by that of the greedy algorithm. The greedy algorithm evaluates $f_{\mathbf{d}}(\cdot)$ for at most $k|V|$ vertex sets, and each function evaluation completes in time $\mathcal{O}(r|E|)$. Therefore, it requires $O(k|V||E|r)$ time. □

8.1.5 Putting All Together

The overall algorithm is shown in Algorithms 8.3 and 8.4. In Algorithms 8.3, we first generate r random graphs $G_1, \dots, G_r$ from $\mathcal{G}$ and then invoke Algorithm 8.4 with these random graphs. Algorithm 8.4 performs bisection search $\lg \frac{2}{\epsilon'}$ times on the interval $[\gamma_l, \gamma_h]$. For the midpoint $\gamma = (\gamma_l + \gamma_h)/2$, we call Algorithm 8.1 to check the feasibility of Eq. (8.6). At the end of the algorithm, we return the portfolio having the maximum CVaR made so far.

Algorithm 8.4 Bisection search with approximate feasibility checking.

```
Input: random graphs $G_1, \ldots, G_r$, $\alpha$, $\epsilon'$
 1: $\gamma_l \leftarrow 0$ and $\gamma_h \leftarrow 1$.
 2: for $\lg \frac{2}{\epsilon'}$ times do
 3:     $\gamma \leftarrow (\gamma_l + \gamma_h)/2$.
 4:     call Algorithm 8.1 with $G_1, \ldots, G_r$, $\alpha$, $\epsilon_{\mathrm{mw}} = \epsilon'/2$, and $\gamma = \gamma$.
 5:     if Algorithm 8.1 declares no feasible solution then
 6:         $\gamma_h \leftarrow \gamma$.
 7:     else
 8:         $\gamma_l \leftarrow \gamma$.
 9: call Algorithm 8.1 with $G_1, \ldots, G_r$, $\alpha$, $\epsilon_{\mathrm{mw}} = \epsilon'/2$, and $\gamma = \gamma_l$.
10: return the obtained portfolio $\boldsymbol{\pi}$.
```

We now prove Theorem 8.1. We first measure the gap of the empirical CVaR between the optimal portfolio and the obtained portfolio.

Lemma 8.8. *Algorithm 8.4 returns a portfolio $\boldsymbol{\pi} \in \Delta_{V,k}$ such that*

$$\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{Y} \rangle] \geq \hat{\gamma}^* - \mathrm{e}^{-1} - \epsilon', \tag{8.11}$$

where $\hat{\gamma}^*$ is the optimal value of Eq. (8.4). It runs in $\mathcal{O}(\frac{k|V||E|r \log r}{\epsilon'^2} \log \frac{1}{\epsilon'})$ time. Moreover, each weight of $\boldsymbol{\pi}$ is a multiple of $1/T$.

Proof. We note that $\gamma_h \geq \hat{\gamma}^*$ (because Algorithm 8.1 always returns a solution when there is a feasible solution to Eq. (8.6)), and $\gamma_h - \gamma_l \leq \epsilon'/2$. It follows that $\gamma_l \geq \hat{\gamma}^* - \epsilon'/2$.

Let $(\tau, \mathbf{y}, \boldsymbol{\pi})$ be the solution obtained at line 9 and let $\mathbf{v} := \mathbf{v}(\tau, \mathbf{y}, \boldsymbol{\pi})$. Then, we obtain $v_i \geq -\mathrm{e}^{-1} - \epsilon'/2$ for any $i \in [r]$ from Lemma 8.3 and Theorem 8.7. This means that $(\tau - \mathrm{e}^{-1} - \epsilon'/2, \mathbf{y}, \boldsymbol{\pi})$ is a feasible solution to Eq. (8.4) with an objective value of at most $\gamma_l - \mathrm{e}^{-1} - \epsilon'/2 \geq \hat{\gamma}^* - \mathrm{e}^{-1} - \epsilon'$.

The time complexity and requirement for $\boldsymbol{\pi}$ are obvious from Lemma 8.3 and Theorem 8.7. □

We then bound the gap between the actual CVaR and empirical CVaR for every portfolio that the proposed algorithm may produce and for the optimal portfolio. Let $L \subseteq \Delta_{V,k}$ be the set of all portfolios such that each coordinate is a multiple of $1/T$.

Lemma 8.9. *It holds that*

$$\begin{aligned} \Big|\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}', \mathbf{X} \rangle] - \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}', \mathbf{Y} \rangle]\Big| &\leq \frac{\epsilon}{3} \quad \textit{and} \\ \Big|\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}^*, \mathbf{X} \rangle] - \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}^*, \mathbf{Y} \rangle]\Big| &\leq \frac{\epsilon}{3} \end{aligned} \tag{8.12}$$

for every portfolio $\boldsymbol{\pi}'$ that Algorithm 8.4 may return and for the optimal portfolio $\boldsymbol{\pi}^* \in \Delta_{V,k}$ with probability at least $1 - \delta$.

Proof. Remark that Algorithm 8.4 returns a portfolio only in L as its weights are a multiple of $1/T$. Then, we use Lemma 2.44 with $\epsilon/3$ and $\delta/(|L|+1)$. Since $|L| \leq \binom{\binom{|V|}{k}+T-1}{T}$ holds, the required number of random graphs is

$$\begin{aligned} r &= \Omega\Big(\frac{1}{\epsilon^2} \log \frac{|L|+1}{\delta}\Big) = \Omega\Big(\frac{1}{\epsilon^2}\big(T \log \frac{\binom{|V|}{k} + T - 1}{T} + \log \frac{1}{\delta}\big)\Big) \\ &= \Omega\Big(\frac{1}{\epsilon^2}\big(\frac{k \log r}{\epsilon^2} \log \frac{|V|}{k} + \log \frac{1}{\delta}\big)\Big). \end{aligned} \tag{8.13}$$

By taking union bound over $L \cup \{\boldsymbol{\pi}^*\}$, we obtain the desired claim. □

Finally, by using Lemmas 8.8 and 8.9, we prove Theorem 8.1.

Proof. **Approximation guarantee.** Let $\gamma^* = \max_{\boldsymbol{\pi} \in \Delta_{V,k}} \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{X} \rangle]$ be the optimal CVaR, and let $\hat{\gamma}^*$ be the optimal value of Eq. (8.4). We obtain $\mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{Y} \rangle] \geq \hat{\gamma}^* - \mathrm{e}^{-1} - \epsilon' = \hat{\gamma}^* - \mathrm{e}^{-1} - \epsilon/3$ by Lemma 8.8.

Since $\boldsymbol{\pi} \in L$, we obtain

$$\begin{aligned} \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{X} \rangle] &\underbrace{\geq}_{\text{Lem. 8.9}} \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}, \mathbf{Y} \rangle] - \epsilon/3 \\ &\underbrace{\geq}_{\text{Lem. 8.8}} \hat{\gamma}^* - \mathrm{e}^{-1} - 2\epsilon/3 \geq \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}^*, \mathbf{Y} \rangle] - \mathrm{e}^{-1} - 2\epsilon/3 \\ &\underbrace{\geq}_{\text{Lem. 8.9}} \mathsf{CVaR}_\alpha[\langle \boldsymbol{\pi}^*, \mathbf{X} \rangle] - \mathrm{e}^{-1} - \epsilon = \gamma^* - \mathrm{e}^{-1} - \epsilon \end{aligned} \tag{8.14}$$

with probability at least $1 - \delta$.

Time complexity. The analysis of the time complexity is obvious. □

8.2 Experimental Evaluations

In this section, we demonstrate the effectiveness of the proposed algorithm by experiment. We conducted experiments on a Linux server with an Intel Xeon E5-2670 2.60 GHz CPU and 512 GB memory. All algorithms were implemented in C++ and compiled using g++ 4.8.2 with the -O2 option.

8.2.1 Setup

We used three publicly available real-world network datasets, Karate with 34 vertices and 78 bidirectional edges, Physicians with 117 vertices and 542 directed edges, and Advogato with 5,042 vertices and 78,454 directed edges from the Koblenz Network Collection [112, 112]. We extracted the subgraphs induced by the largest connected components. For edge influence probabilities, we adopt the OWC and $\mathrm{UC}_{0.1}$ settings. The significance level α was set to be 0.01 and 0.05.

For our method, we set $\epsilon = 0.4$ and assigned $s = \frac{k \log |V|}{\epsilon^4}, T = \frac{\log r}{\epsilon^2}, \eta = \sqrt{\frac{\log s}{T}}$, and the bisection search was repeated 32 times. We compared our method with the following three baseline algorithms that output only a single seed set:

- *Greedy*, which is a standard influence maximization algorithm. We use *PMC* proposed in Chapter 5.
- *Degree*, which selects k vertices in the decreasing order of degrees.
- *Random*, which selects k vertices uniformly at random.

8.2.2 Results

First, we verify the effectiveness of our proposed method. Tables 8.1 and 8.3 report the CVaR at $\alpha = 0.01$ and $\alpha = 0.05$ of the portfolio obtained by each method. We conducted Monte Carlo simulations of influence spread 10,000 times to obtain an estimation of the CVaR for each portfolio. Under the OWC setting, the proposed method significantly outperformed existing methods for all settings. The difference is especially large when k is small, but even when k is large, e.g., the CVaRs at $\alpha = 0.01$ of our portfolios on Physicians and Advogato for $k = 15$ are 17.5% and 67.6% better than the second-best, respectively. If we use the UC$_{0.1}$ setting, we can observe smaller than OWC or no improvements. For example, portfolio construction is of almost no benefit for Advogato network. This is because of the giant component (GC); once we select a vertex frequently appearing the GC, we get a large cascade with a certain probability. Still, we are able to obtain high-CVaR portfolios for networks where cascades immediately stop spreading, e.g., Physicians network.

Tables 8.2 and 8.4 show the expected cascade sizes. Although our method does not explicitly optimize the expected cascade size, we can observe that those of the portfolios obtained by our method are comparable to those of seed sets obtained by the greedy method.

Figure 8.2 shows the histogram of the cascade size for each method. The histograms of baseline algorithms spread out, which easily result in extremely smaller cascades than the average. On the other hand, the histograms of the portfolios computed by our method are promising; they are well concentrated on the mean value, which is more desirable in terms of risk aversion.

Table 8.5 shows the number of positive weights of each portfolio obtained by our method. The number of positive weights is at most 100, which are reasonably small.

Finally, we show the structure of the portfolio obtained by our method. Figure 8.3 illustrates the weights in the portfolio obtained by applying our method to Karate network with $k = 1$ and $\alpha = 0.01$. In this network, there are two overlapping communities centered at vertices 1 and 34. Our method assigns positive weights to vertices in both communities, for example, vertices 12 and 27, which are connected to vertex 1 and 34, respectively. Note that *Greedy*, *Degree*, and *Random* selected vertices 12, 34, and 19 as a seed vertices, respectively.

From the abovementioned results, we have shown the effectiveness of a portfolio optimization approach for risk aversion.

Table 8.1: CVaR for portfolios obtained by each method under OWC. Best results are in bold.

$\alpha = 0.01$	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work	1.7	**8.7**	**15.6**	**21.3**	**1.5**	**12.0**	**23.6**	**34.2**	**1.2**	**18.7**	**42.3**	**69.9**
Greedy	**2.0**	6.8	14.5	20.3	1.0	5.7	16.9	29.1	1.0	7.5	21.5	41.7
Degree	1.0	5.0	10.2	15.4	1.0	5.5	14.2	24.1	1.0	5.3	14.7	26.8
Random	1.0	5.5	12.8	17.6	1.0	5.5	15.1	22.8	1.0	6.5	17.8	34.4
$\alpha = 0.05$	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work	1.9	**9.6**	**16.8**	**22.4**	**1.9**	**13.7**	**26.3**	**37.2**	**1.4**	**22.3**	**48.9**	**78.7**
Greedy	**2.0**	8.1	15.8	21.6	1.0	7.6	20.9	33.4	1.0	9.0	27.1	50.7
Degree	1.0	5.8	10.8	16.1	1.0	7.0	16.9	27.3	1.0	6.5	18.8	34.4
Random	1.0	6.6	13.9	18.6	1.0	6.8	17.8	25.7	1.0	7.9	23.1	43.4

Table 8.2: Mean value for portfolios obtained by each method under OWC. Best results are in bold.

	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work ($\alpha = 0.01$)	3.9	14.0	21.0	26.1	5.6	22.8	37.7	49.0	**11.3**	**50.8**	**94.0**	**131.6**
Greedy	**4.3**	**14.3**	**21.2**	**26.2**	**6.2**	**23.6**	**38.2**	**49.9**	**11.3**	50.7	92.9	130.3
Degree	3.3	10.0	14.8	19.7	5.2	21.0	30.8	40.7	9.9	40.4	68.9	93.6
Random	3.8	12.6	18.9	22.6	4.5	18.3	31.0	38.2	9.1	45.6	86.3	119.6

Table 8.3: CVaR for portfolios obtained by each method under $\textsc{uc}_{0.1}$. Best results are in bold.

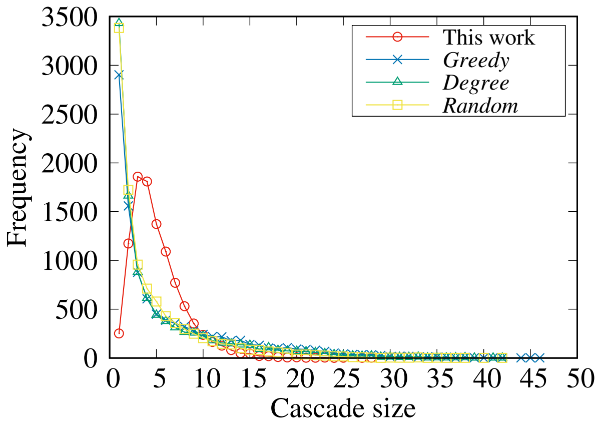
$\alpha = 0.01$	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work	**1.1**	**5.8**	**11.3**	**16.3**	**1.0**	**5.7**	**13.5**	**20.9**	1,809.2	**2,187.1**	2,083.8	**2,199.1**
Greedy	1.0	5.7	10.8	15.8	**1.0**	5.0	11.6	18.6	1,941.7	2,078.0	2,127.6	2,155.1
Degree	1.0	5.7	10.3	15.0	**1.0**	5.0	11.5	18.6	**2,181.3**	2,181.3	**2,181.3**	2,181.3
Random	1.0	5.0	10.0	15.0	**1.0**	5.0	10.4	16.1	1.0	5.0	2,166.3	2,190.9
$\alpha = 0.05$	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work	**1.3**	**6.6**	**12.1**	**17.0**	**1.0**	**6.4**	**14.6**	**22.4**	2,125.0	**2,205.8**	2,189.8	**2,217.8**
Greedy	1.0	6.4	11.6	16.6	**1.0**	5.6	13.1	20.4	2,151.7	2,183.7	2,198.6	2,208.9
Degree	1.0	6.5	11.0	15.7	**1.0**	5.7	12.7	20.2	**2,200.1**	2,200.1	2,200.1	2,200.1
Random	1.0	5.0	10.0	15.5	**1.0**	5.0	11.3	17.3	1.0	538.5	**2,202.4**	2,209.7

Table 8.4: Mean value for portfolios obtained by each method under $\textsc{uc}_{0.1}$. Best results are in bold.

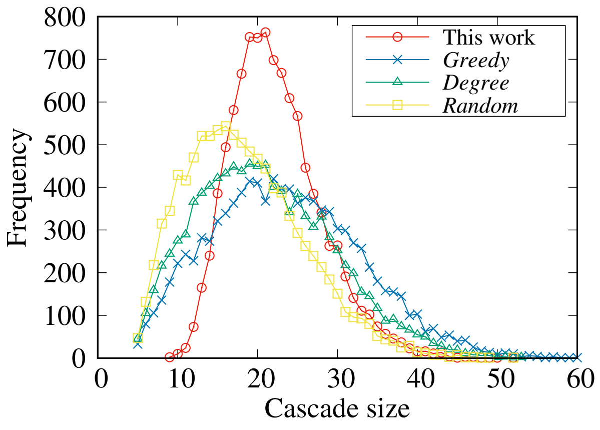
	Karate dataset				Physicians dataset				Advogato dataset			
Seed size k	1	5	10	15	1	5	10	15	1	5	10	15
This work ($\alpha = 0.01$)	3.4	10.6	**15.8**	20.2	2.4	**11.5**	21.0	**29.3**	2,260.8	**2,270.1**	2,275.0	**2,282.1**
Greedy	**3.5**	**10.7**	**15.8**	**20.4**	**2.5**	**11.5**	**21.2**	**29.3**	2,262.2	2,268.9	**2,275.5**	2,281.6
Degree	**3.5**	10.5	14.6	18.9	2.3	**11.5**	19.7	28.2	**2,264.7**	2,264.7	2,264.7	2,264.7
Random	1.6	7.8	14.2	19.4	2.0	9.3	17.3	23.9	584.8	2,182.9	2,271.0	2,274.2

Table 8.5: Number of positive weights of portfolios obtained by our method ($\alpha = 0.01, owc$).

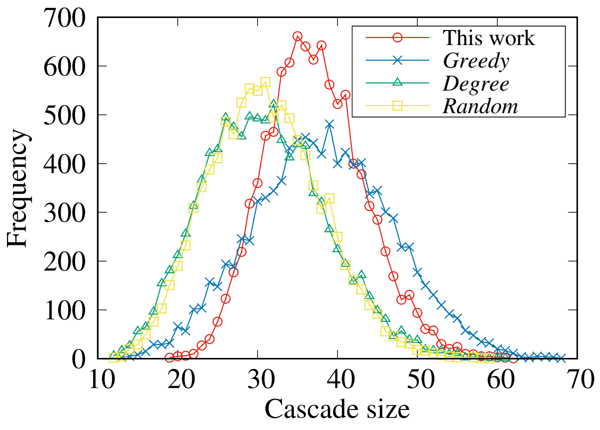
Seed size k	1	5	10	15
Karate dataset	9	31	36	30
Physicians dataset	10	39	42	45
Advogato dataset	5	36	44	51



(a) Seed size $k = 1$

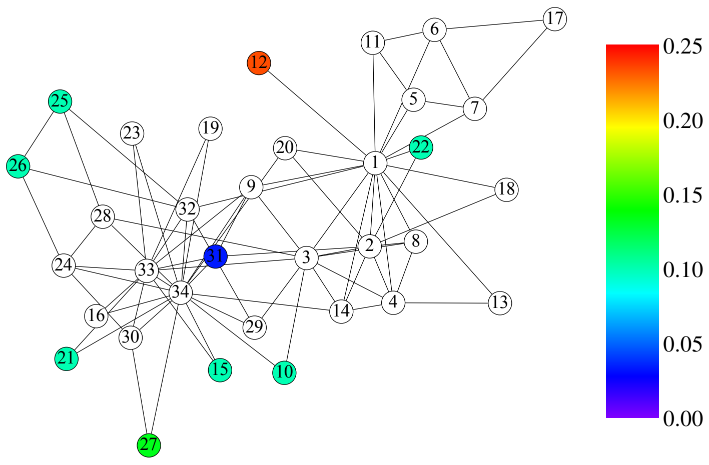


(b) Seed size $k = 5$



(c) Seed size $k = 10$

Figure 8.2: Histogram of cascade sizes for portfolios constructed by each algorithm on Physicians ($\alpha = 0.01$, OWC).



v	π_v
12	0.233
27	0.133
10	0.100
15	0.100
21	0.100
22	0.100
25	0.100
26	0.100
31	0.033

Figure 8.3: Visualization of Karate network. Each vertex is colored according to its portfolio weight ($\alpha = 0.01$, OWC).

Chapter 9

Conclusions

In this thesis, we studied the influence maximization problem, which is a central graph problem in computational social influence. While influence maximization has been made more suitable for practical situations, solving itself has remained as a difficult task. This difficulty comes from the massive scale of today's real-world networks and the probabilistic nature of diffusion process. We approached these issues with algorithmic studies in two aspects.

Efficient Computation

Summary. In the first aspect, we explored how to boost influence maximization in practice. Our tool for this purpose was the empirical observation of diffusive behaviors (Chapter 4). We conducted intensive experimental studies in order to grasp the trend of the diffusion process on real-world networks. In particular, we revealed the differences between the strategies of influence probability assignment, unweighted and degree-weighted. Then, we exploited the empirical observations to devise efficient algorithms under various situations. First, we proposed a fast influence maximization algorithm (Chapter 5). Our observation told us that descendant counting – the most challenging part – can be solved more quickly by using a simple linear-time preprocessing. We experimentally confirmed that the proposed technique plays a critical role in performance improvements.

Next, we developed a dynamic indexing algorithm for real-time influence analysis in evolving networks (Chapter 6). We first designed an index structure, query algorithm for influence maximization, and vanilla update algorithm for an index, and devised theoretical analysis. Then, we exploited our observation to speed-up index update algorithms. We experimentally verified that our speed-up techniques enabled our indexing algorithm to track dynamically-changing networks.

Then, we presented a reduction algorithm for massive influence networks (Chapter 7). Our observation provided a guideline to identify which part of influence graphs is redundant. We designed a coarsening strategy for influence graph reduction and the corresponding algorithms. We demonstrated that our coarsening strategy produced smaller graphs, and running influence maximization on them was much faster than on the input graph.

Throughout these applications, we demonstrated the effectiveness of our approach.

Future directions. While most of the previous research has aimed at the development of the single state-of-the-art, we observed that the best algorithm is different for different problem instances. This was first revealed in the benchmarking study paper of Arora, Galhotra, and Ranu [12]. Since its publication, researchers have been actively discussing this fact [13, 130]. Then, we need a systematic way for automatically choosing an appropriate algorithm according to an instance of influence maximization. Chapter 5 and [12] provide a guideline for such a selection strategy.

In addition, one may be interested in handling other diffusion models. For example, whereas it is not difficult to extend our methods for another well-established model called linear threshold, where the diffusion process is equivalent to the reachability on random graphs like the independent cascade model, our methods cannot directly manage time-independent activation trials.

Effective Strategies

Summary. In the second aspect, we aimed at answering the question "what is an effective strategy for influence diffusion?" We considered an effective strategy should avoid a risk ending with a few influenced individuals. To this end, we employed portfolio optimization to optimize the conditional value at risk – a common approach for risk aversion – and presented the corresponding algorithm (Chapter 8). We found that our approach yielded a low-risk strategy while standard influence maximization was not able to resolve that risk.

Future directions. While our algorithm was able to construct a risk-averse portfolio on vertex sets that achieves a higher conditional value at risk compared to a single vertex set, we still have a number of possible choices for both the objective function (e.g., the conditional value at risk) and the representation of solutions (e.g., portfolios on sets). We need further investigation from several aspects, e.g., how easy to interpret the solution is. For example, Wilder [186] proposes to maximize the conditional value at risk of continuous submodular functions, where we are able to assign a continuous value to each element in the ground set.

References

[1] Gnu octave. https://www.gnu.org/software/octave/.

[2] https://www.statisticbrain.com/instagram-company-statistics/. Accessed on December 1st, 2017.

[3] Influence maximization for social good. http://teamcore.usc.edu/people/SocialGood/. Accessed on Devember 7th, 2017.

[4] https://www.statisticbrain.com/youtube-statistics/. Accessed on December 1st, 2017.

[5] Twitter usage statistics, 2017. URL http://www.internetlivestats.com/twitter-statistics/. Accessed on December 1st, 2017.

[6] An exhaustive study of twitter users across the world, 2017. URL http://www.beevolve.com/twitter-statistics/. Accessed on December 1st, 2017.

[7] Bruno D. Abrahao, Flavio Chierichetti, Robert Kleinberg, and Alessandro Panconesi. Trace complexity of network inference. In *KDD – Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 491–499, 2013.

[8] Carlo Acerbi and Dirk Tasche. On the coherence of expected shortfall. *Journal of Banking & Finance*, 26(7):1487–1503, 2002.

[9] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *CIKM – Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 909–918, 2013.

[10] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *KDD – Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 7–15, 2008.

[11] Akhil Arora, Galhotra Sainyam, and Ranu Sayan. im_benchmarking repository. https://github.com/sigdata/im_benchmarking. Accessed on September 30th, 2017.

[12] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD – Proceedings of the 43rd ACM SIGMOD International Conference on Management of Data*, pages 651–666, 2017.

[13] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. Refutations to ”refutations on debunking the myths of influence maximization: An in-depth benchmarking study”. Technical report, 2017.

[14] Sanjeev Arora, Elad Hazan, and Satyen Kale. $o(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. *SIAM Journal on Computing*, 39(5):1748–1771, 2010.

[15] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[16] Philippe Artzner, Freddy Delbaen, Jean Marc Eber, and David Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[17] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada A. Adamic. The role of social networks in information diffusion. In *WWW – Proceedings of the 21st International Conference on World Wide Web*, pages 519–528, 2012.

[18] Michael O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980.

[19] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[20] Nicola Barbieri and Francesco Bonchi. Influence maximization with viral product design. In *SDM – Proceedings of the 14th SIAM International Conference on Data Mining*, pages 55–63, 2014.

[21] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. In *ICDM – Proceedings of the 12th IEEE International Conference on Data Mining*, pages 81–90, 2012.

[22] Frank M. Bass. A new product growth model for consumer durables. *Management Science*, 15(5):215–227, 1969.

[23] Jonathan Berry, William E. Hart, Cynthia A. Phillips, James G. Uber, and Jean-Paul Watson. Sensor placement in municipal water networks with temporal integer programming models. *Journal of Water Resources Planning and Management*, 132(4):218–224, 2006.

[24] Garrett Birkhoff. On the combination of subalgebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 29(4):441–464, 1933.

[25] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *WWW – Proceedings of the 13th International Conference on World Wide Web*, pages 595–602, 2004.

[26] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW – Proceedings of the 20th International Conference on World Wide Web*, pages 587–596, 2011.

[27] Michele Borassi. *Algorithms for metric properties of large real-world networks from theory to practice and back.* PhD thesis, IMT School for Advanced Studies Lucca, 2016.

[28] Michele Borassi. A note on the complexity of computing the number of reachable vertices in a digraph. *Information Processing Letters*, 116(10): 628–630, 2016.

[29] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *SODA – Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 946–957, 2014.

[30] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. *CoRR*, abs/1212.0884v5, 2016.

[31] Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *ICCV – Proceedings of the 8th International Conference on Computer Vision*, pages 105–112, 2001.

[32] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[33] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *WWW – Proceedings of the 20th International Conference on World Wide Web*, pages 665–674, 2011.

[34] Javier Calatrava and Alberto Garrido. Spot water markets and risk in water supply. *Agricultural Economics*, 33(2):131–143, 2005.

[35] Meeyoung Cha, Mislove. Alan, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *WWW – Proceedings of the 18th International Conference on World Wide Web*, pages 721–730, 2009.

[36] Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. Recommendations to boost content spread in social networks. In *WWW – Proceedings of the 21st International Conference on World Wide Web*, pages 529–538, 2012.

[37] Ning Chen. On the approximability of influence in social networks. In *SODA – Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1037, 2008.

[38] Ning Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.

[39] Wei Chen. Computational social influence. In *SocInf – Proceedings of the 1st International Workshop on Social Influence Analysis*, pages 1–1, 2015.

[40] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD – Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 199–208, 2009.

[41] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD – Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1029–1038, 2010.

[42] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM – Proceedings of the 10th IEEE International Conference on Data Mining*, pages 88–97, 2010.

[43] Wei Chen, Wei. Lu, and Ning Zhang. Time-critical influence maximization in social networks with time-delayed diffusion process. In *AAAI – Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 592–598, 2012.

[44] Wei Chen, Laks V.S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[45] Yi-Cheng Chen, Wen-Chih Peng, and Suh-Yin Lee. Efficient algorithms for influence maximization in social networks. *Knowledge and Information Systems*, 33(3):577–601, 2012.

[46] Suqi Cheng, Huawei Shen, Junming Huang, Guoqing Zhang, and Xueqi Cheng. StaticGreedy: Solving the scalability-accuracy dilemma in influence maximization. In *CIKM – Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 509–518, 2013.

[47] Suqi Cheng, Huawei Shen, Junming Huang, Wei Chen, and Xueqi Cheng. IMRank: Influence maximization via finding self-consistent ranking. In *SIGIR – Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 475–484, 2014.

[48] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at Facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.

[49] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3): 441–453, 1997.

[50] Edith Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. In *PODS – Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 88–99, 2014.

[51] Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *PODC – Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing*, pages 225–234, 2007.

[52] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM – Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, pages 629–638, 2014.

[53] James S. Coleman, Elihu Katz, and Herbert Menzel. *Medical Innovation: A Diffusion Study.* Bobbs Merrill, 1966.

[54] Biru Cui, Shanchieh Jay Yang, and Christopher Homan. Non-independent cascade formation: Temporal and spatial effects. In *CIKM – Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, pages 1923–1926, 2014.

[55] Erik D. Demaine, Hajiaghayi. MohammadTaghi, Hamid Mahini, David L. Malec, S. Raghavan, Anshul Sawant, and Morteza Zadimoghaddam. How to influence people with partial incentives. In *WWW – Proceedings of the 23rd International Conference on World Wide Web*, pages 937–948, 2014.

[56] Luke Dickens, Ian Molloy, Jorge Lobo, Pau-Chen Cheng, and Alessandra Russo. Learning stochastic models of information flow. In *ICDE – Proceedings of the 28th International Conference on Data Engineering*, pages 570–581, 2012.

[57] Thang Dinh, Hung Nguyen, Preetam Ghosh, and Michael Mayo. Social influence spectrum with guarantees: Computing more in less time. In *CSoNet – Proceedings of the 4th International Conference on Computational Social Networks*, pages 84–103, 2015.

[58] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *KDD – Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 57–66, 2001.

[59] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS – Proceedings of the Advances in Neural Information Processing Systems 26*, pages 3147–3155, 2013.

[60] Nan Du, Yingyu Liang, Maria-Florina Balcan, and Le Song. Influence function learning in information diffusion networks. In *ICML – Proceedings of the 31st International Conference on Machine Learning*, pages 2016–2024, 2014.

[61] Eyal Even-Dar and Asaf Shapira. A note on maximizing the spread of influence in social networks. In *WINE – Proceedings of the 3rd International Workshop on Internet and Network Economics*, pages 281–286, 2007.

[62] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM – Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 251–262, 1999.

[63] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[64] Shanshan Feng, Xuefeng Chen, Gao Cong, Yifeng Zeng, Yeow Meng Chee, and Yanping Xiang. Influence maximization with novelty decay in social networks. In *AAAI – Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 37–43, 2014.

[65] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[66] Satoru Fujishige. Polymatroidal dependence structure of a set of random variables. *Information and Control*, 39(1):55–72, 1978.

[67] Sainyam Galhotra, Akhil Arora, and Shourya Roy. Holistic influence maximization: Combining scalability and efficiency with opinion-aware models. In *SIGMOD – Proceedings of the 42nd ACM SIGMOD International Conference on Management of Data*, pages 743–758, 2016.

[68] Wojciech Galuba, Karl Aberer, Dipanjan Chakraborty, Zoran Despotovic, and Wolfgang Kellerer. Outtweeting the twitterers - predicting information cascades in microblogs. In *WOSN – Proceedings of the 3rd Wonference on Online Social Networks*, 2010.

[69] David L Gibbs and Ilya Shmulevich. Solving the influence maximization problem reveals regulatory organization of the yeast cell cycle. *PLoS computational biology*, 13(6):e1005591, 2017.

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

[71] Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 9(3):1–18, 2001.

[72] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[73] Manuel Gomez-Rodriguez and Bernhard Schölkopf. Influence maximization in continuous time diffusion networks. In *ICML – Proceedings of the 29th International Conference on Machine Learning*, pages 313–320, 2012.

[74] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *KDD – Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1019–1028, 2010.

[75] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *KDD – Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1019–1028, 2010.

[76] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML – Proceedings of the 28th International Conference on Machine Learning*, pages 561–568, 2011.

[77] Manuel Gomez-Rodriguez, Le Song, Hadi Daneshmand, and Bernhard Schölkopf. Estimating diffusion networks: Recovery conditions, sample complexity and soft-thresholding algorithm. *Journal of Machine Learning Research*, 17:90:1–90:29, 2016.

[78] Amit Goyal. Source code release of amit goyal's home page. https://www.cs.ubc.ca/~goyal/code-release.php. Accessed on September 30th, 2017.

[79] Amit Goyal, Francesco Bonchi, and Laks V.S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM – Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, pages 241–250, 2010.

[80] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. CELF++: Optimizing the greedy algorithm for influence maximization in social networks. In *WWW – Proceedings of the 20th International Conference on World Wide Web*, pages 47–48, 2011.

[81] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. SIMPATH: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM – Proceedings of the 11th IEEE International Conference on Data Mining*, pages 211–220, 2011.

[82] 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*, 3(2): 179–192, 2013.

[83] Mark Granovetter. Threshold models of collective behavior. *The American Journal of Sociology*, 83(6):1420–1443, 1978.

[84] Michael D. Grigoriadis and Leonid G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

[85] Adrien Guille and Hakim Hacid. A predictive model for the temporal dynamics of information diffusion in online social networks. In *WWW – Proceedings of the 21st International Conference on World Wide Web, Companion Volume*, pages 1145–1152, 2012.

[86] Jing Guo, Peng Zhang, Chuan Zhou, Yanan Cao, and Li Guo. Personalized influence maximization on social networks. In *CIKM – Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 199–208, 2013.

[87] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[88] Paul W. Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative Group Studies*, 2(2):107–124, 1971.

[89] John Hopcroft and Robert Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[90] Keke Huang, Tang Youze, and Xiaokui Xiao. Influence Maximization via Martingales. https://sourceforge.net/projects/im-imm/, . Accessed on September 30th, 2017.

[91] Keke Huang, Tang Youze, and Xiaokui Xiao. Tim_plus. https://sourceforge.net/projects/timplus/, . Accessed on September 30th, 2017.

[92] Keke Huang, Sibo Wang, Glenn S. Bevilacqua, Xiaokui Xiao, and Laks V. S. Lakshmanan. Revisiting the stop-and-stare algorithms for influence maximization. *Proceedings of the VLDB Endowment*, 10(9):913–924, 2017.

[93] Mohsen Jamali. http://www.cs.ubc.ca/~jamalim/datasets/. Accessed on October 7th, 2015.

[94] Qingye Jiang, Guojie Song, Gao Cong, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *AAAI – Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 127–132, 2011.

[95] Kyuri Jo, Inuk Jung, Ji Hwan Moon, and Sun Kim. Influence maximization in time bounded network identifies transcription factors regulating perturbed pathways. *Bioinformatics*, 32(12):i128–i136, 2016.

[96] Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *ICDM – Proceedings of the 12th IEEE International Conference on Data Mining*, pages 918–923, 2012.

[97] David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, 1999.

[98] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD – Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[99] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *ICALP – Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 1127–1138, 2005.

[100] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

[101] William O. Kermack and Anderson G. McKendrick. A contribution to the mathematical theory of epidemics. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 115, pages 700–721, 1927.

[102] Elias Boutros Khalil, Bistra N. Dilkina, and Le Song. Scalable diffusion-aware optimization of network topology. In *KDD – Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1226–1235, 2014.

[103] Sanjeev Khanna and Brendan Lucier. Influence maximization in undirected networks. In *SODA – Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1482–1496, 2014.

[104] Jinha Kim, Seung-Keol Kim, and Hwanjo Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks. In *ICDE – Proceedings of the 29th International Conference on Data Engineering*, pages 266–277, 2013.

[105] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *PKDD – Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases*, pages 259–271, 2006.

[106] Masahiro Kimura, Kazumi Saito, and Ryohei Nakano. Extracting influential nodes for information diffusion on a social network. In *AAAI – Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1371–1376, 2007.

[107] Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Blocking links to minimize contamination spread in a social network. *ACM Transactions on Knowledge Discovery from Data*, 3(2):9:1–9:23, 2009.

[108] Masahiro Kimura, Kazumi Saito, Ryohei Nakano, and Hiroshi Motoda. Extracting influential nodes on a social network for information diffusion. *Data Mining and Knowledge Discovery*, 20(1):70–97, 2010.

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

[110] Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

[111] Jérôme Kunegis. KONECT – the Koblenz Network Collection. In *WWW – Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350, 2013.

[112] Jérôme Kunegis. Konect Network Dataset – KONECT. `http://konect.uni-koblenz.de/networks/konect`, April 2017.

[113] Konstantin Kutzkov, Albert Bifet, Francesco Bonchi, and Aristides Gionis. Strip: Stream learning of influence probabilities. In *KDD – Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 275–283, 2013.

[114] Luigi Laura and Federico Santaroni. Computing strongly connected components in the streaming model. In *TAPAS – Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems*, pages 193–205, 2011.

[115] Siyu Lei, Silviu Maniu, Luyi Mo, Reynold Cheng, and Pierre Senellart. Online influence maximization. In *KDD – Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–654, 2015.

[116] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[117] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2, 2007.

[118] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *KDD – Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 420–429, 2007.

[119] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S. Glance, and Matthew Hurst. Cascading behavior in large blog graphs. In *SDM – Proceedings of the 7th SIAM International Conference on Data Mining*, pages 551–556, 2007.

[120] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW – Proceedings of the 17th International Conference on World Wide Web*, pages 695–704, 2008.

[121] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-Lee Tan, and Wen-Syan Li. Efficient location-aware influence maximization. In *SIGMOD – Proceedings of the 40th ACM SIGMOD International Conference on Management of Data*, pages 87–98, 2014.

[122] Hui Li, Sourav S. Bhowmick, and Aixin Sun. CINEMA: conformity-aware greedy algorithm for influence maximization in online social networks. In *EDBT – Proceedings of the 16th International Conference on Extending Database Technology*, pages 323–334, 2013.

[123] Xiang Li, J. David Smith, Thang N. Dinh, and My T. Thai. Why approximate when you can get the exact? optimal targeted viral marketing at scale. In *INFOCOM – Proceedings of the 36th IEEE International Conference on Computer Communications*, pages 1–9, 2017.

[124] Yanhua Li, Wei Chen, Yajun Wang, and Zhi-Li Zhang. Influence diffusion dynamics and influence maximization in social networks with friend and foe relationships. In *WSDM – Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 657–666, 2013.

[125] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *ACL-HLT – Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, 2011.

[126] Hui Lin and Jeff Bilmes. Learning mixtures of submodular shells with application to document summarization. In *UAI – Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, pages 479–490, 2012.

[127] Bo Liu, Gao Cong, Dong Xu, and Yifeng Zeng. Time constrained influence maximization in social networks. In *ICDM – Proceedings of the 12th IEEE International Conference on Data Mining*, pages 439–448, 2012.

[128] Xiaodong Liu, Mo Li, Shanshan Li, Shaoliang Peng, Xiangke Liao, and Xiaopei Lu. IMGPU: GPU-accelerated influence maximization in large-scale social networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):136–145, 2014.

[129] Wei Lu and Laks V. S. Lakshmanan. Profit maximization over social networks. In *ICDM – Proceedings of the 12th IEEE International Conference on Data Mining*, pages 479–488, 2012.

[130] Wei Lu, Xiaokui Xiao, Amit Goyal, Keke Huang, and Laks V. S. Lakshmanan. Refutations on "Debunking the Myths of Influence Maximization: An In-Depth Benchmarking Study". *CoRR*, abs/1705.05144, 2017.

[131] Wei-Xue Lu, Peng Zhang, Chuan Zhou, Chunyi Liu, and Li Gao. Influence maximization in big networks: An incremental algorithm for streaming subgraph influence spread estimation. In *IJCAI – Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2076–2082, 2015.

[132] Brendan Lucier, Joel Oren, and Yaron Singer. Influence at scale: Distributed computation of complex contagion in networks. In *KDD – Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744, 2015.

[133] Hao Ma, Haixuan Yang, Michael R. Lyu, and Irwin King. Mining social networks using heat diffusion processes for marketing candidates selection. In *CIKM – Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, pages 233–242, 2008.

[134] Takanori Maehara. Risk averse submodular utility maximization. *Operations Research Letters*, 43(5):526–529, 2015.

[135] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *Proceedings of the VLDB Endowment*, 7(12):1023–1034, 2014.

[136] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[137] Michael Mathioudakis. http://queens.db.toronto.edu/~mathiou/spine/. Accessed on October 22nd, 2015.

[138] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *KDD – Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 529–537, 2011.

[139] Stanley Milgram. The small-world problem. *Psychology Today*, 1(1):61–67, 1967.

[140] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques*, 7:234–243, 1978.

[141] Edward F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292, 1959.

[142] Elchanan Mossel and Sebastien Roch. Submodularity of influence in social networks: From local to global. *SIAM Journal on Computing*, 39(6):2176–2188, 2010.

[143] Harikrishna Narasimhan, David C. Parkes, and Yaron Singer. Learnability of influence in networks. In *NIPS – Proceedings of the Advances in Neural Information Processing Systems 28*, pages 3186–3194, 2015.

[144] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[145] Praneeth Netrapalli and Sujay Sanghavi. Learning the graph of epidemic cascades. In *SIGMETRICS – ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, pages 211–222, 2012.

[146] Hung T. Nguyen and Thang N. Dinh. SSA/D-SSA Influence Maximization Algorithms. https://github.com/hungnt55/Stop-and-Stare. Accessed on September 30th, 2017.

[147] Hung T. Nguyen, Thang N. Dinh, and My T. Thai. Cost-aware targeted viral marketing in billion-scale networks. In *INFOCOM – Proceedings of the 35th IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[148] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *SIGMOD – Proceedings of the 42nd ACM SIGMOD International Conference on Management of Data*, pages 695–710, 2016.

[149] Hung T. Nguyen, Preetam Ghosh, Michael L. Mayo, and Thang N. Dinh. Social influence spectrum at scale: Near-optimal solutions for multiple budgets at once. *ACM Transactions on Information Systems*, 36(2):14:1–14:26, 2017.

[150] Hung T. Nguyen, Tri P. Nguyen, Tam N. Vu, and Thang N. Dinh. Outward influence and cascade size estimation in billion-scale networks. In *SIGMETRICS – Proceedings of the 2017 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, page 63, 2017.

[151] Hung T. Nguyen, Tri P. Nguyen, Tam N. Vu, and Thang N. Dinh. Outward influence and cascade size estimation in billion-scale networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1): 20:1–20:30, 2017.

[152] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. A billion-scale approximation algorithm for maximizing benefit in viral marketing. *IEEE/ACM Transactions on Networking*, 25(4):2419–2429, 2017.

[153] Huy Nguyen and Rong Zheng. Influence spread in large-scale social networks — a belief propagation approach. In *ECML PKDD – Proceedings of the 2012 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 515–530, 2012.

[154] Huy Nguyen and Rong Zheng. On budgeted influence maximization in social networks. *IEEE Journal on Selected Areas in Communications*, 31 (6):1084–1094, 2013.

[155] Naoto Ohsaka and Yuichi Yoshida. Portfolio optimization for influence spread. In *WWW – Proceedings of the 26th International Conference on World Wide Web*, pages 977–985, 2017. doi: 10.1145/3038912.3052628. URL http://doi.acm.org/10.1145/3038912.3052628.

[156] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *AAAI – Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 138–144, 2014.

[157] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Dynamic influence analysis in evolving networks. *Proceedings of the VLDB Endowment*, 9(12):1077–1088, 2016. doi: 10.14778/2994509.2994525. URL http://dx.doi.org/10.14778/2994509.2994525.

[158] Naoto Ohsaka, Tomohiro Sonobe, Sumio Fujita, and Ken-ichi Kawarabayashi. Coarsening massive influence networks for scalable diffusion analysis. In *SIGMOD – Proceedings of the 43rd ACM SIGMOD International Conference on Management of Data*, pages 635–650, 2017. doi: 10.1145/3035918.3064045. URL http://doi.acm.org/10.1145/3035918.3064045.

[159] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. In *FOCS – Proceedings of the IEEE 32nd Annual Symposium on Foundations of Computer Science*, pages 495–504, 1991.

[160] B. Aditya Prakash. http://people.cs.vt.edu/~badityap/CODE/coarsenet.tgz. Accessed on October 22nd, 2015.

[161] Manish Purohit, B. Aditya Prakash, Chanhyun Kang, Yao Zhang, and V.S. Subrahmanian. Fast influence-based coarsening for large networks. In *KDD – Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1296–1305, 2014.

[162] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD – Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70, 2002.

[163] R T Rockafellar and S Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–42, 2000.

[164] R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471, 2002.

[165] Everett M. Rogers. *Diffusion of Innovations*. Free Press, 1962.

[166] Bryce Ryan and Neal C. Gross. The diffusion of hybrid seed corn in two iowa communities. *Rural Sociology*, 8(1):15–24, 1943.

[167] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *KES – Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, pages 67–75, 2008.

[168] Kazumi Saito, Masahiro Kimura, Kouzou Ohara, and Hiroshi Motoda. Learning continuous-time information diffusion model for social behavioral data analysis. In *ACML – Proceedings of the 1st Asian Conference on Machine Learning*, pages 322–337, 2009.

[169] Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. Learning diffusion probability based on node attributes in social networks. In *ISMIS – Proceedings of the 19th International Symposium on Methodologies for Intelligent Systems*, pages 153–162, 2011.

[170] John K Salmon, Mark A Moraes, Ron O Dror, and David E Shaw. Parallel random numbers: as easy as 1, 2, 3. In *SC – Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[171] Thomas C. Schelling. *Micromotives and Macrobehavior.* Norton, 1978.

[172] Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers and Mathematics with Applications*, 7(1):67–72, 1981.

[173] Dravyansh Sharma, Ashish Kapoor, and Amit Deshpande. On greedy maximization of entropy. In *ICML – Proceedings of the 32nd International Conference on Machine Learning*, pages 1330–1338, 2015.

[174] Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *NIPS – Proceedings of the Advances in Neural Information Processing Systems 21*, pages 1577–1584, 2008.

[175] Mani R. Subramani and Balaji Rajagopalan. Knowledge-sharing and influence in online social networks via viral marketing. *Communications of the ACM*, 46(12):300–307, 2003.

[176] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD – Proceedings of the 40th ACM SIGMOD International Conference on Management of Data*, pages 75–86, 2014.

[177] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD – Proceedings of the 41st ACM SIGMOD International Conference on Management of Data*, pages 1539–1554, 2015.

[178] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[179] Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM – Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 245–254, 2012.

[180] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[181] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[182] Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012.

[183] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Distance-aware influence maximization in geo-social network. In *ICDE – Proceedings of the 32nd International Conference on Data Engineering*, pages 1–12, 2016.

[184] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD – Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1039–1048, 2010.

[185] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[186] Bryan Wilder. Risk-sensitive submodular optimization. In *AAAI – Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018. To appear.

[187] Amulya Yadav, Hau Chan, Albert Xin Jiang, Haifeng Xu, Eric Rice, and Milind Tambe. Using social networks to aid homeless shelters: Dynamic influence maximization under uncertainty. In *AAMAS – Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems*, pages 740–748, 2016.

[188] Amulya Yadav, Bryan Wilder, Eric Rice, Robin Petering, Jaih Craddock, Amanda Yoshioka-Maxwell, Mary Hemler, Laura Onasch-Vera, Milind Tambe, and Darlene Woo. Influence maximization in the field: The arduous journey from emerging to deployed application. In *AAMAS – Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pages 150–158, 2017.

[189] Yu Yang, Enhong Chen, Qi Liu, Biao Xiang, Tong Xu, and Shafqat Ali Shad. On approximation of real-world influence spread. In *ECML PKDD – Proceedings of the 2012 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 548–564, 2012.

[190] Miao Zhang, Chunni Dai, Chris Ding, and Enhong Chen. Probabilistic solutions of influence propagation on social networks. In *CIKM – Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 429–438, 2013.

[191] Chuan Zhou, Peng Zhang, Jing Guo, Xingquan Zhu, and Li Guo. Ublf: An upper bound based approach to discover influential nodes in social networks. In *ICDM – Proceedings of the 13th IEEE International Conference on Data Mining*, pages 907–916, 2013.

[192] Chuan Zhou, Peng Zhang, Jing Guo, and Li Guo. An upper bound based greedy algorithm for mining top-k influential nodes in social networks. In *WWW – Proceedings of the 23rd International Conference on World Wide Web, Companion Volume*, pages 421–422, 2014.

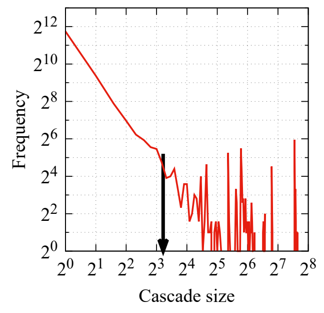
[193] Chuan Zhou, Peng Zhang, Wenyu Zang, and Li Guo. Maximizing the long-term integral influence in social networks under the voter model. In *WWW – Proceedings of the 23rd International Conference on World Wide Web, Companion Volume*, pages 423–424, 2014.

[194] Tao Zhou, Jiuxin Cao, Bo Liu, Shuai Xu, Ziqing Zhu, and Junzhou Luo. Location-based influence maximization in social networks. In *CIKM – Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 1211–1220, 2015.

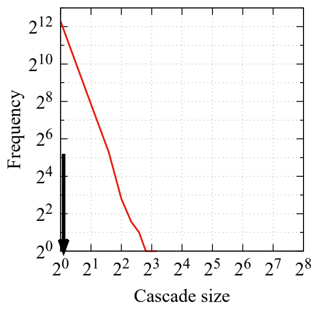
Appendix A

Additional Experimental Results in Chapter 4

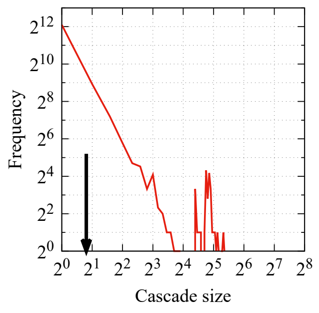
We here provide complete experimental results performed in Chapter 4. Figures A.1–A.18 show the size distribution of reachable sets for each configuration of network and influence probability setting. Figures A.19–A.32 show the scatter plot of the structure of RR sets for each configuration of network and influence probability setting.



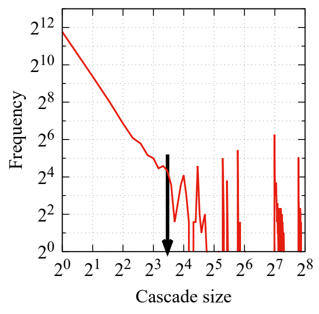
(a) ca-GrQc (UC$_{0.1}$)



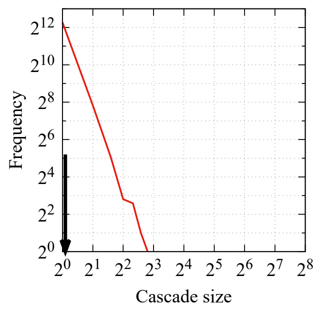
(b) ca-GrQc (UC$_{0.01}$)



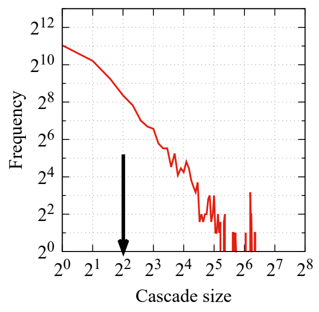
(c) ca-GrQc (TRI)



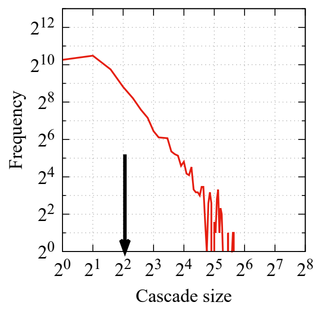
(d) ca-GrQc (EXP$_{0.1}$)



(e) ca-GrQc (EXP$_{0.01}$)

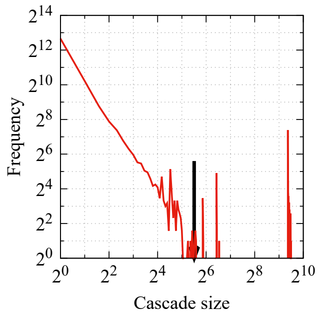


(f) ca-GrQc (IWC)

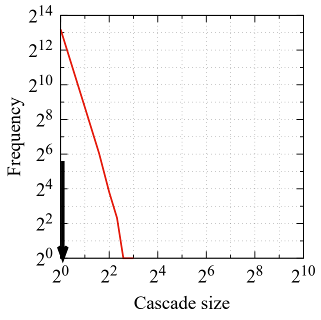


(g) ca-GrQc (OWC)

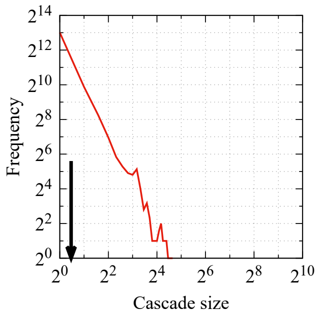
Figure A.1: Size distribution of reachable sets in ca-GrQc network.



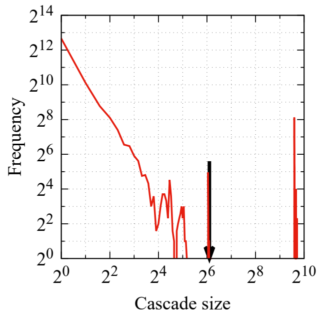
(a) ca-HepTh (UC$_{0.1}$)



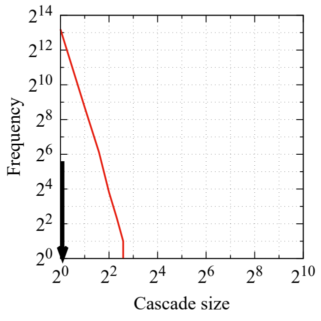
(b) ca-HepTh (UC$_{0.01}$)



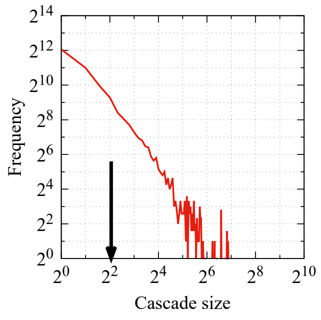
(c) ca-HepTh (TRI)



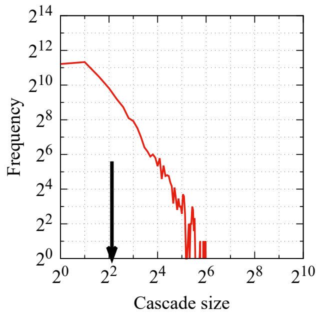
(d) ca-HepTh (EXP$_{0.1}$)



(e) ca-HepTh (EXP$_{0.01}$)

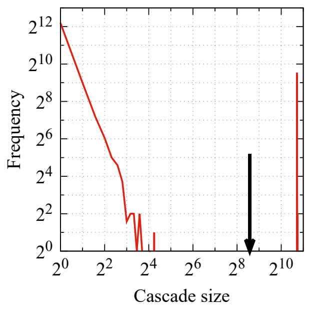


(f) ca-HepTh (IWC)

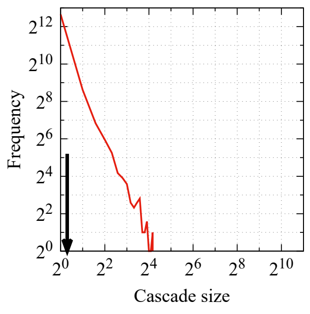


(g) ca-HepTh (OWC)

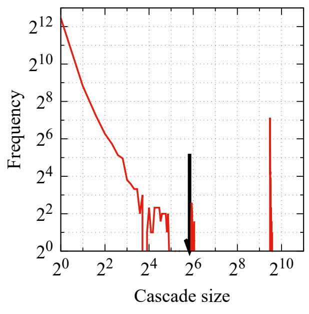
Figure A.2: Size distribution of reachable sets in ca-HepTh network.



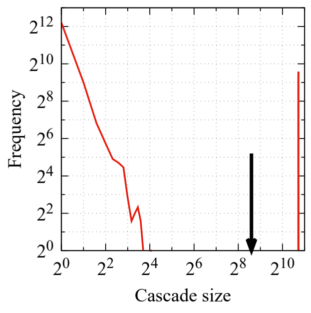
(a) wiki-Vote (UC$_{0.1}$)



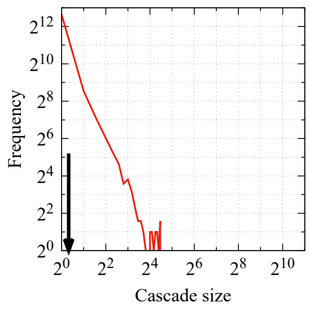
(b) wiki-Vote (UC$_{0.01}$)



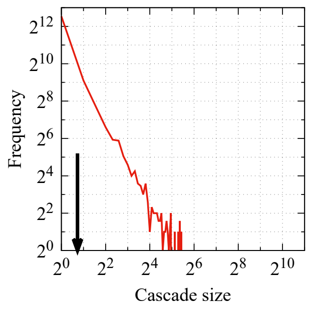
(c) wiki-Vote (TRI)



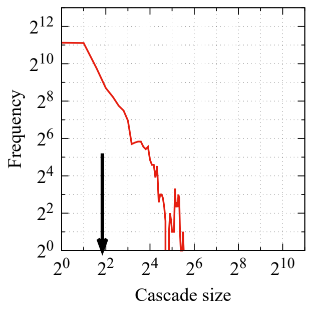
(d) wiki-Vote (EXP$_{0.1}$)



(e) wiki-Vote (EXP$_{0.01}$)

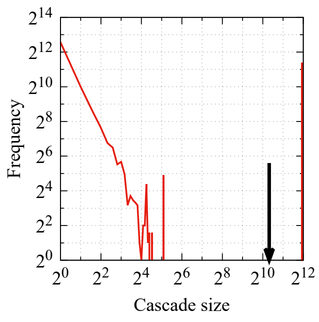


(f) wiki-Vote (IWC)

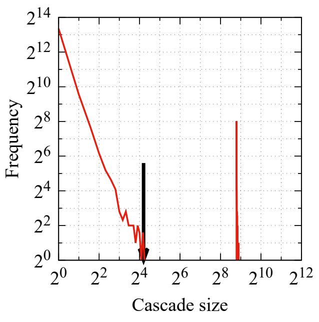


(g) wiki-Vote (OWC)

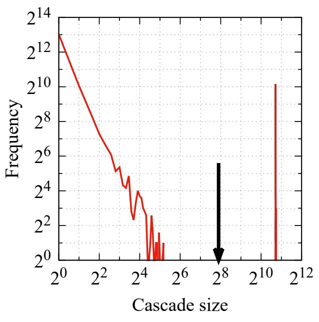
Figure A.3: Size distribution of reachable sets in wiki-Vote network.



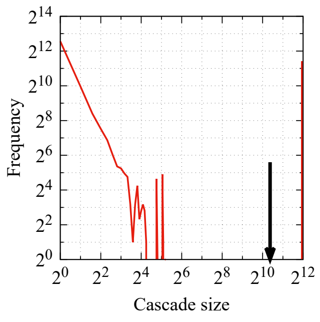
(a) ca-HepPh (UC$_{0.1}$)



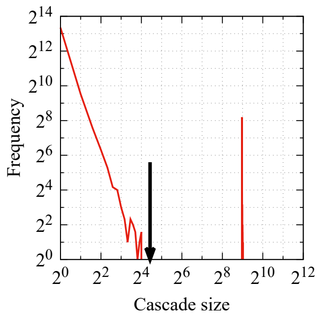
(b) ca-HepPh (UC$_{0.01}$)



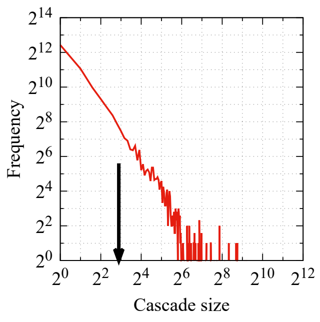
(c) ca-HepPh (TRI)



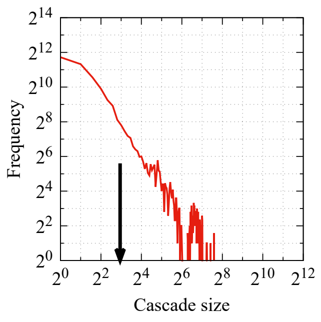
(d) ca-HepPh (EXP$_{0.1}$)



(e) ca-HepPh (EXP$_{0.01}$)

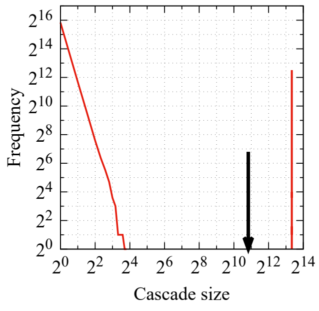


(f) ca-HepPh (IWC)

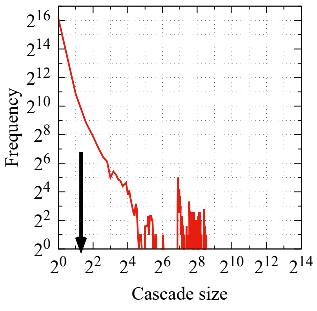


(g) ca-HepPh (OWC)

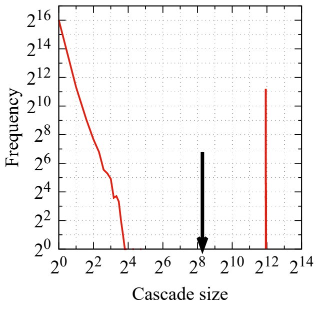
Figure A.4: Size distribution of reachable sets in ca-HepPh network.



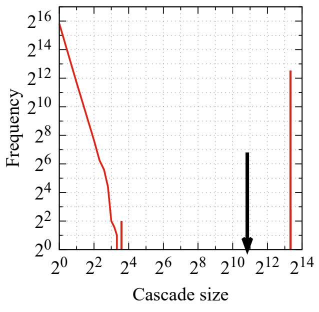
(a) soc-Epinions1 (UC$_{0.1}$)



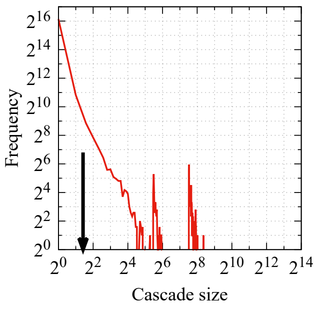
(b) soc-Epinions1 (UC$_{0.01}$)



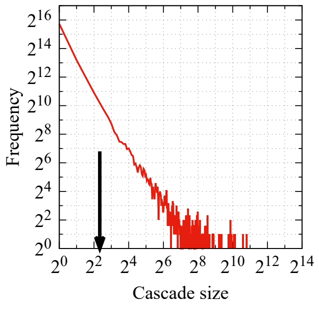
(c) soc-Epinions1 (TRI)



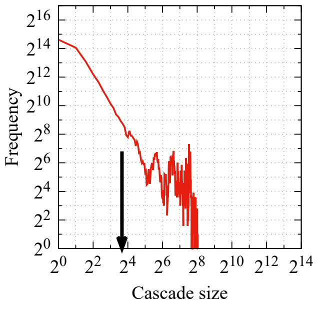
(d) soc-Epinions1 (EXP$_{0.1}$)



(e) soc-Epinions1 (EXP$_{0.01}$)

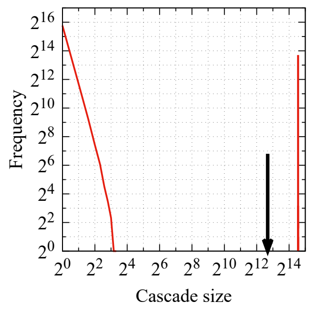


(f) soc-Epinions1 (IWC)

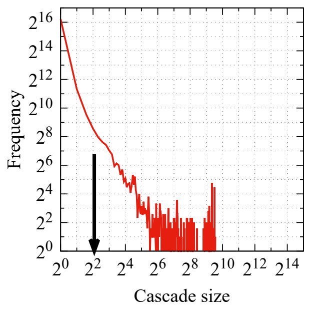


(g) soc-Epinions1 (OWC)

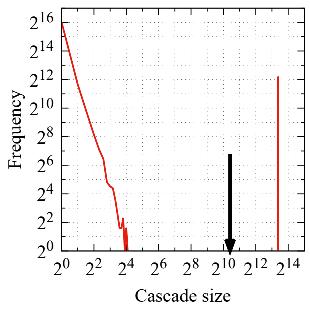
Figure A.5: Size distribution of reachable sets in soc-Epinions1 network.



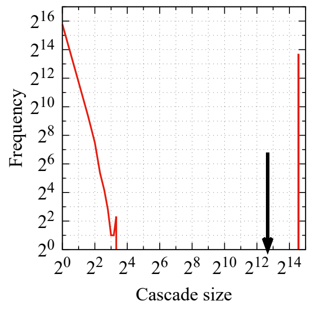
(a) soc-Slashdot0922 ($\mathrm{UC}_{0.1}$)



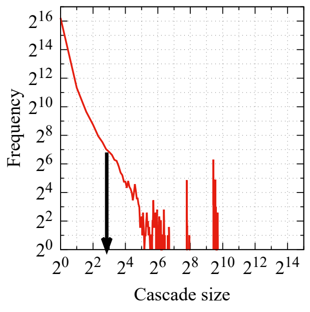
(b) soc-Slashdot0922 ($\mathrm{UC}_{0.01}$)



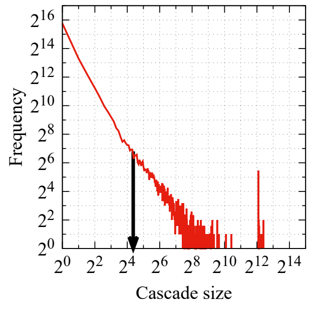
(c) soc-Slashdot0922 (TRI)



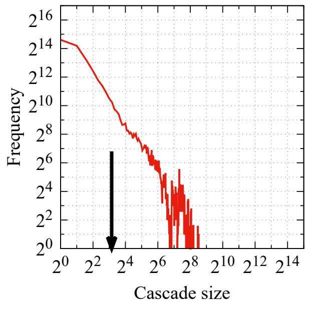
(d) soc-Slashdot0922 ($\mathrm{EXP}_{0.1}$)



(e) soc-Slashdot0922 ($\mathrm{EXP}_{0.01}$)

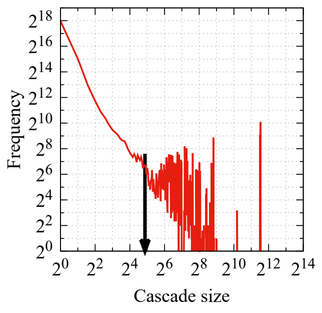


(f) soc-Slashdot0922 (IWC)

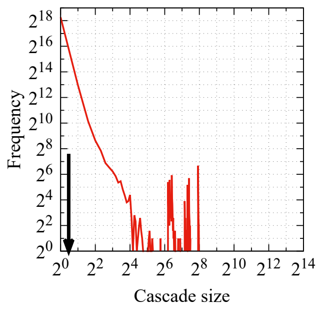


(g) soc-Slashdot0922 (OWC)

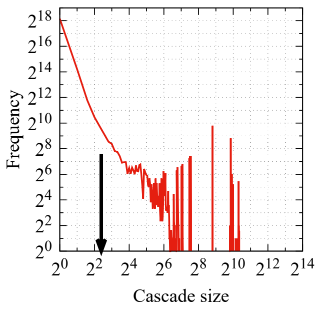
Figure A.6: Size distribution of reachable sets in soc-Slashdot0922 network.



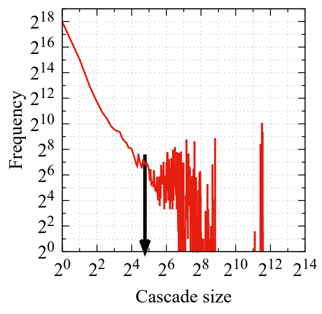
(a) web-NotreDame ($\textsc{uc}_{0.1}$)



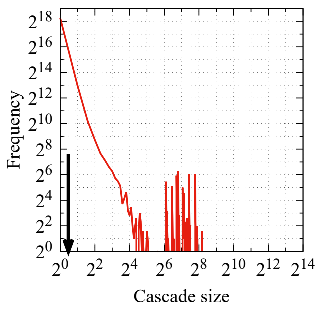
(b) web-NotreDame ($\textsc{uc}_{0.01}$)



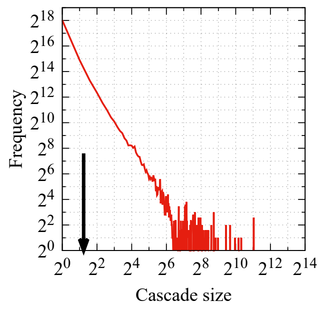
(c) web-NotreDame ($\textsc{tri}$)



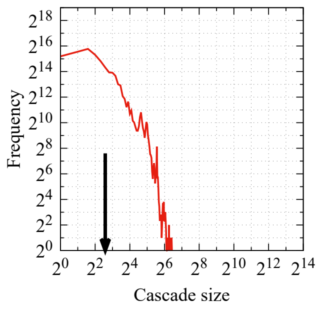
(d) web-NotreDame ($\textsc{exp}_{0.1}$)



(e) web-NotreDame ($\textsc{exp}_{0.01}$)

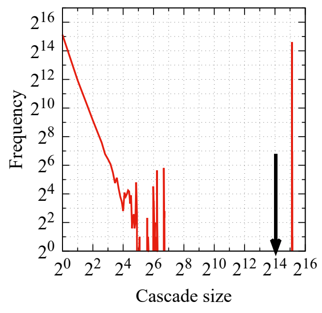


(f) web-NotreDame ($\textsc{iwc}$)

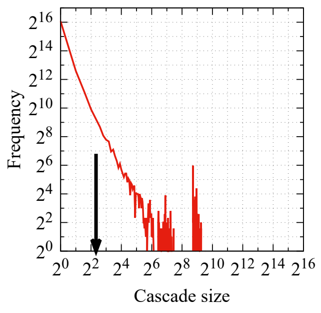


(g) web-NotreDame ($\textsc{owc}$)

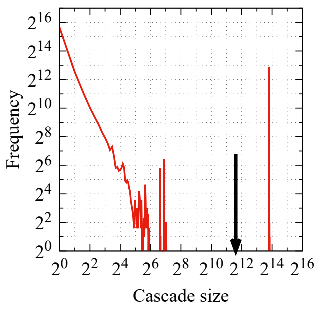
Figure A.7: Size distribution of reachable sets in web-NotreDame network.



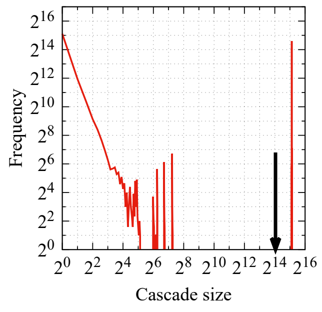
(a) ego-Twitter ($UC_{0.1}$)



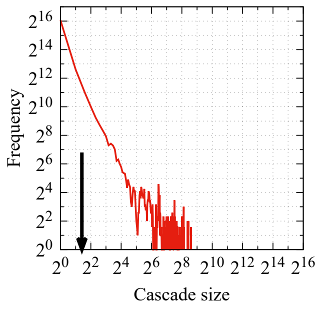
(b) ego-Twitter ($UC_{0.01}$)



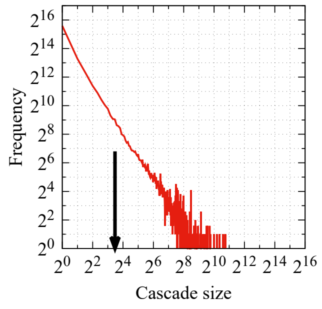
(c) ego-Twitter (TRI)



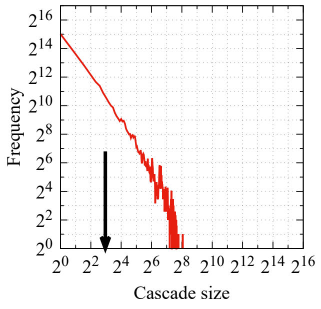
(d) ego-Twitter ($EXP_{0.1}$)



(e) ego-Twitter ($EXP_{0.01}$)

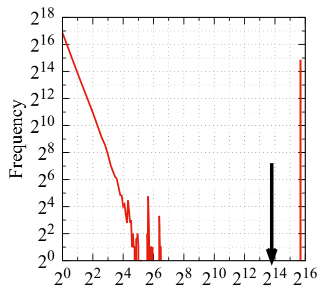


(f) ego-Twitter (IWC)

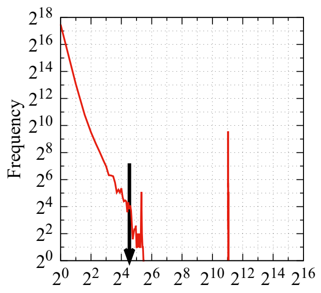


(g) ego-Twitter (OWC)

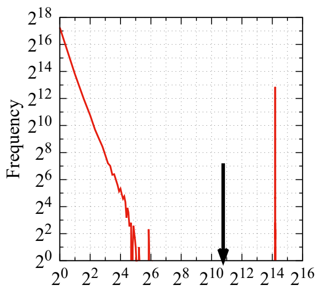
Figure A.8: Size distribution of reachable sets in ego-Twitter network.



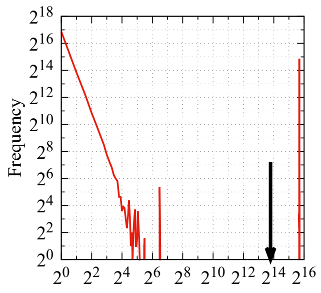
(a) loc-Gowalla ($\text{UC}_{0.1}$)



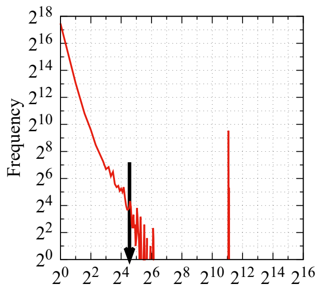
(b) loc-Gowalla ($\text{UC}_{0.01}$)



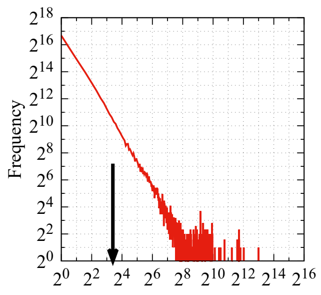
(c) loc-Gowalla (TRI)



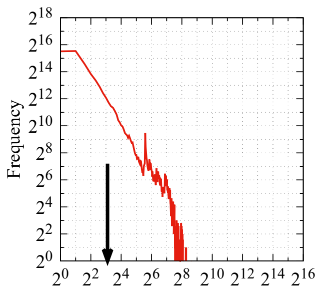
(d) loc-Gowalla ($\text{EXP}_{0.1}$)



(e) loc-Gowalla ($\text{EXP}_{0.01}$)

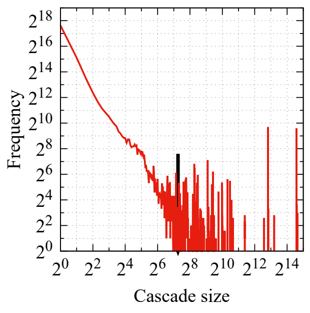


(f) loc-Gowalla (IWC)

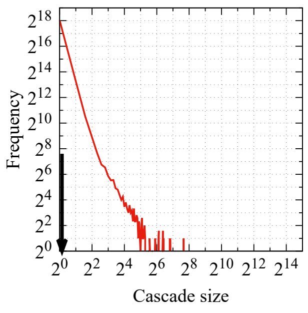


(g) loc-Gowalla (OWC)

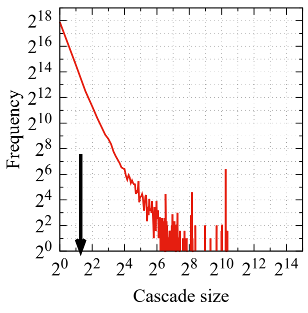
Figure A.9: Size distribution of reachable sets in loc-Gowalla network.



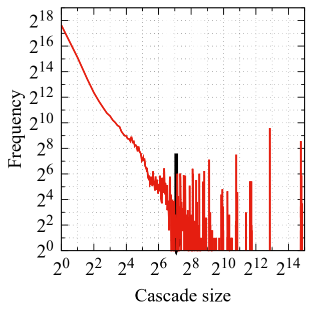
(a) web-Stanford ($\text{UC}_{0.1}$)



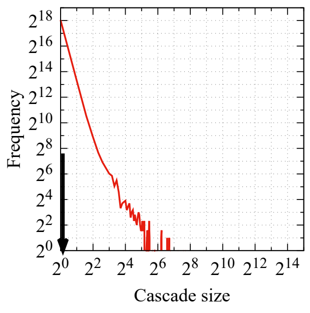
(b) web-Stanford ($\text{UC}_{0.01}$)



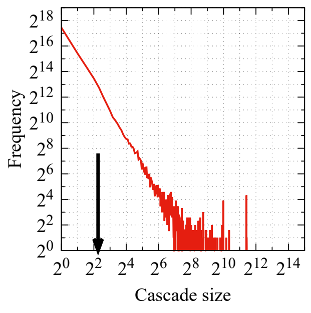
(c) web-Stanford (TRI)



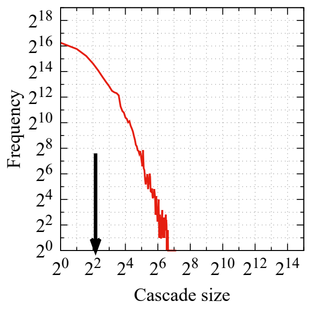
(d) web-Stanford ($\text{EXP}_{0.1}$)



(e) web-Stanford ($\text{EXP}_{0.01}$)

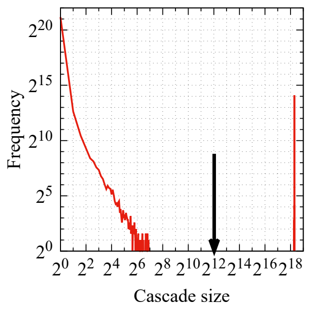


(f) web-Stanford (IWC)

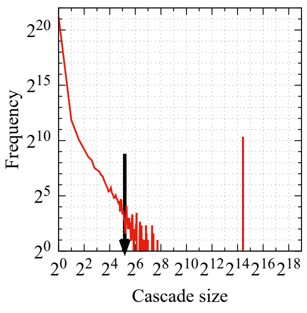


(g) web-Stanford (OWC)

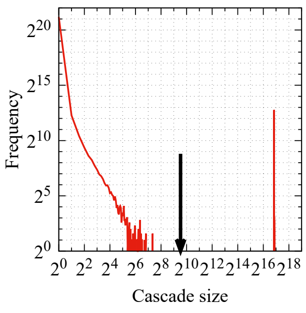
Figure A.10: Size distribution of reachable sets in web-Stanford network.



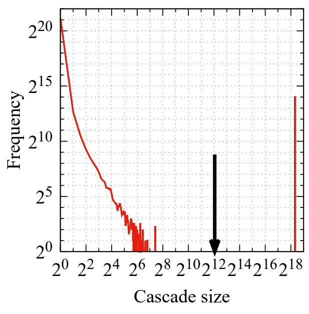
(a) wiki-Talk (UC$_{0.1}$)



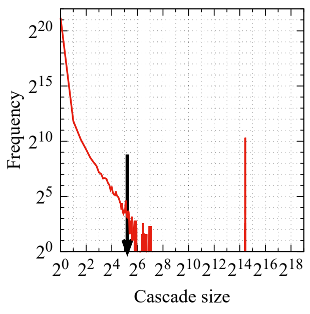
(b) wiki-Talk (UC$_{0.01}$)



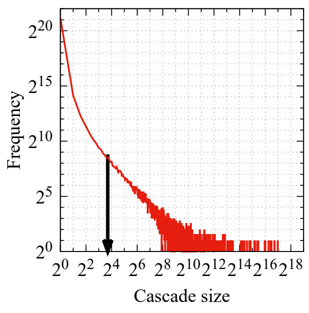
(c) wiki-Talk (TRI)



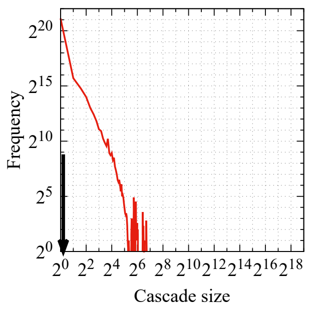
(d) wiki-Talk (EXP$_{0.1}$)



(e) wiki-Talk (EXP$_{0.01}$)

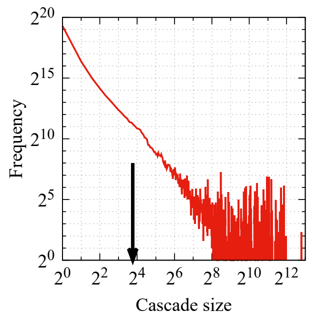


(f) wiki-Talk (IWC)

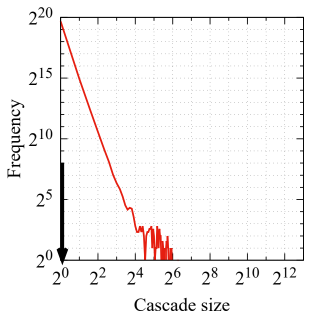


(g) wiki-Talk (OWC)

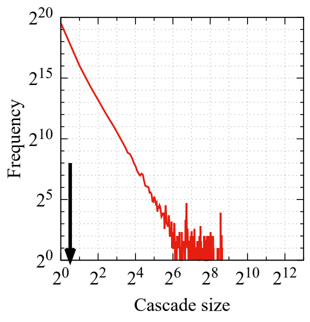
Figure A.11: Size distribution of reachable sets in wiki-Talk network.



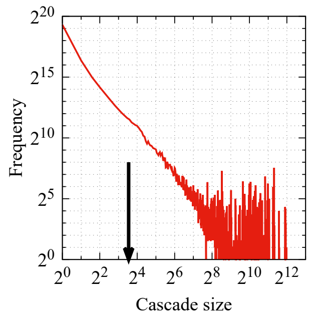
(a) web-Google ($\textsc{uc}_{0.1}$)



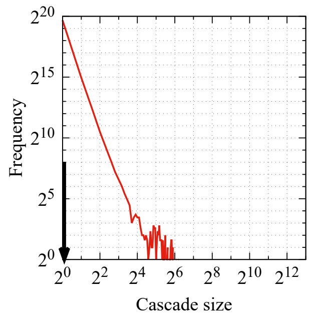
(b) web-Google ($\textsc{uc}_{0.01}$)



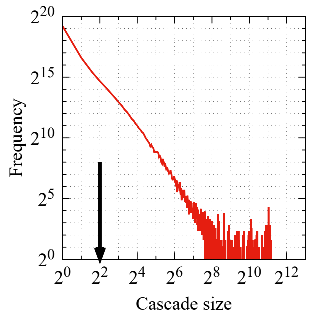
(c) web-Google ($\textsc{tri}$)



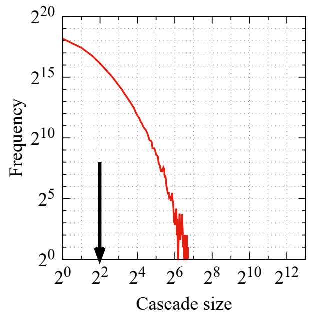
(d) web-Google ($\textsc{exp}_{0.1}$)



(e) web-Google ($\textsc{exp}_{0.01}$)

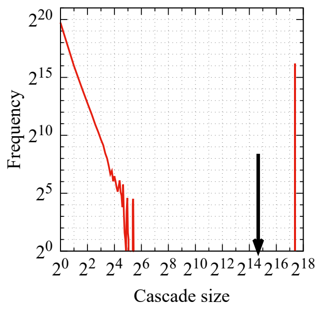


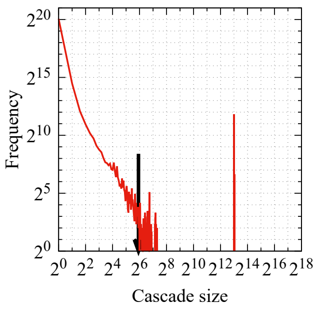
(f) web-Google ($\textsc{iwc}$)

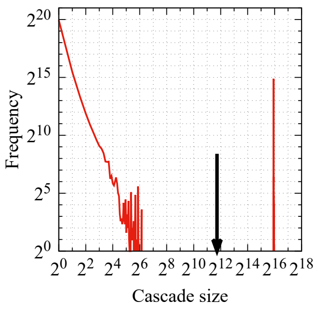


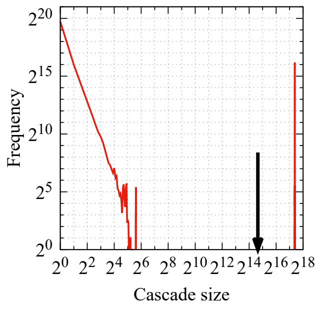
(g) web-Google ($\textsc{owc}$)

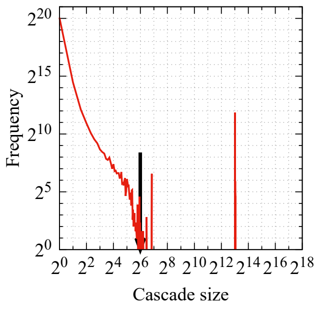
Figure A.12: Size distribution of reachable sets in web-Google network.

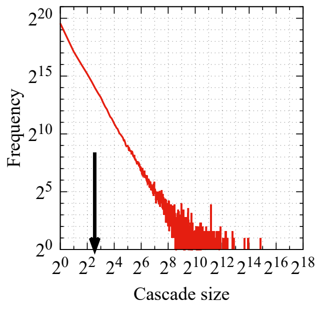

(a) com-Youtube ($\text{UC}_{0.1}$)


(b) com-Youtube ($\text{UC}_{0.01}$)


(c) com-Youtube (TRI)


(d) com-Youtube ($\text{EXP}_{0.1}$)


(e) com-Youtube ($\text{EXP}_{0.01}$)


(f) com-Youtube (IWC)

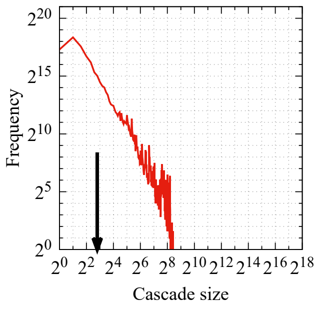
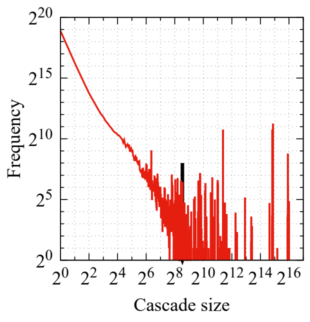
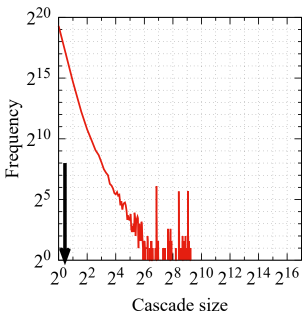

(g) com-Youtube (OWC)

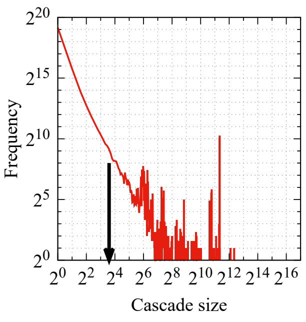
Figure A.13: Size distribution of reachable sets in com-Youtube network.



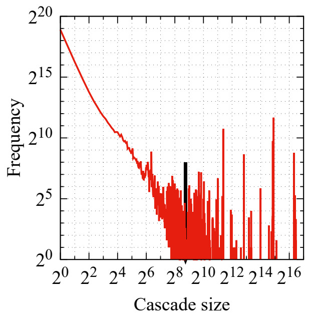
(a) web-BerkStan ($\textsc{uc}_{0.1}$)



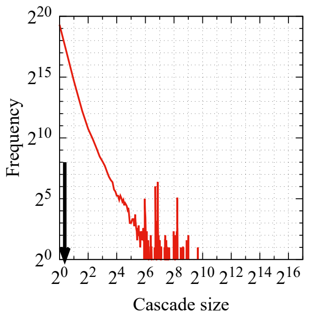
(b) web-BerkStan ($\textsc{uc}_{0.01}$)



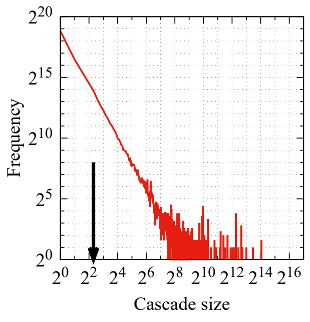
(c) web-BerkStan ($\textsc{tri}$)



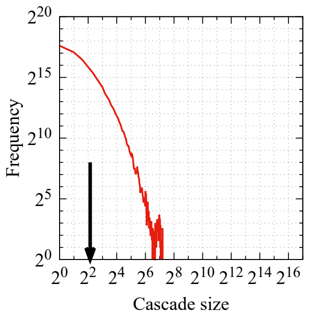
(d) web-BerkStan ($\textsc{exp}_{0.1}$)



(e) web-BerkStan ($\textsc{exp}_{0.01}$)

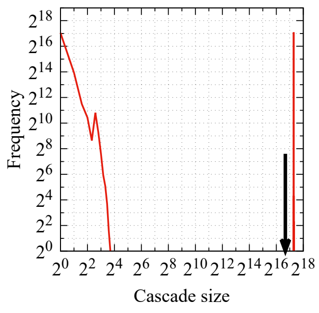


(f) web-BerkStan ($\textsc{iwc}$)

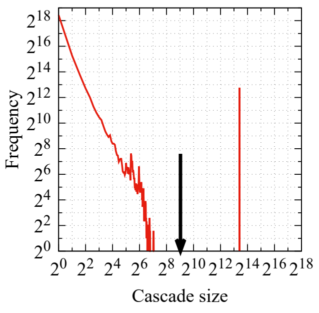


(g) web-BerkStan ($\textsc{owc}$)

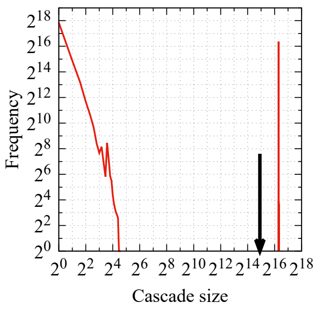
Figure A.14: Size distribution of reachable sets in web-BerkStan network.



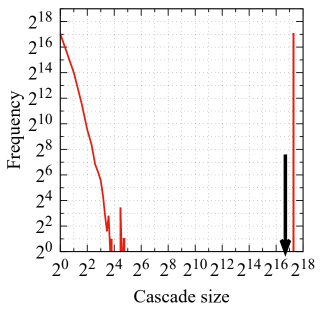
(a) higgs-twitter ($\text{UC}_{0.1}$)



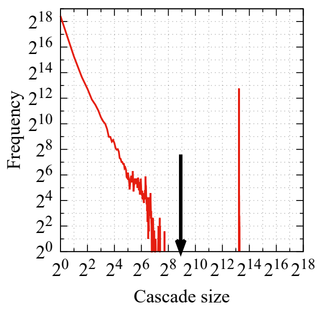
(b) higgs-twitter ($\text{UC}_{0.01}$)



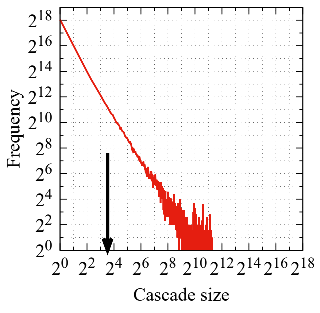
(c) higgs-twitter (TRI)



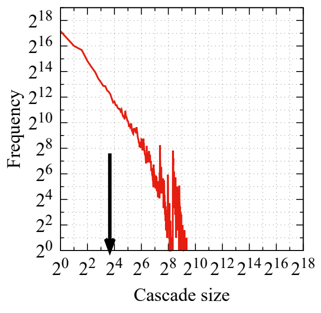
(d) higgs-twitter ($\text{EXP}_{0.1}$)



(e) higgs-twitter ($\text{EXP}_{0.01}$)

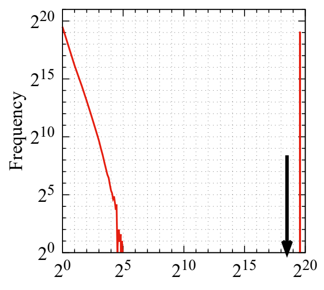


(f) higgs-twitter (IWC)

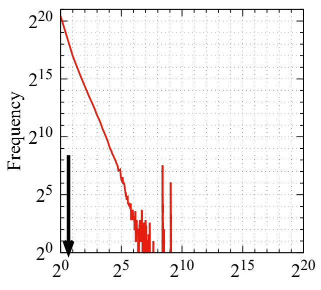


(g) higgs-twitter (OWC)

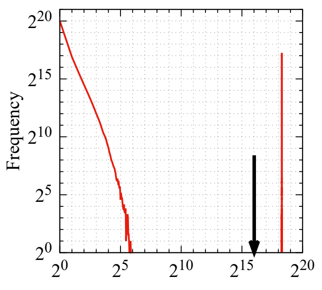
Figure A.15: Size distribution of reachable sets in higgs-twitter network.



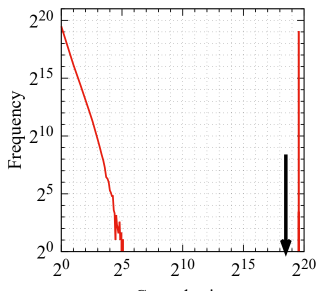
(a) soc-Pokec (UC$_{0.1}$)



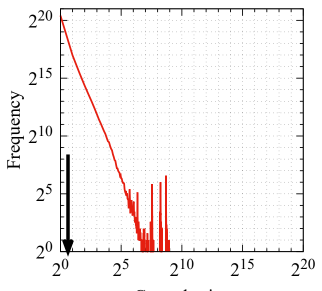
(b) soc-Pokec (UC$_{0.01}$)



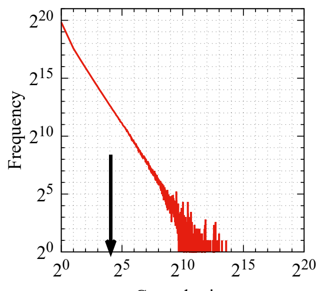
(c) soc-Pokec (TRI)



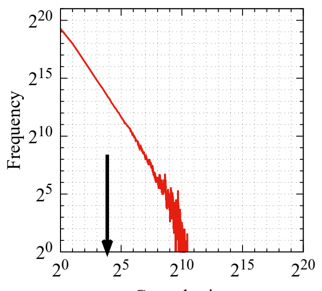
(d) soc-Pokec (EXP$_{0.1}$)



(e) soc-Pokec (EXP$_{0.01}$)

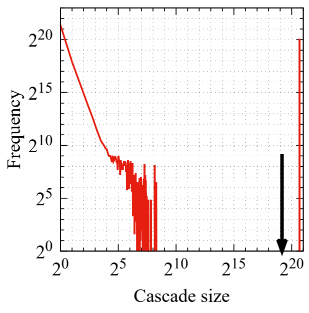


(f) soc-Pokec (IWC)

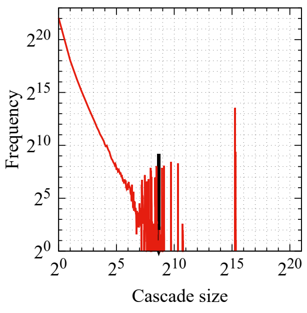


(g) soc-Pokec (OWC)

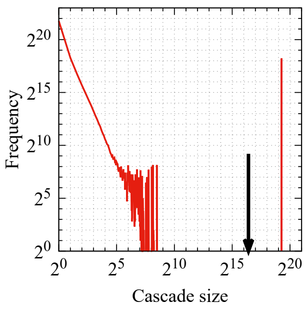
Figure A.16: Size distribution of reachable sets in soc-Pokec network.



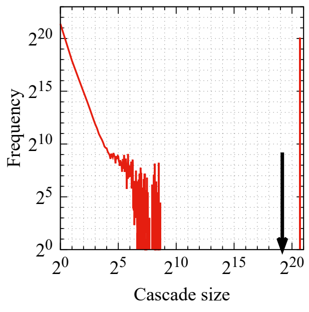
(a) soc-LiveJournal1 ($\text{UC}_{0.1}$)



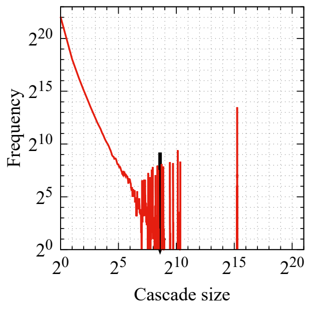
(b) soc-LiveJournal1 ($\text{UC}_{0.01}$)



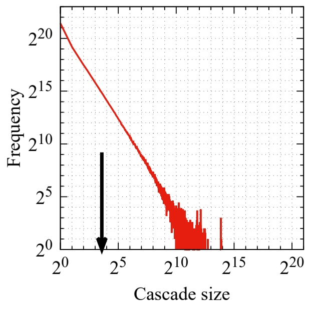
(c) soc-LiveJournal1 (TRI)



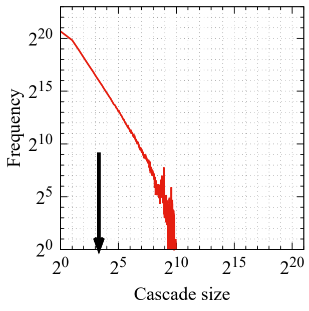
(d) soc-LiveJournal1 ($\text{EXP}_{0.1}$)



(e) soc-LiveJournal1 ($\text{EXP}_{0.01}$)

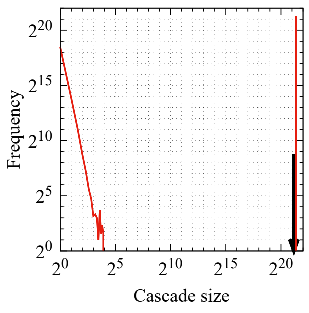


(f) soc-LiveJournal1 (IWC)

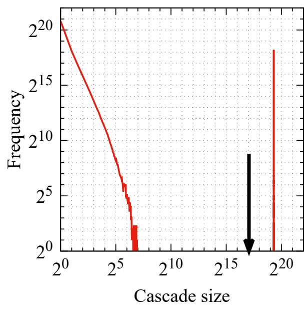


(g) soc-LiveJournal1 (OWC)

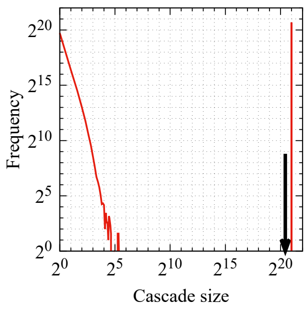
Figure A.17: Size distribution of reachable sets in soc-LiveJournal1 network.



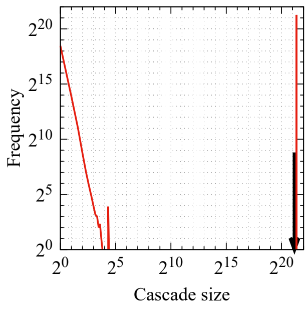
(a) com-Orkut ($\text{UC}_{0.1}$)



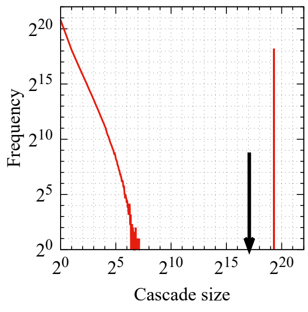
(b) com-Orkut ($\text{UC}_{0.01}$)



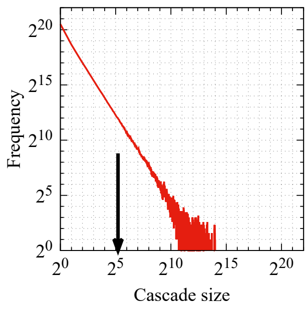
(c) com-Orkut (TRI)



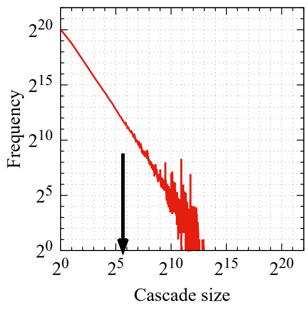
(d) com-Orkut ($\text{EXP}_{0.1}$)



(e) com-Orkut ($\text{EXP}_{0.01}$)

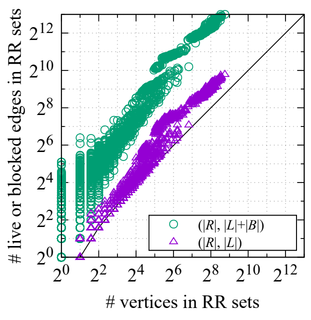


(f) com-Orkut (IWC)

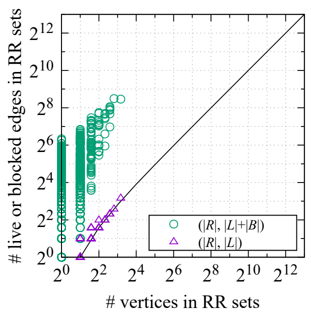


(g) com-Orkut (OWC)

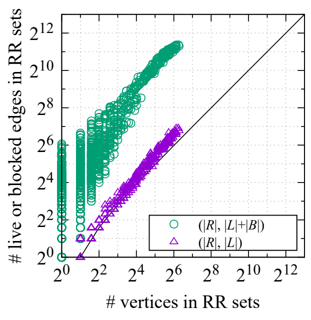
Figure A.18: Size distribution of reachable sets in com-Orkut network.



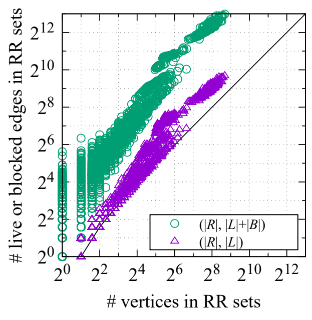
(a) ca-GrQc ($\text{UC}_{0.1}$)



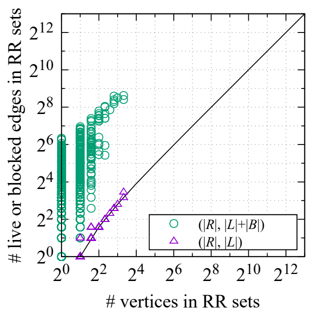
(b) ca-GrQc ($\text{UC}_{0.01}$)



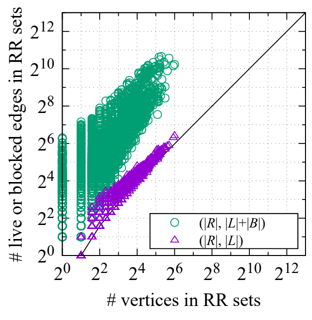
(c) ca-GrQc (TRI)



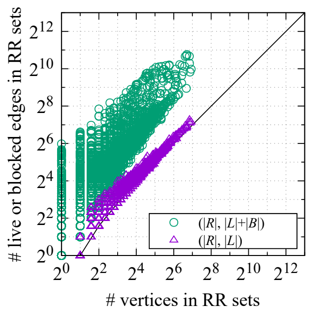
(d) ca-GrQc ($\text{EXP}_{0.1}$)



(e) ca-GrQc ($\text{EXP}_{0.01}$)

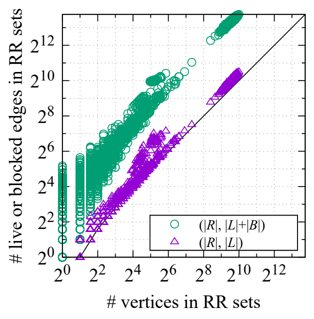


(f) ca-GrQc (IWC)

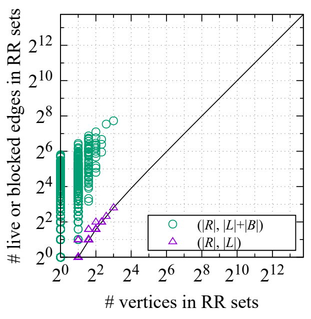


(g) ca-GrQc (OWC)

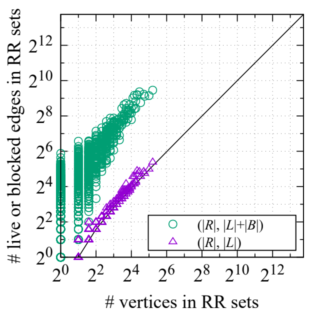
Figure A.19: Structures of RR sets in ca-GrQc network.



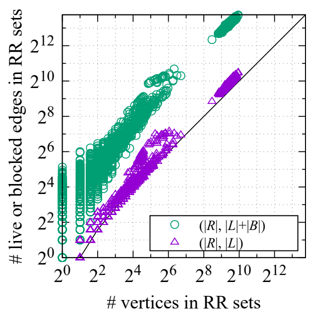
(a) ca-HepTh (UC$_{0.1}$)



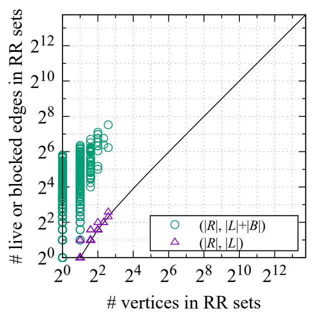
(b) ca-HepTh (UC$_{0.01}$)



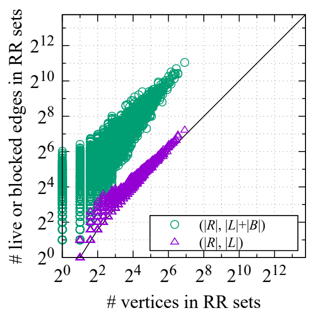
(c) ca-HepTh (TRI)



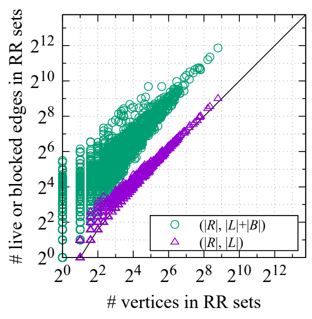
(d) ca-HepTh (EXP$_{0.1}$)



(e) ca-HepTh (EXP$_{0.01}$)

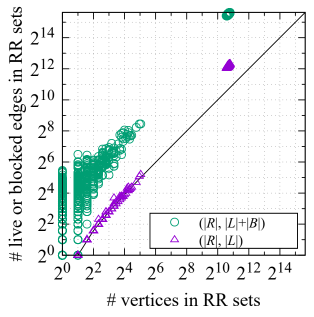


(f) ca-HepTh (IWC)

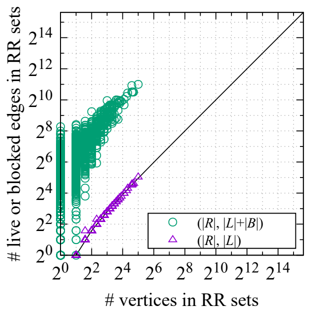


(g) ca-HepTh (OWC)

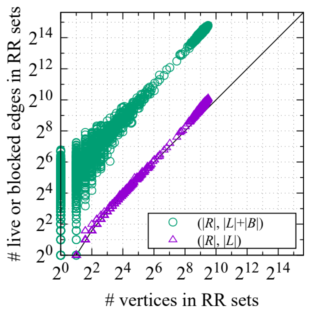
Figure A.20: Structures of RR sets in ca-HepTh network.



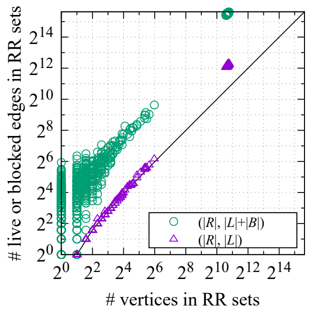
(a) wiki-Vote ($\text{UC}_{0.1}$)



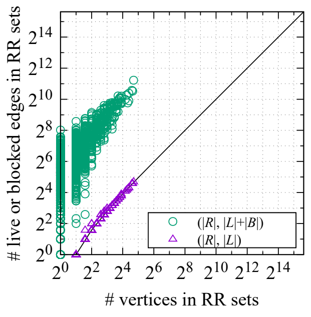
(b) wiki-Vote ($\text{UC}_{0.01}$)



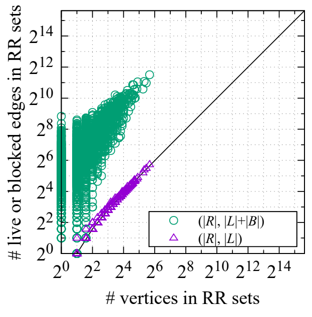
(c) wiki-Vote (TRI)



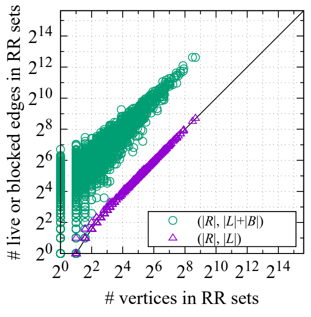
(d) wiki-Vote ($\text{EXP}_{0.1}$)



(e) wiki-Vote ($\text{EXP}_{0.01}$)

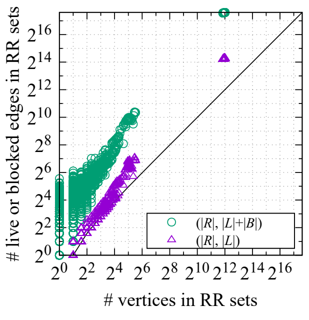


(f) wiki-Vote (IWC)

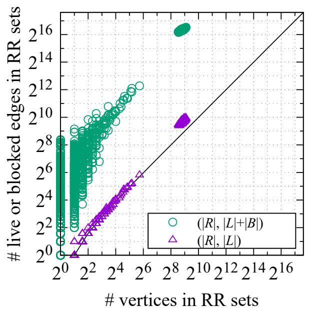


(g) wiki-Vote (OWC)

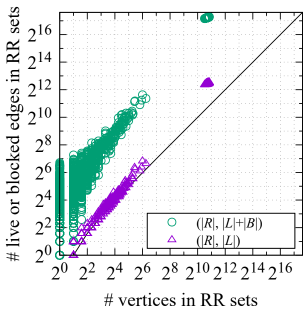
Figure A.21: Structures of RR sets in wiki-Vote network.



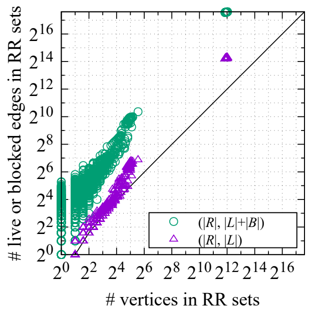
(a) ca-HepPh (UC$_{0.1}$)



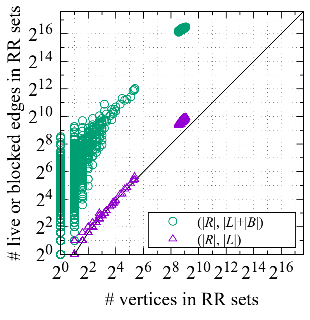
(b) ca-HepPh (UC$_{0.01}$)



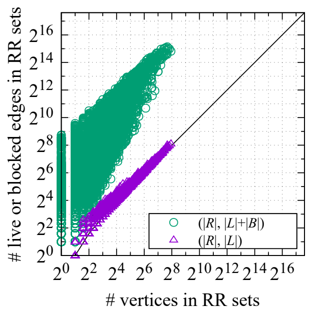
(c) ca-HepPh (TRI)



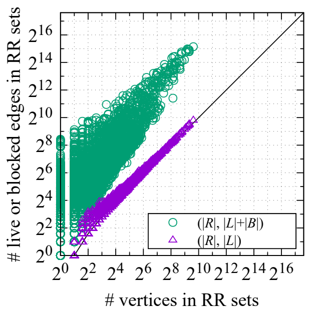
(d) ca-HepPh (EXP$_{0.1}$)



(e) ca-HepPh (EXP$_{0.01}$)

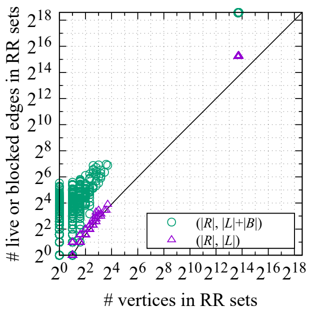


(f) ca-HepPh (IWC)

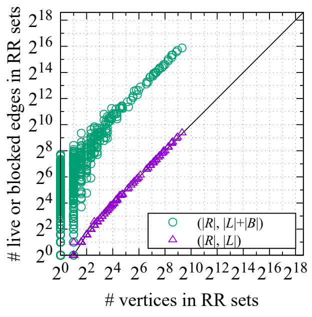


(g) ca-HepPh (OWC)

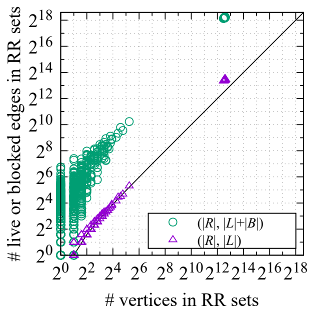
Figure A.22: Structures of RR sets in ca-HepPh network.



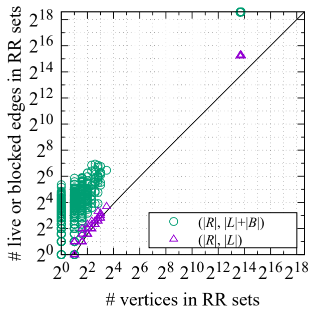
(a) soc-Epinions1 (UC$_{0.1}$)



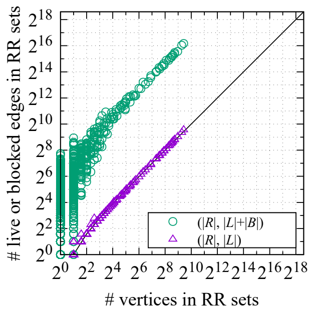
(b) soc-Epinions1 (UC$_{0.01}$)



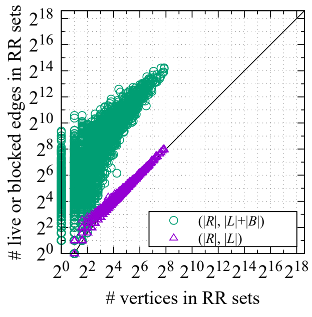
(c) soc-Epinions1 (TRI)



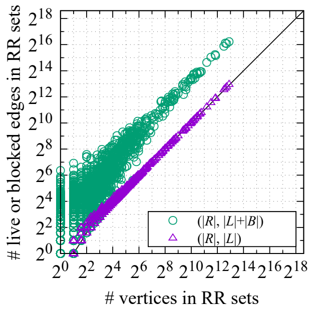
(d) soc-Epinions1 (EXP$_{0.1}$)



(e) soc-Epinions1 (EXP$_{0.01}$)

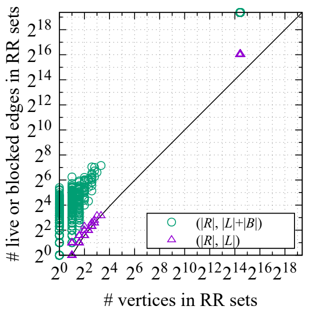


(f) soc-Epinions1 (IWC)

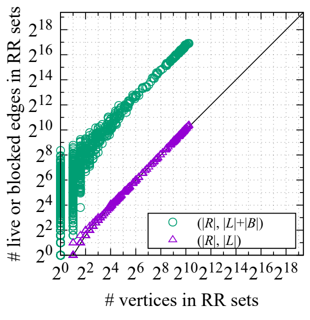


(g) soc-Epinions1 (OWC)

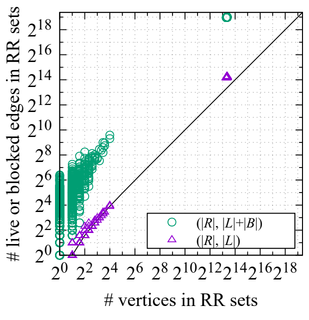
Figure A.23: Structures of RR sets in soc-Epinions1 network.



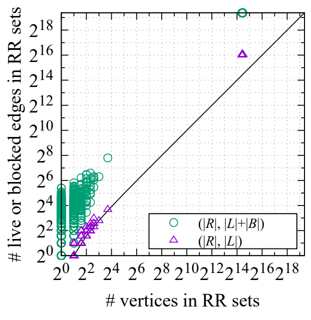
(a) soc-Slashdot0922 ($\textsc{uc}_{0.1}$)



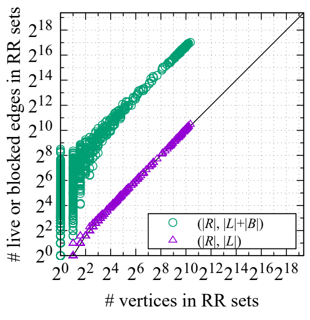
(b) soc-Slashdot0922 ($\textsc{uc}_{0.01}$)



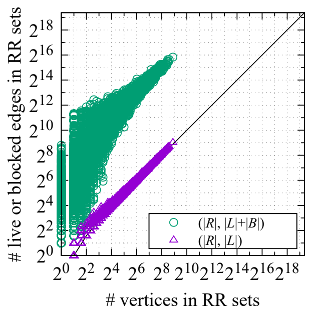
(c) soc-Slashdot0922 ($\textsc{tri}$)



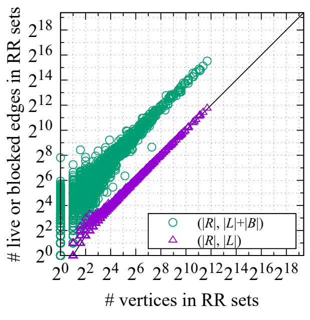
(d) soc-Slashdot0922 ($\textsc{exp}_{0.1}$)



(e) soc-Slashdot0922 ($\textsc{exp}_{0.01}$)

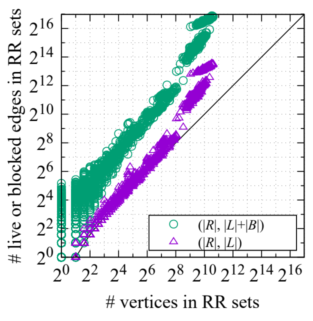


(f) soc-Slashdot0922 ($\textsc{iwc}$)

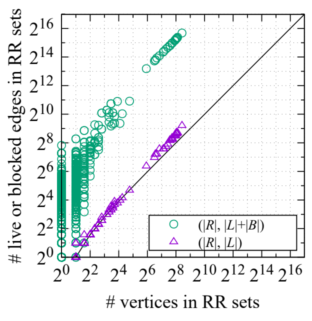


(g) soc-Slashdot0922 ($\textsc{owc}$)

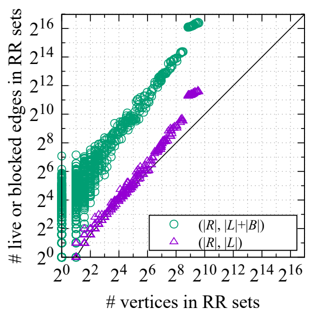
Figure A.24: Structures of RR sets in soc-Slashdot0922 network.



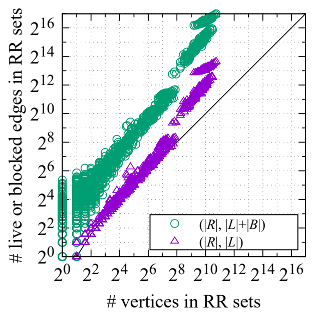
(a) web-NotreDame ($\text{UC}_{0.1}$)



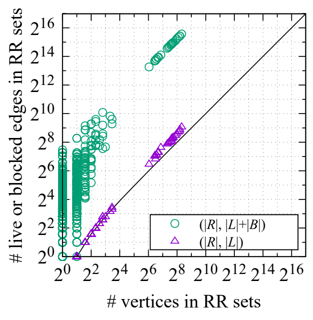
(b) web-NotreDame ($\text{UC}_{0.01}$)



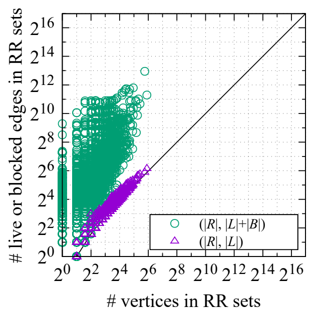
(c) web-NotreDame (TRI)



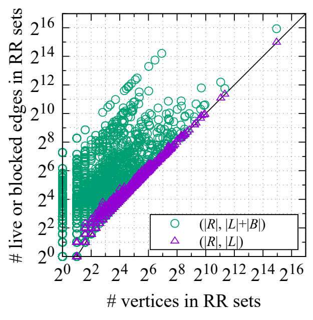
(d) web-NotreDame ($\text{EXP}_{0.1}$)



(e) web-NotreDame ($\text{EXP}_{0.01}$)

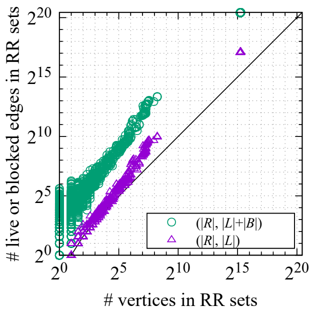


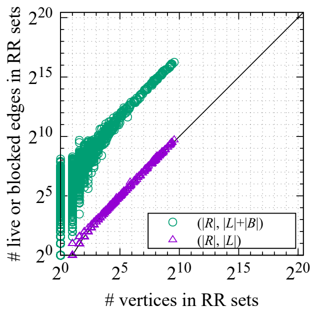
(f) web-NotreDame (IWC)

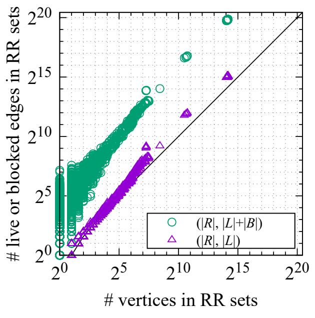


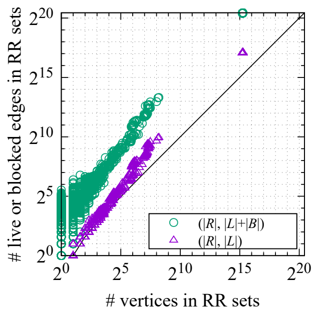
(g) web-NotreDame (OWC)

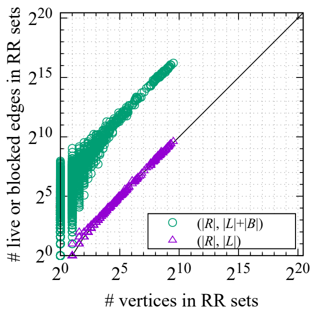
Figure A.25: Structures of RR sets in web-NotreDame network.

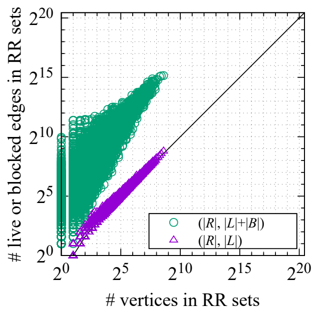

(a) ego-Twitter ($\text{UC}_{0.1}$)


(b) ego-Twitter ($\text{UC}_{0.01}$)


(c) ego-Twitter (TRI)


(d) ego-Twitter ($\text{EXP}_{0.1}$)


(e) ego-Twitter ($\text{EXP}_{0.01}$)


(f) ego-Twitter (IWC)

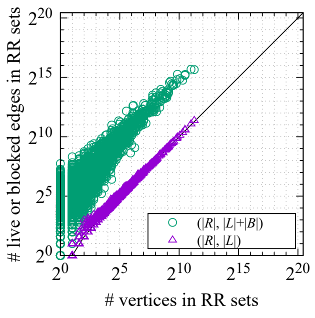
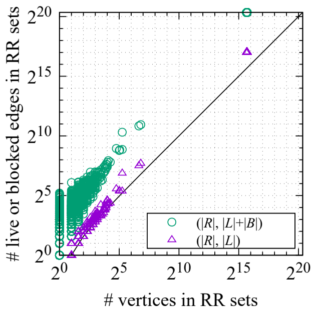
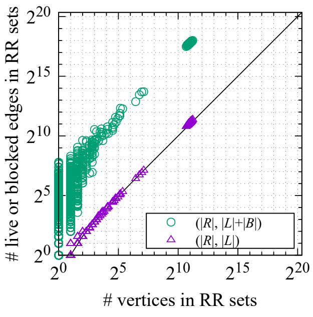

(g) ego-Twitter (OWC)

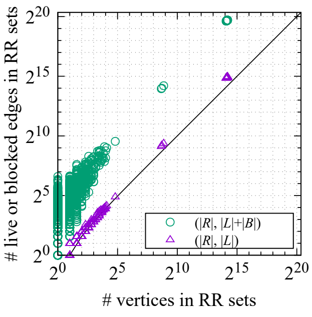
Figure A.26: Structures of RR sets in ego-Twitter network.



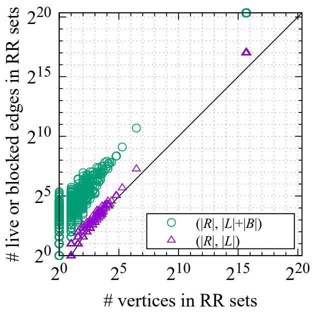
(a) loc-Gowalla ($\text{UC}_{0.1}$)



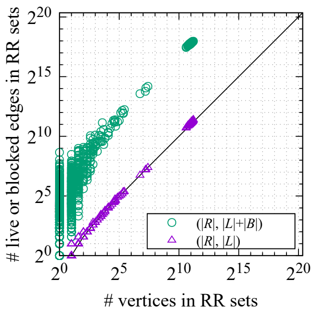
(b) loc-Gowalla ($\text{UC}_{0.01}$)



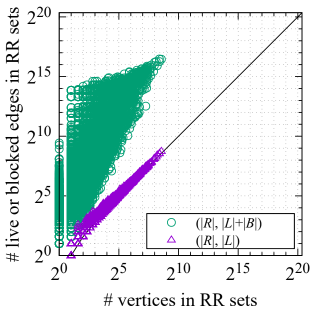
(c) loc-Gowalla (TRI)



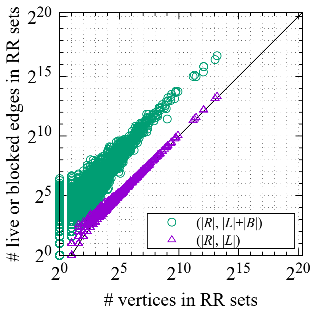
(d) loc-Gowalla ($\text{EXP}_{0.1}$)



(e) loc-Gowalla ($\text{EXP}_{0.01}$)

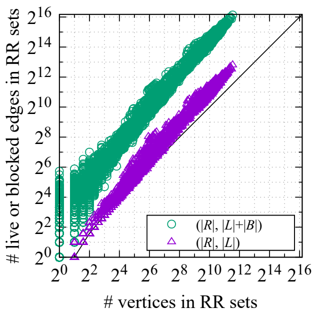


(f) loc-Gowalla (IWC)

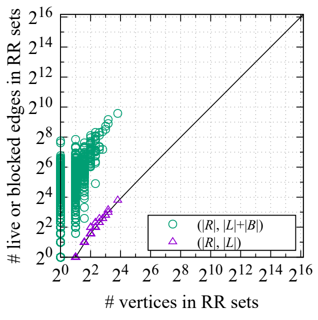


(g) loc-Gowalla (OWC)

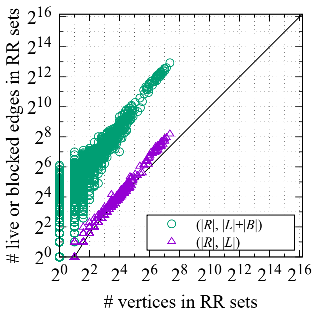
Figure A.27: Structures of RR sets in loc-Gowalla network.



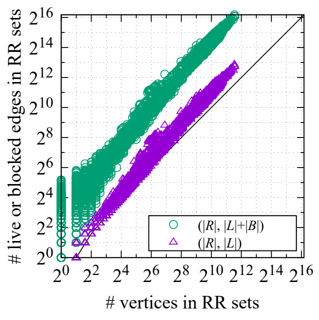
(a) web-Stanford (UC$_{0.1}$)



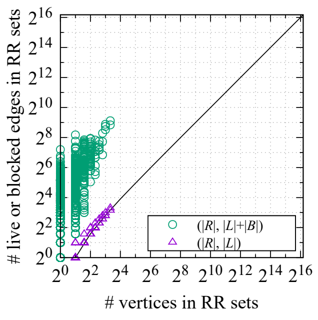
(b) web-Stanford (UC$_{0.01}$)



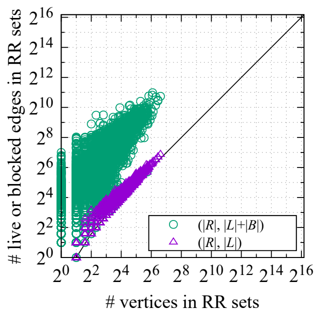
(c) web-Stanford (TRI)



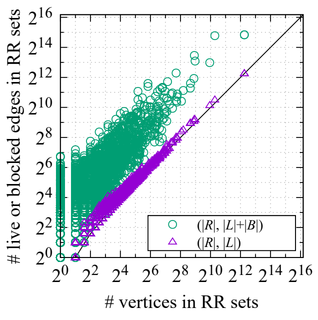
(d) web-Stanford (EXP$_{0.1}$)



(e) web-Stanford (EXP$_{0.01}$)

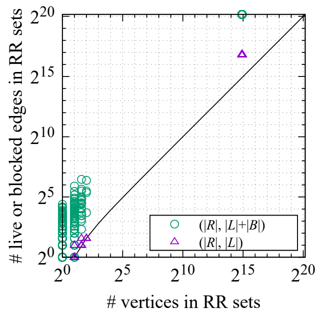


(f) web-Stanford (IWC)

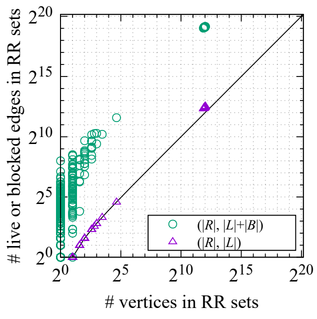


(g) web-Stanford (OWC)

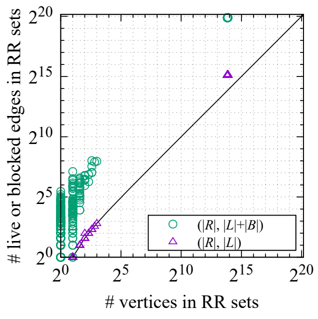
Figure A.28: Structures of RR sets in web-Stanford network.



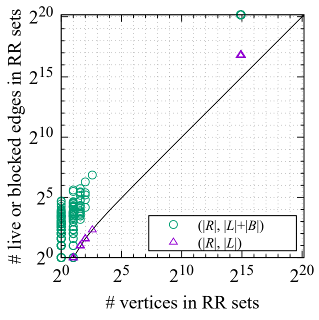
(a) wiki-Talk ($\text{UC}_{0.1}$)



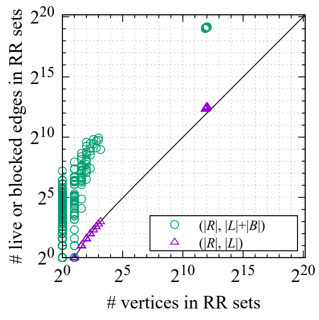
(b) wiki-Talk ($\text{UC}_{0.01}$)



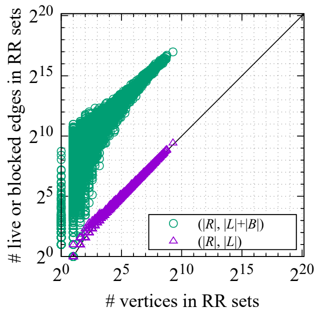
(c) wiki-Talk (TRI)



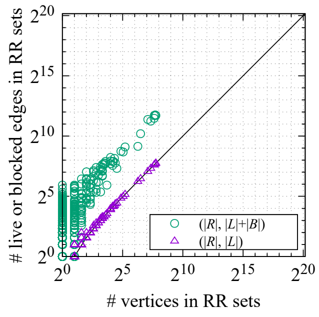
(d) wiki-Talk ($\text{EXP}_{0.1}$)



(e) wiki-Talk ($\text{EXP}_{0.01}$)

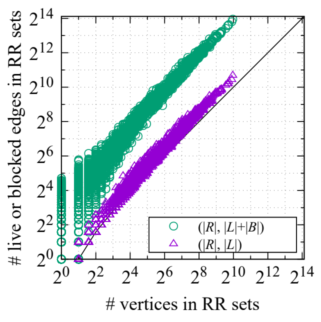


(f) wiki-Talk (IWC)

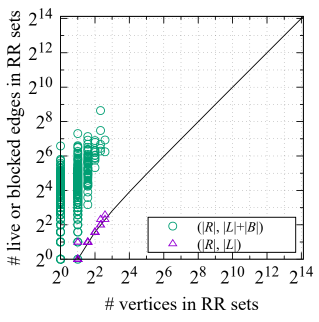


(g) wiki-Talk (OWC)

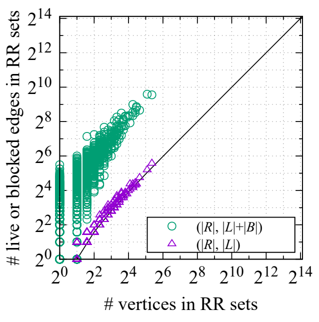
Figure A.29: Structures of RR sets in wiki-Talk network.



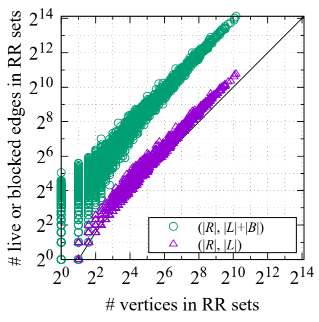
(a) web-Google (UC$_{0.1}$)



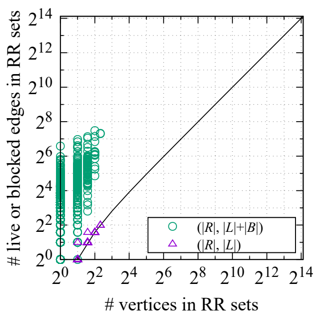
(b) web-Google (UC$_{0.01}$)



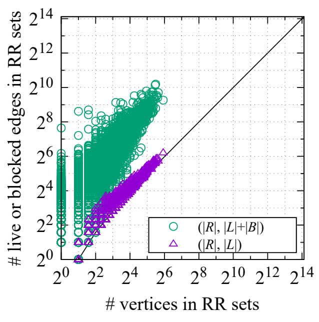
(c) web-Google (TRI)



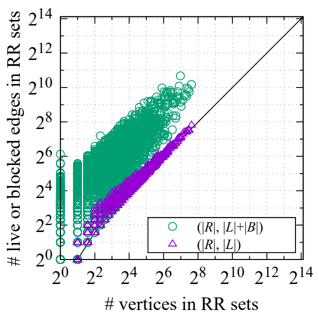
(d) web-Google (EXP$_{0.1}$)



(e) web-Google (EXP$_{0.01}$)

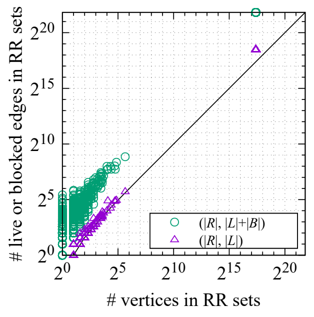


(f) web-Google (IWC)

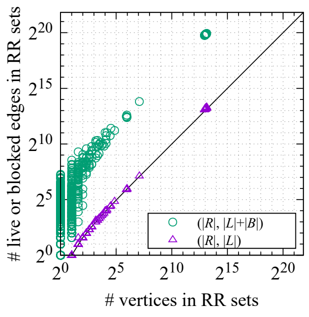


(g) web-Google (OWC)

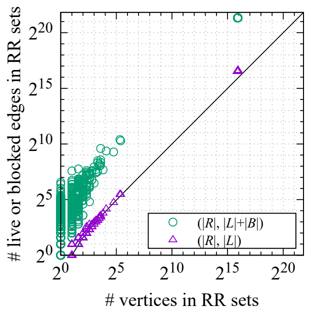
Figure A.30: Structures of RR sets in web-Google network.



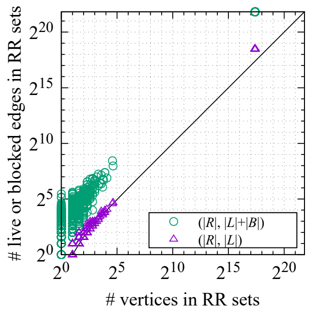
(a) com-Youtube ($\text{UC}_{0.1}$)



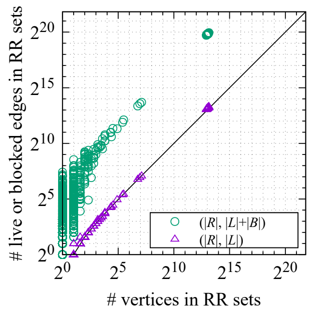
(b) com-Youtube ($\text{UC}_{0.01}$)



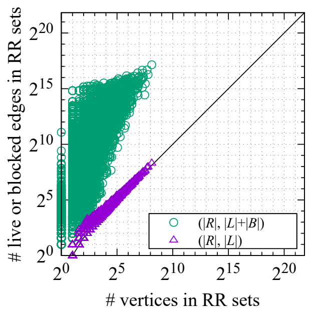
(c) com-Youtube (TRI)



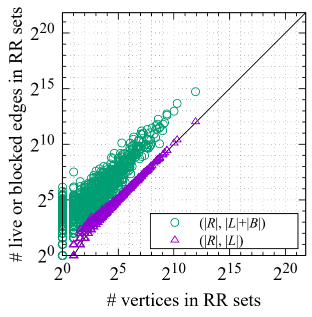
(d) com-Youtube ($\text{EXP}_{0.1}$)



(e) com-Youtube ($\text{EXP}_{0.01}$)

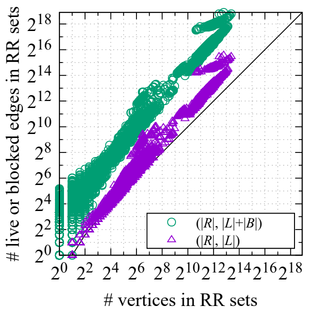


(f) com-Youtube (IWC)

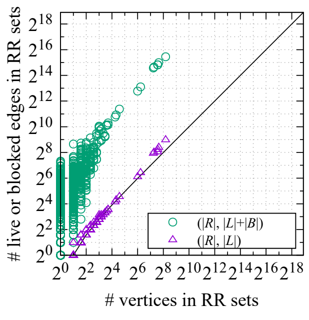


(g) com-Youtube (OWC)

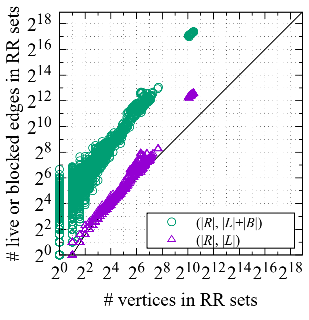
Figure A.31: Structures of RR sets in com-Youtube network.



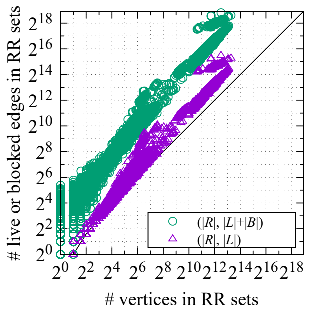
(a) web-BerkStan (UC$_{0.1}$)



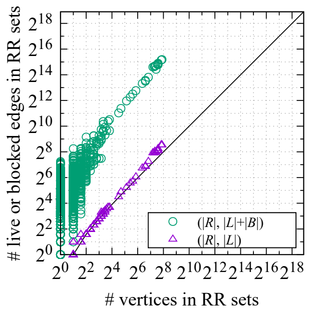
(b) web-BerkStan (UC$_{0.01}$)



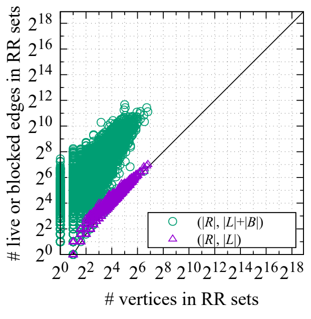
(c) web-BerkStan (TRI)



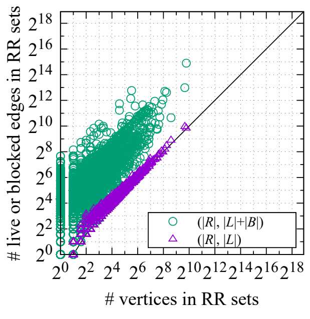
(d) web-BerkStan (EXP$_{0.1}$)



(e) web-BerkStan (EXP$_{0.01}$)

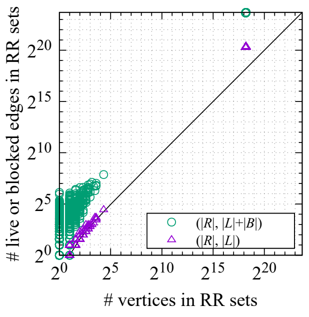


(f) web-BerkStan (IWC)

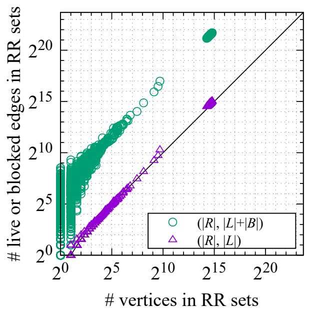


(g) web-BerkStan (OWC)

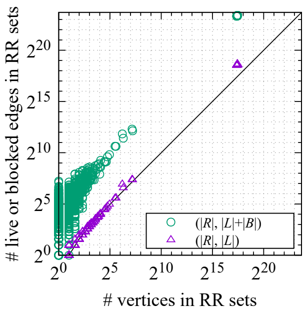
Figure A.32: Structures of RR sets in web-BerkStan network.



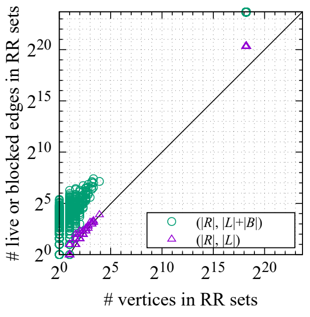
(a) higgs-twitter (UC$_{0.1}$)



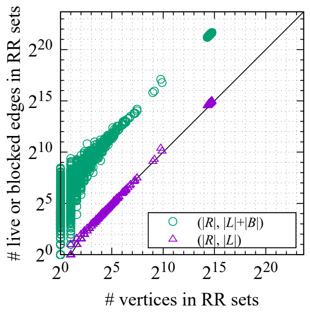
(b) higgs-twitter (UC$_{0.01}$)



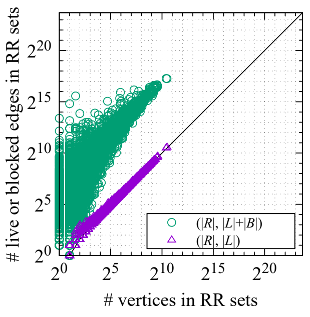
(c) higgs-twitter (TRI)



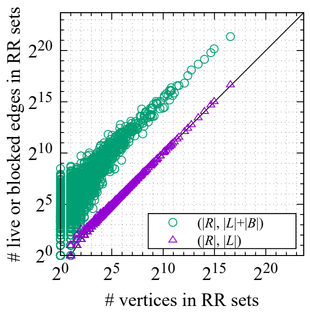
(d) higgs-twitter (EXP$_{0.1}$)



(e) higgs-twitter (EXP$_{0.01}$)

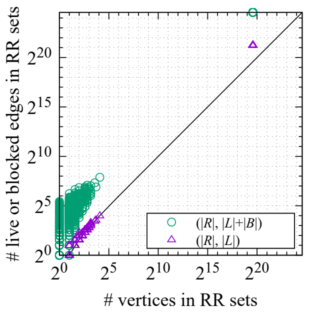


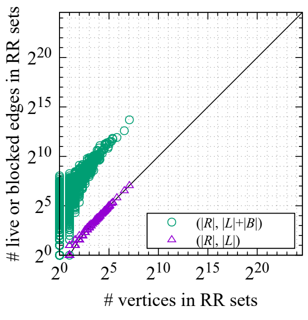
(f) higgs-twitter (IWC)

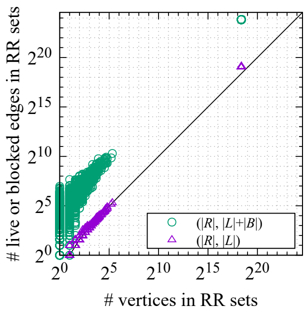


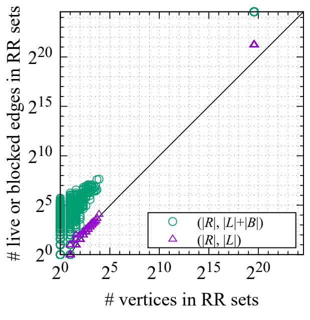
(g) higgs-twitter (OWC)

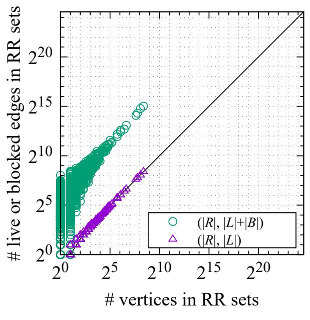
Figure A.33: Structures of RR sets in higgs-twitter network.

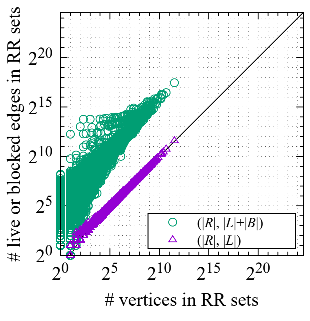

(a) soc-Pokec ($\mathrm{UC}_{0.1}$)


(b) soc-Pokec ($\mathrm{UC}_{0.01}$)


(c) soc-Pokec (TRI)


(d) soc-Pokec ($\mathrm{EXP}_{0.1}$)


(e) soc-Pokec ($\mathrm{EXP}_{0.01}$)


(f) soc-Pokec (IWC)

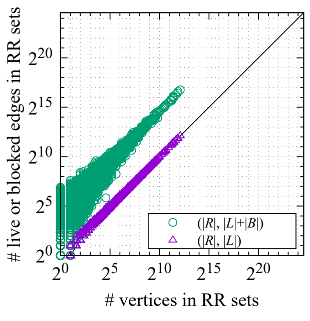
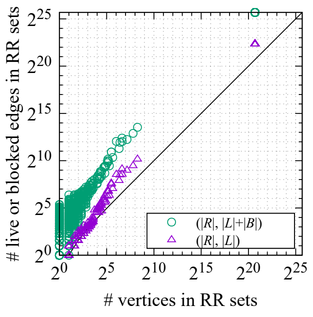
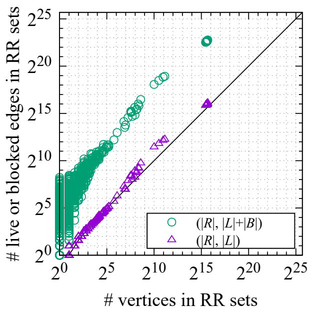

(g) soc-Pokec (OWC)

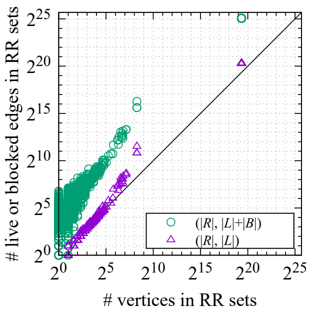
Figure A.34: Structures of RR sets in soc-Pokec network.



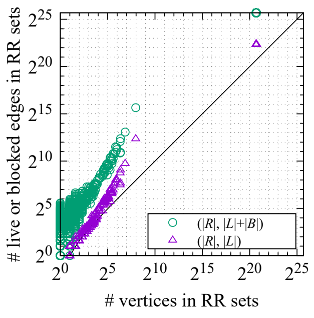
(a) soc-LiveJournal1 ($\textsc{uc}_{0.1}$)



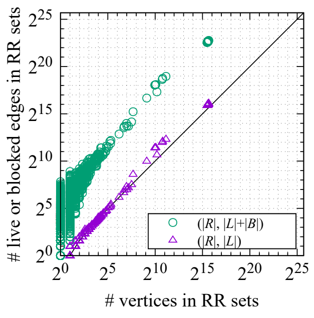
(b) soc-LiveJournal1 ($\textsc{uc}_{0.01}$)



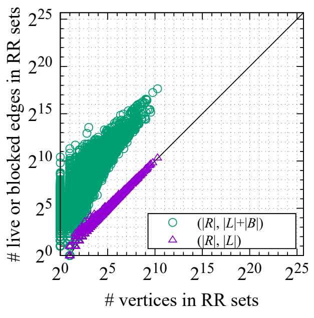
(c) soc-LiveJournal1 ($\textsc{tri}$)



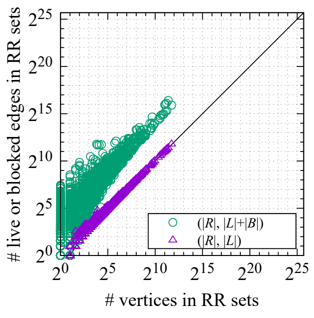
(d) soc-LiveJournal1 ($\textsc{exp}_{0.1}$)



(e) soc-LiveJournal1 ($\textsc{exp}_{0.01}$)

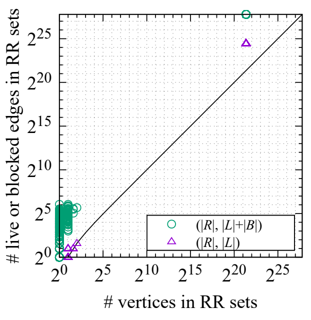


(f) soc-LiveJournal1 ($\textsc{iwc}$)

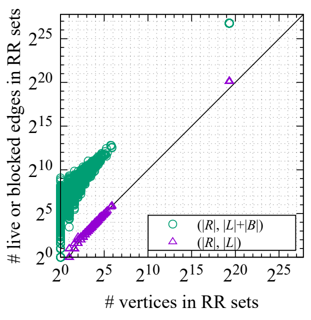


(g) soc-LiveJournal1 ($\textsc{owc}$)

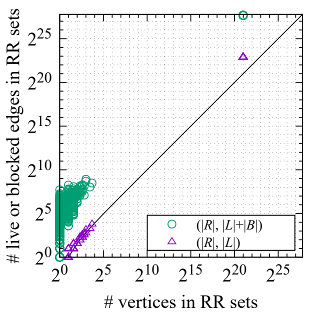
Figure A.35: Structures of RR sets in soc-LiveJournal1 network.



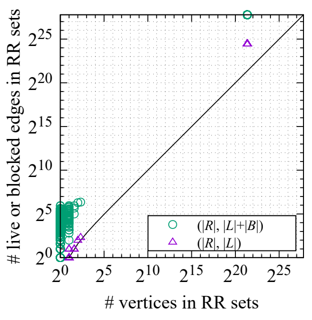
(a) com-Orkut (UC$_{0.1}$)



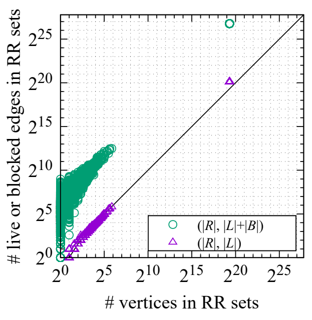
(b) com-Orkut (UC$_{0.01}$)



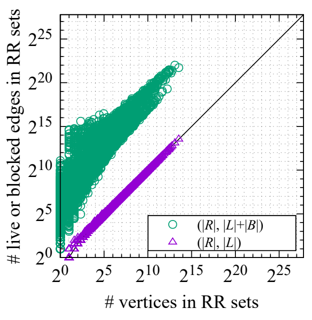
(c) com-Orkut (TRI)



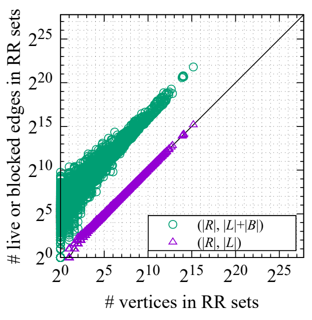
(d) com-Orkut (EXP$_{0.1}$)



(e) com-Orkut (EXP$_{0.01}$)



(f) com-Orkut (IWC)



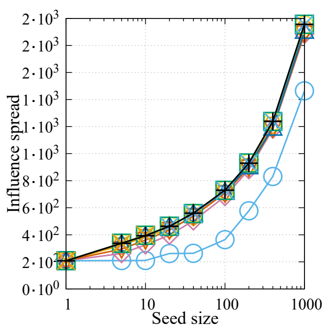
(g) com-Orkut (OWC)

Figure A.36: Structures of RR sets in com-Orkut network.

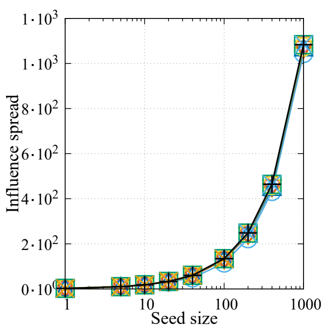
Appendix B

Additional Experimental Results in Chapter 5

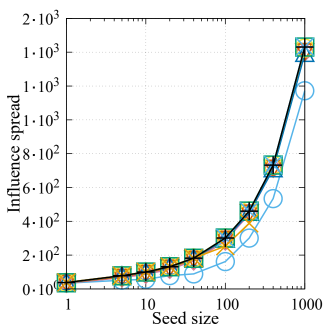
We here provide complete experimental results performed in Chapter 5. Figures B.1–B.18 show the influence spread of the solution that each algorithm produced for each influence probability setting. Figures B.19–B.36 show the running time of each algorithm each influence probability setting.



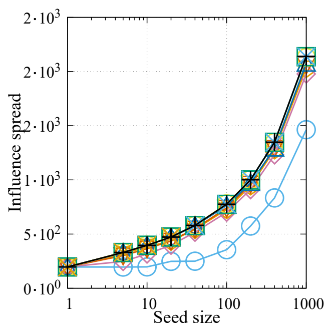
(a) ca-GrQc ($\text{UC}_{0.1}$)



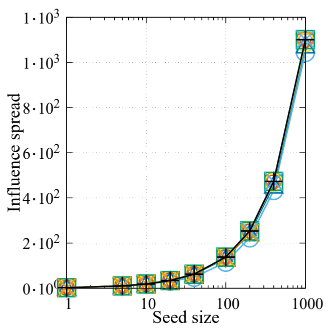
(b) ca-GrQc ($\text{UC}_{0.01}$)



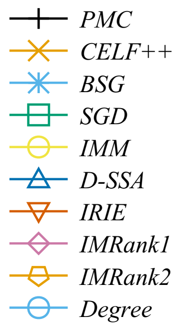
(c) ca-GrQc (TRI)

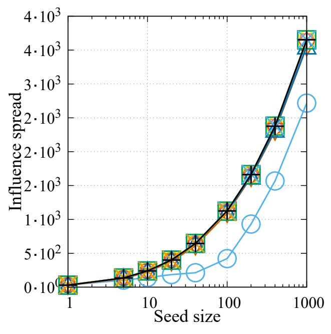


(d) ca-GrQc ($\text{EXP}_{0.1}$)

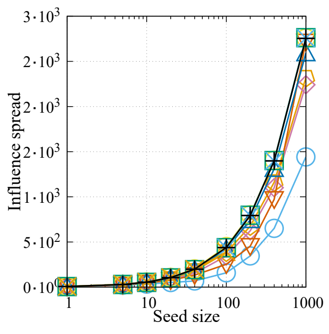


(e) ca-GrQc ($\text{EXP}_{0.01}$)



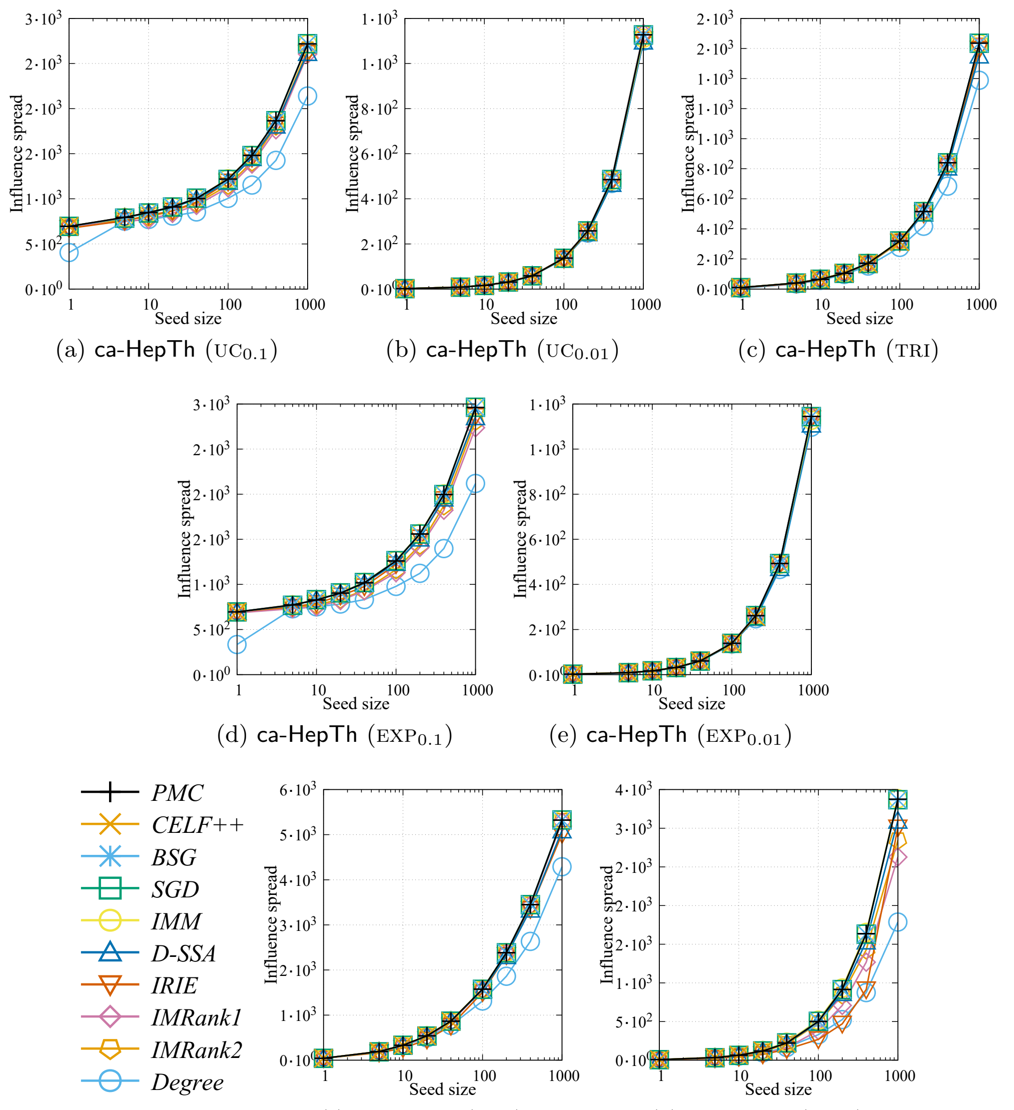


(f) ca-GrQc (IWC)



(g) ca-GrQc (OWC)

Figure B.1: Influence spread of each algorithm for ca-GrQc network.

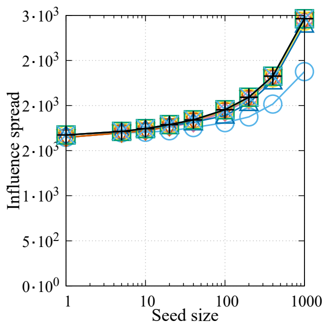


(a) ca-HepTh (UC$_{0.1}$) (b) ca-HepTh (UC$_{0.01}$) (c) ca-HepTh (TRI)

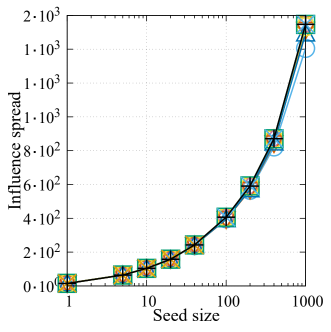
(d) ca-HepTh (EXP$_{0.1}$) (e) ca-HepTh (EXP$_{0.01}$)

(f) ca-HepTh (IWC) (g) ca-HepTh (OWC)

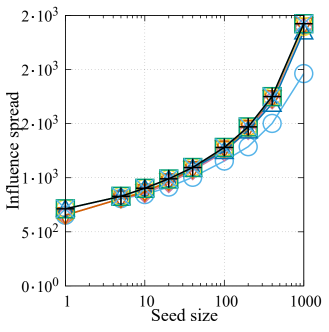
Figure B.2: Influence spread of each algorithm for ca-HepTh network.



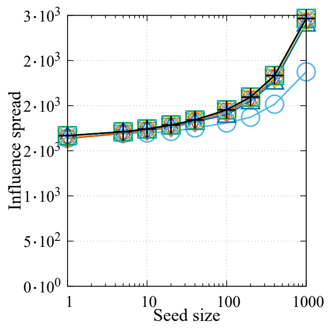
(a) wiki-Vote (UC$_{0.1}$)



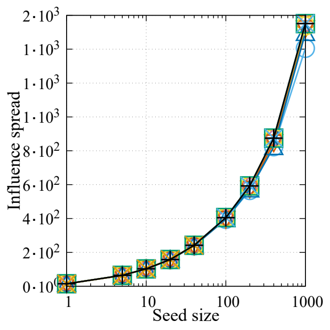
(b) wiki-Vote (UC$_{0.01}$)



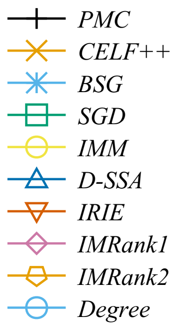
(c) wiki-Vote (TRI)

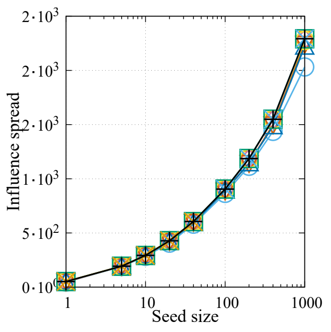


(d) wiki-Vote (EXP$_{0.1}$)

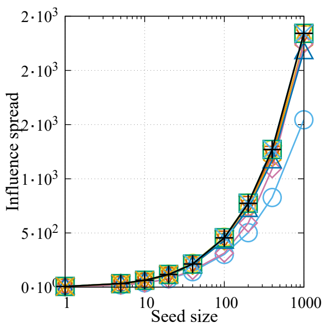


(e) wiki-Vote (EXP$_{0.01}$)



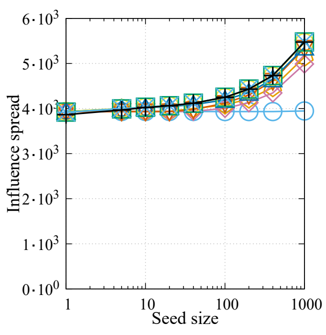


(f) wiki-Vote (IWC)

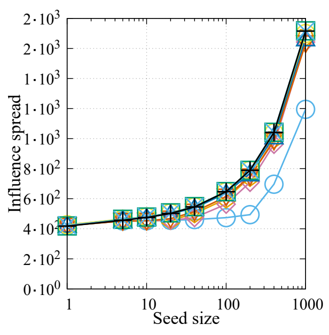


(g) wiki-Vote (OWC)

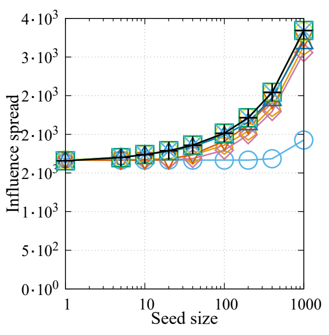
Figure B.3: Influence spread of each algorithm for wiki-Vote network.



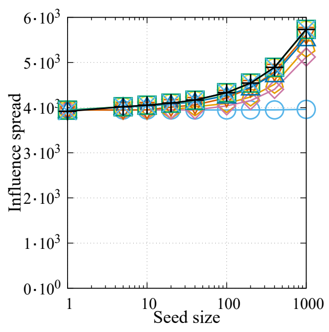
(a) ca-HepPh ($\text{UC}_{0.1}$)



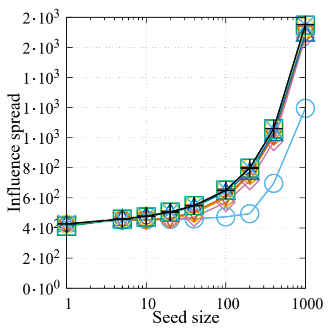
(b) ca-HepPh ($\text{UC}_{0.01}$)



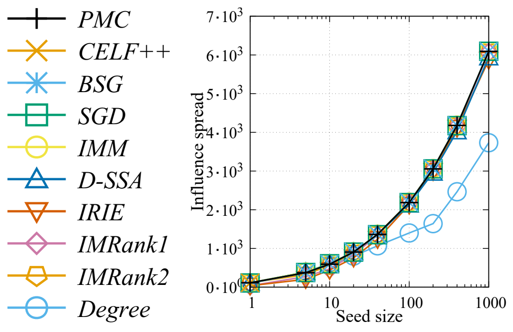
(c) ca-HepPh (TRI)



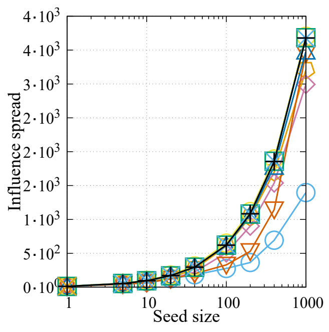
(d) ca-HepPh ($\text{EXP}_{0.1}$)



(e) ca-HepPh ($\text{EXP}_{0.01}$)

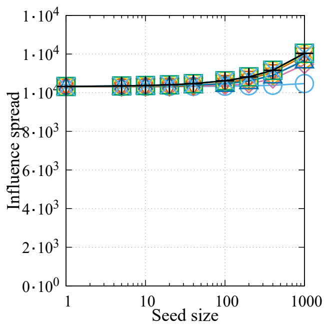


(f) ca-HepPh (IWC)

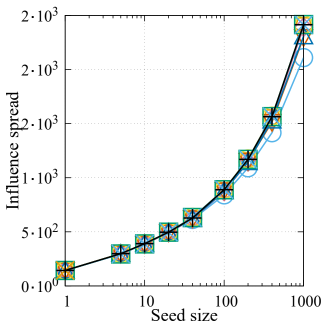


(g) ca-HepPh (OWC)

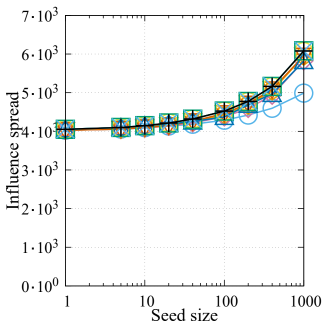
Figure B.4: Influence spread of each algorithm for ca-HepPh network.



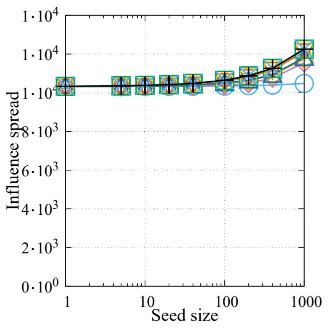
(a) soc-Epinions1 ($\textsc{uc}_{0.1}$)



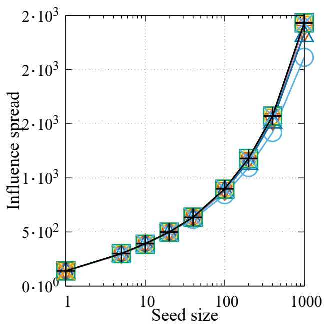
(b) soc-Epinions1 ($\textsc{uc}_{0.01}$)



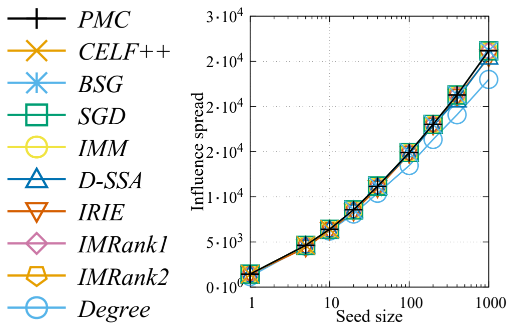
(c) soc-Epinions1 ($\textsc{tri}$)



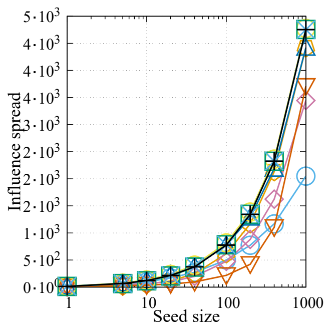
(d) soc-Epinions1 ($\textsc{exp}_{0.1}$)



(e) soc-Epinions1 ($\textsc{exp}_{0.01}$)

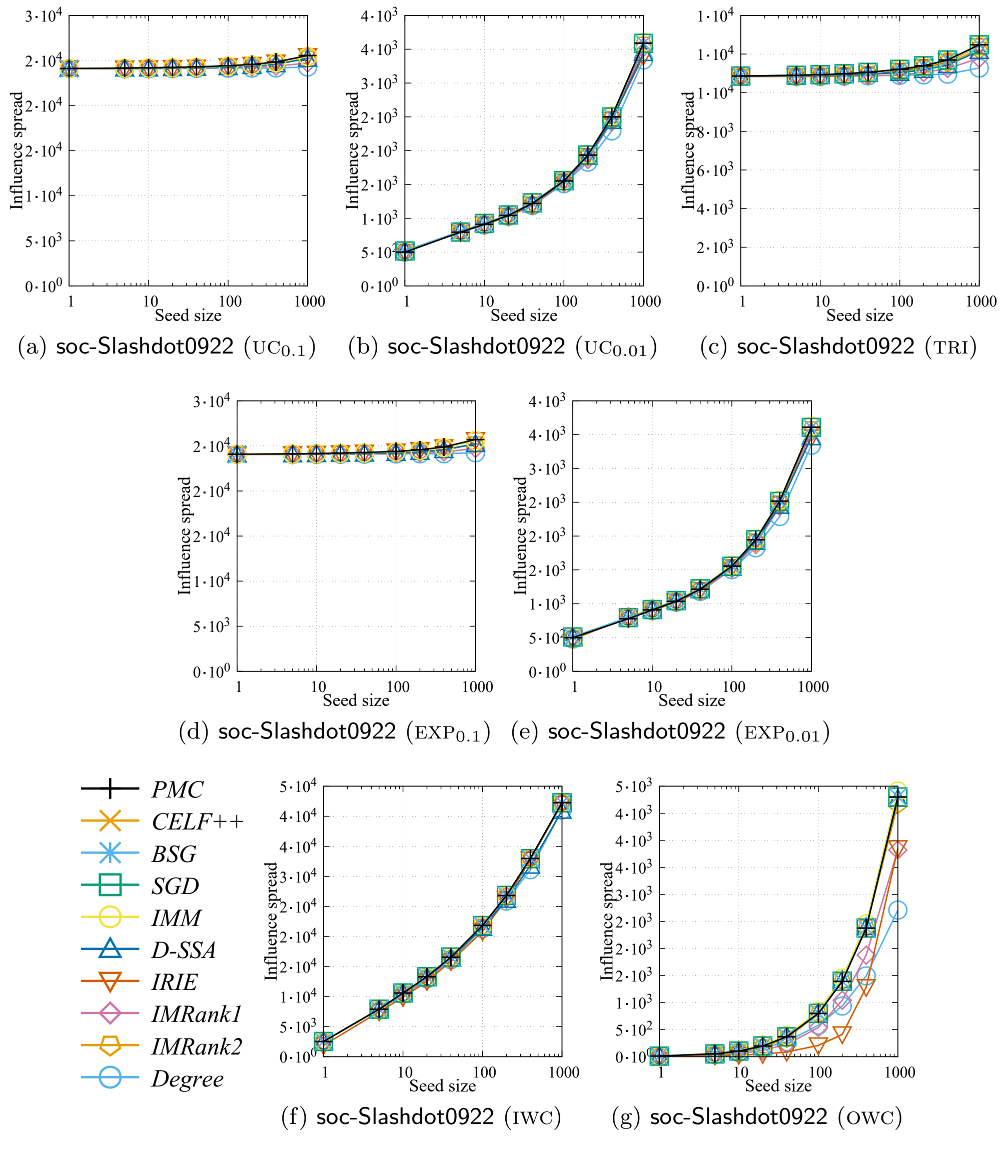


(f) soc-Epinions1 ($\textsc{iwc}$)



(g) soc-Epinions1 ($\textsc{owc}$)

Figure B.5: Influence spread of each algorithm for soc-Epinions1 network.

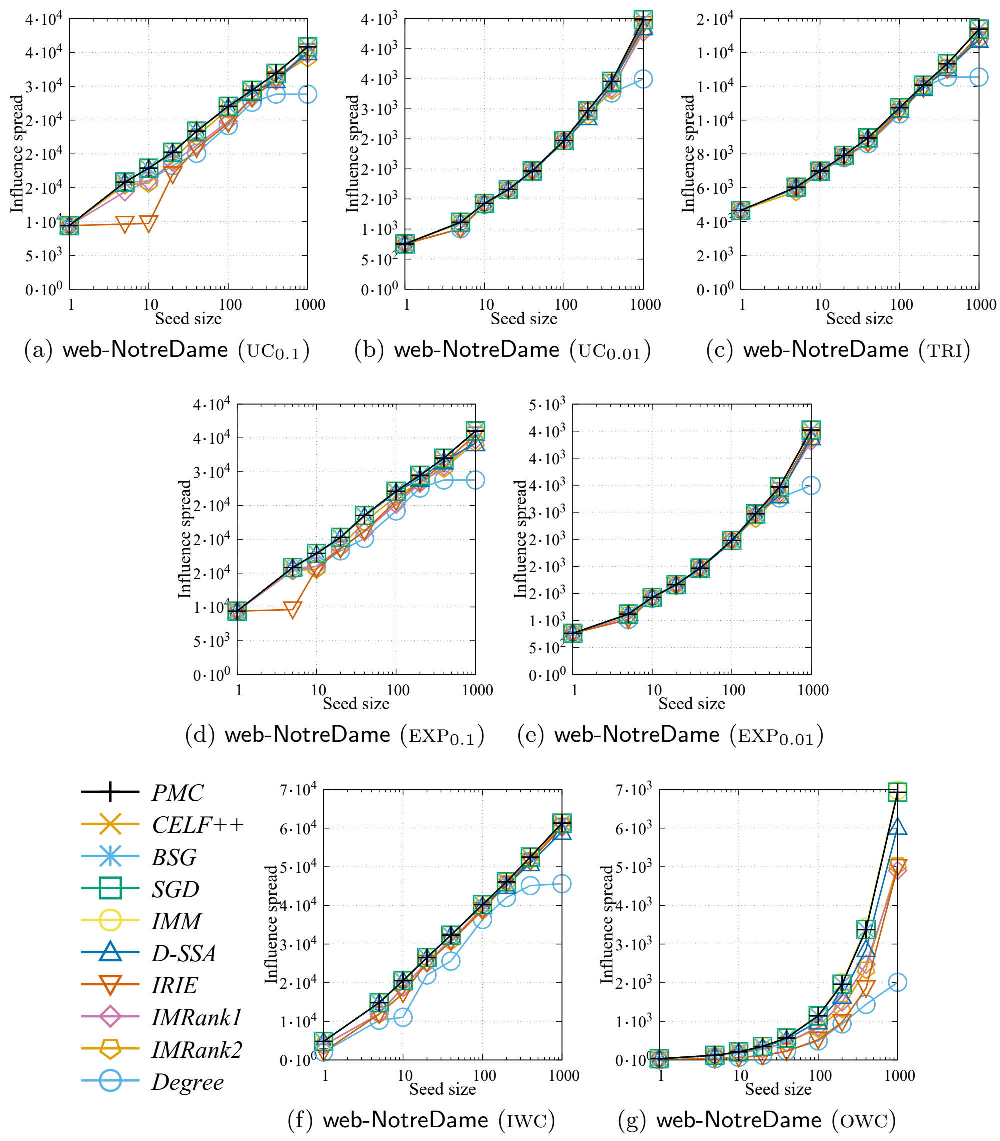


(a) soc-Slashdot0922 ($\text{UC}_{0.1}$) (b) soc-Slashdot0922 ($\text{UC}_{0.01}$) (c) soc-Slashdot0922 (TRI)

(d) soc-Slashdot0922 ($\text{EXP}_{0.1}$) (e) soc-Slashdot0922 ($\text{EXP}_{0.01}$)

(f) soc-Slashdot0922 (IWC) (g) soc-Slashdot0922 (OWC)

Figure B.6: Influence spread of each algorithm for soc-Slashdot0922 network.

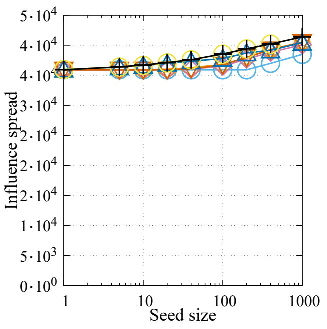


(a) web-NotreDame ($\mathrm{UC}_{0.1}$) (b) web-NotreDame ($\mathrm{UC}_{0.01}$) (c) web-NotreDame (TRI)

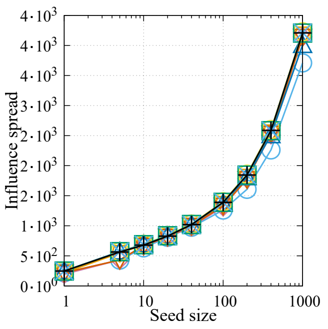
(d) web-NotreDame ($\mathrm{EXP}_{0.1}$) (e) web-NotreDame ($\mathrm{EXP}_{0.01}$)

(f) web-NotreDame (IWC) (g) web-NotreDame (OWC)

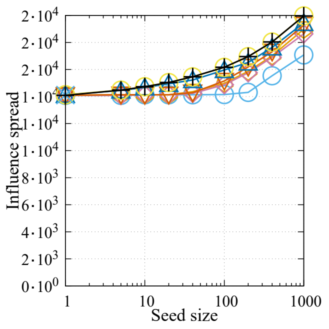
Figure B.7: Influence spread of each algorithm for web-NotreDame network.



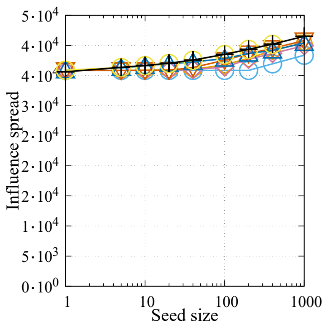
(a) ego-Twitter (UC$_{0.1}$)



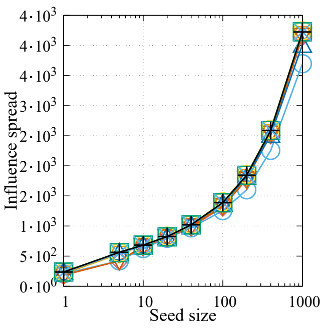
(b) ego-Twitter (UC$_{0.01}$)



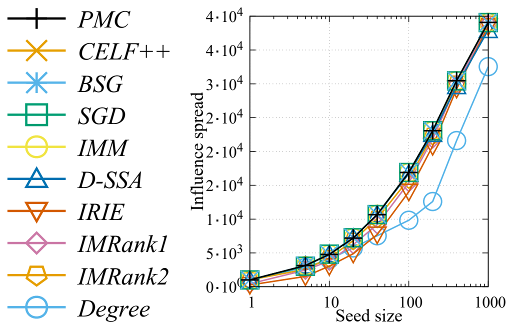
(c) ego-Twitter (TRI)



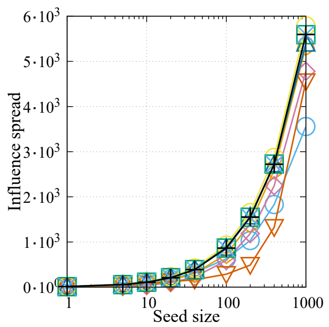
(d) ego-Twitter (EXP$_{0.1}$)



(e) ego-Twitter (EXP$_{0.01}$)

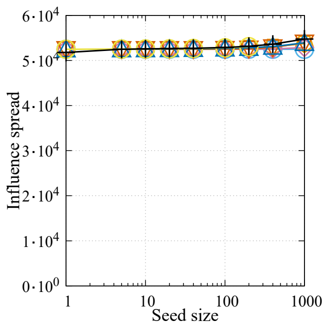


(f) ego-Twitter (IWC)

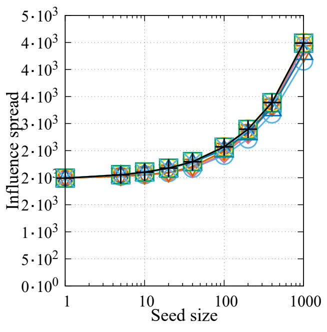


(g) ego-Twitter (OWC)

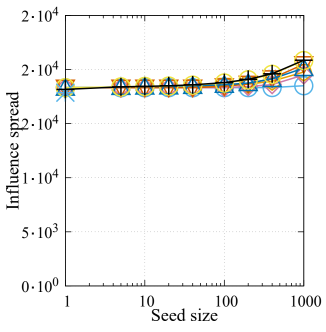
Figure B.8: Influence spread of each algorithm for ego-Twitter network.



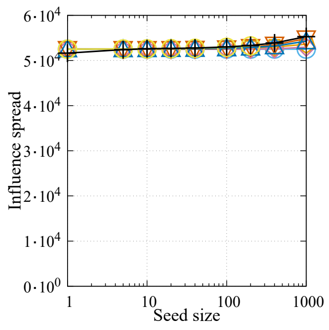
(a) loc-Gowalla ($\textsc{uc}_{0.1}$)



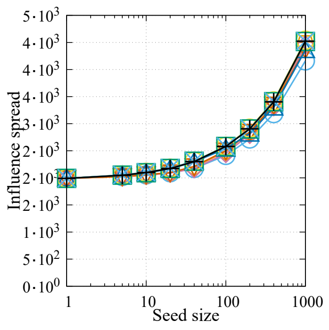
(b) loc-Gowalla ($\textsc{uc}_{0.01}$)



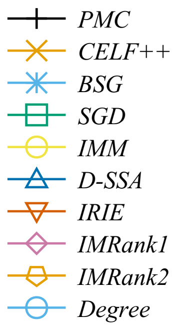
(c) loc-Gowalla ($\textsc{tri}$)

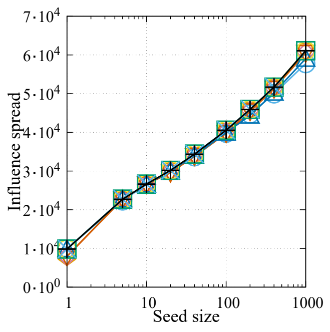


(d) loc-Gowalla ($\textsc{exp}_{0.1}$)

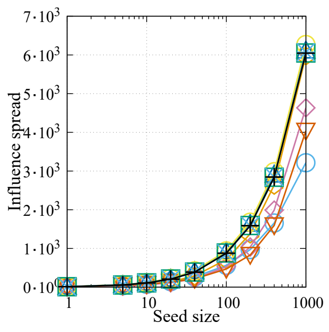


(e) loc-Gowalla ($\textsc{exp}_{0.01}$)



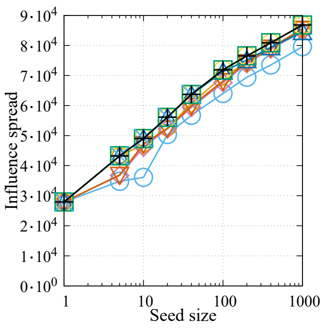


(f) loc-Gowalla ($\textsc{iwc}$)

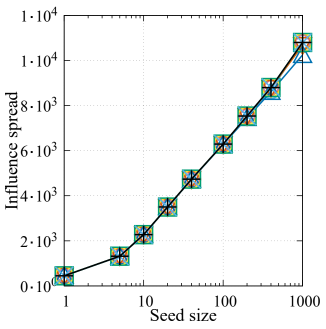


(g) loc-Gowalla ($\textsc{owc}$)

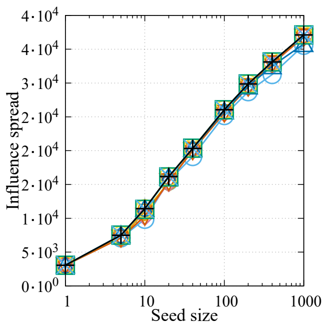
Figure B.9: Influence spread of each algorithm for loc-Gowalla network.



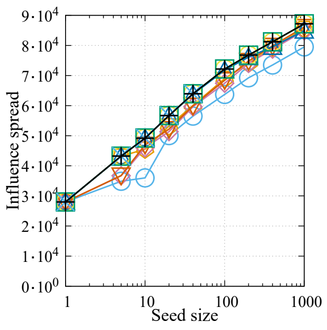
(a) web-Stanford (UC$_{0.1}$)



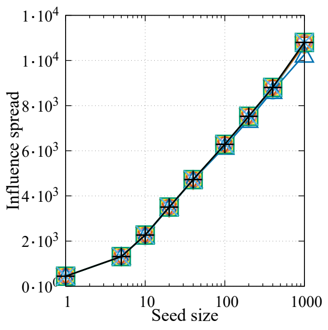
(b) web-Stanford (UC$_{0.01}$)



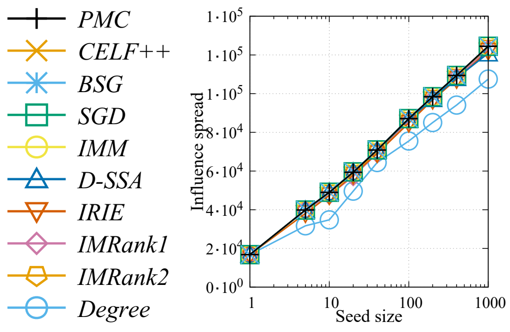
(c) web-Stanford (TRI)



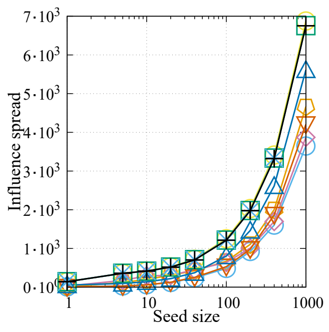
(d) web-Stanford (EXP$_{0.1}$)



(e) web-Stanford (EXP$_{0.01}$)

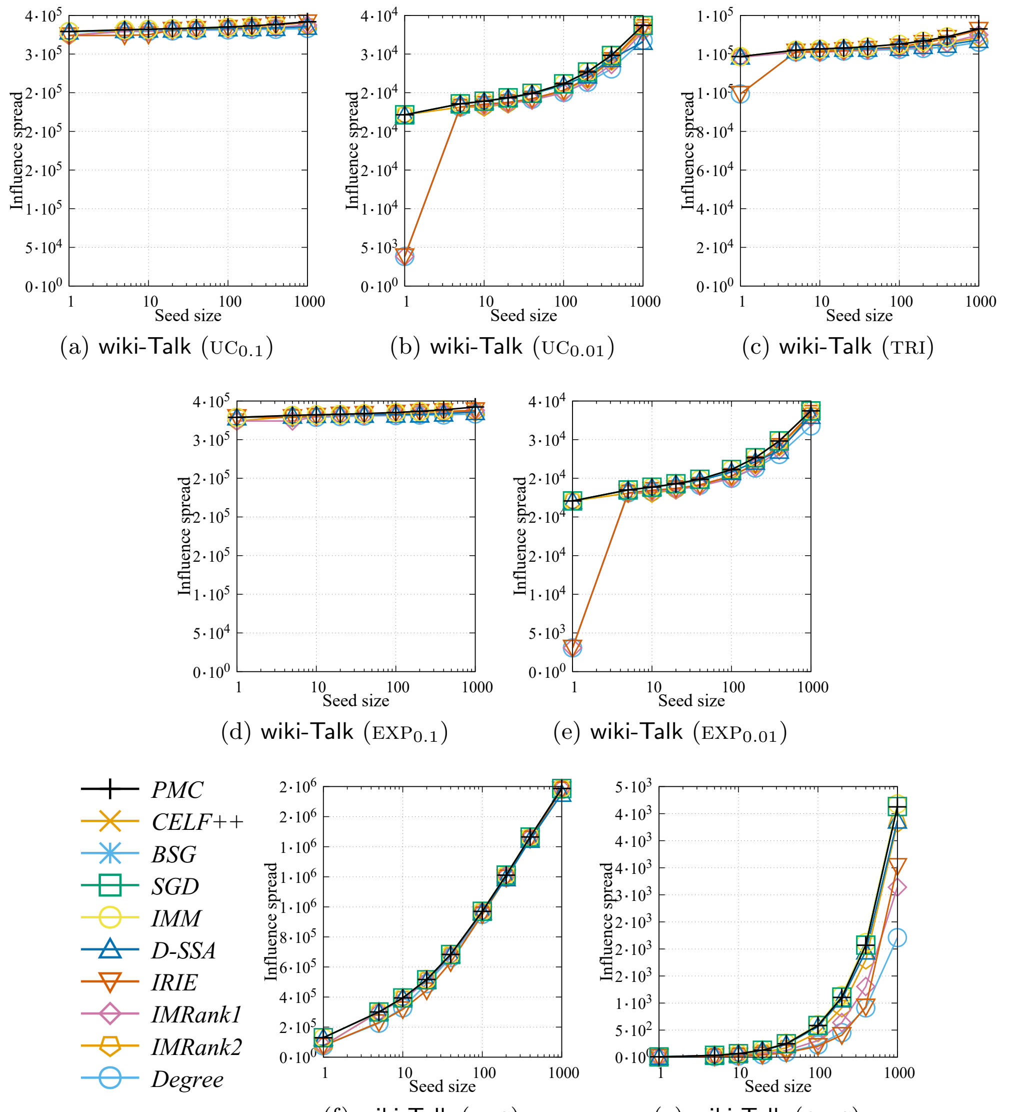


(f) web-Stanford (IWC)



(g) web-Stanford (OWC)

Figure B.10: Influence spread of each algorithm for web-Stanford network.

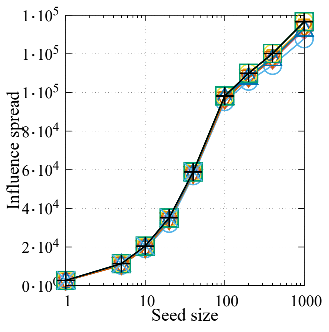


(a) wiki-Talk (UC$_{0.1}$) (b) wiki-Talk (UC$_{0.01}$) (c) wiki-Talk (TRI)

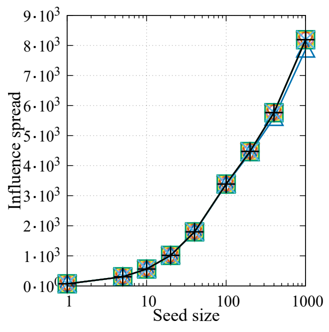
(d) wiki-Talk (EXP$_{0.1}$) (e) wiki-Talk (EXP$_{0.01}$)

(f) wiki-Talk (IWC) (g) wiki-Talk (OWC)

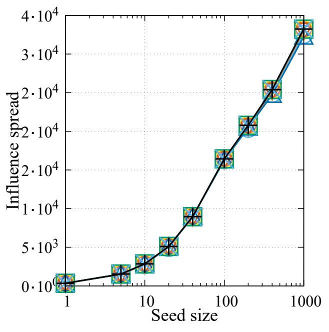
Figure B.11: Influence spread of each algorithm for wiki-Talk network.



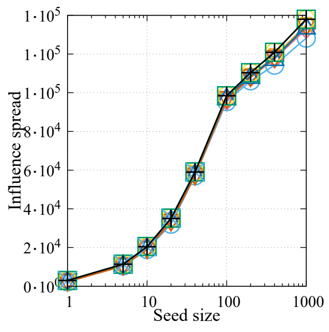
(a) web-Google ($\text{UC}_{0.1}$)



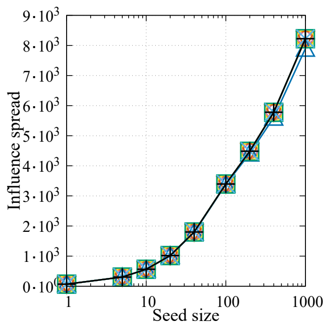
(b) web-Google ($\text{UC}_{0.01}$)



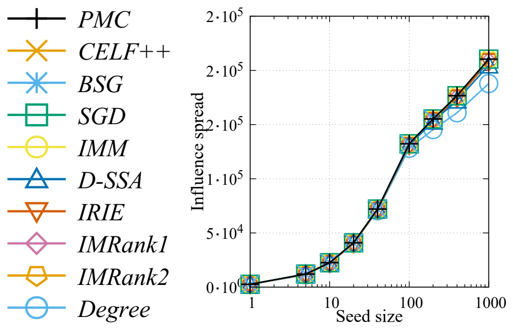
(c) web-Google (TRI)



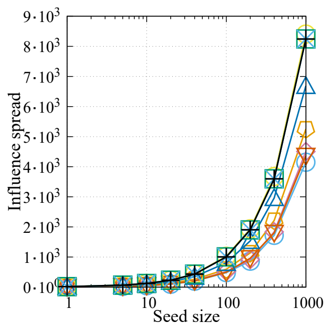
(d) web-Google ($\text{EXP}_{0.1}$)



(e) web-Google ($\text{EXP}_{0.01}$)

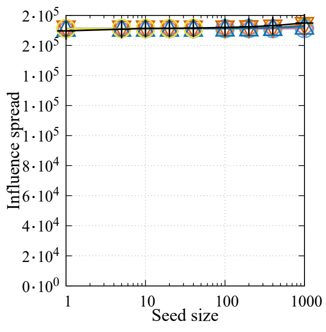


(f) web-Google (IWC)

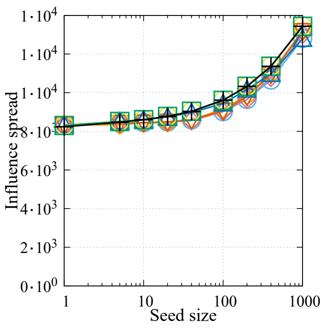


(g) web-Google (OWC)

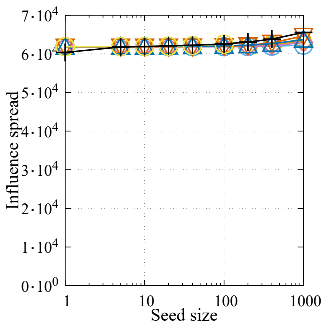
Figure B.12: Influence spread of each algorithm for web-Google network.



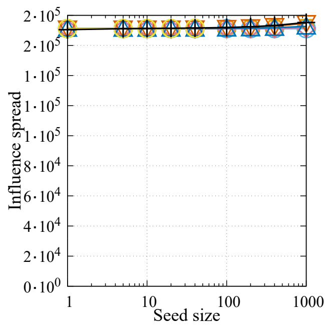
(a) com-Youtube ($\mathrm{UC}_{0.1}$)



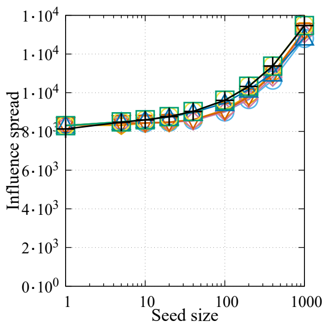
(b) com-Youtube ($\mathrm{UC}_{0.01}$)



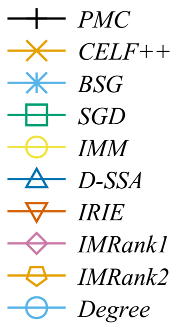
(c) com-Youtube (TRI)

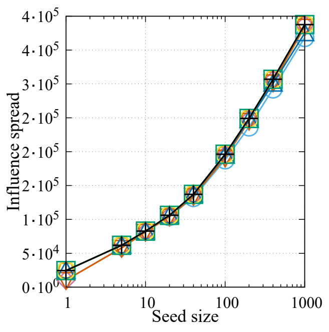


(d) com-Youtube ($\mathrm{EXP}_{0.1}$)

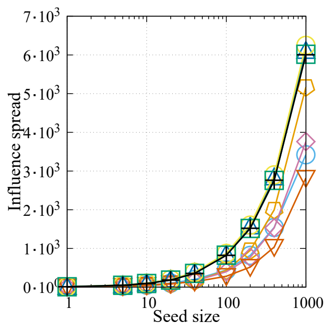


(e) com-Youtube ($\mathrm{EXP}_{0.01}$)



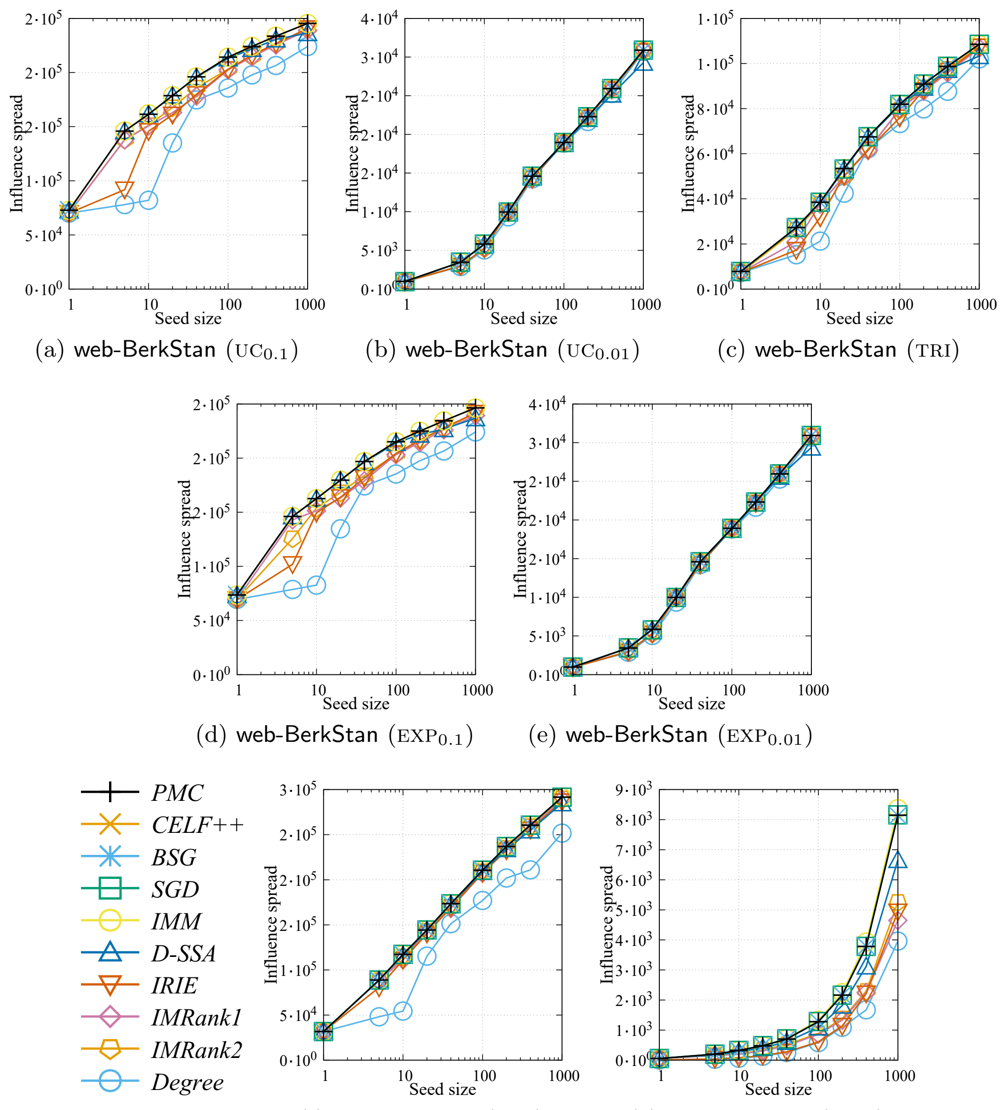


(f) com-Youtube (IWC)



(g) com-Youtube (OWC)

Figure B.13: Influence spread of each algorithm for com-Youtube network.

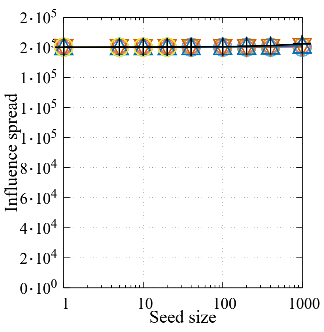


(a) web-BerkStan (UC$_{0.1}$) (b) web-BerkStan (UC$_{0.01}$) (c) web-BerkStan (TRI)

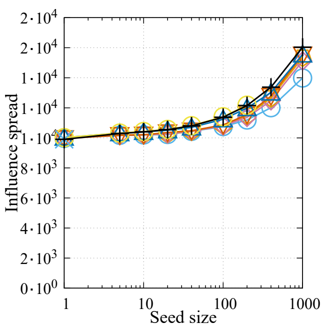
(d) web-BerkStan (EXP$_{0.1}$) (e) web-BerkStan (EXP$_{0.01}$)

(f) web-BerkStan (IWC) (g) web-BerkStan (OWC)

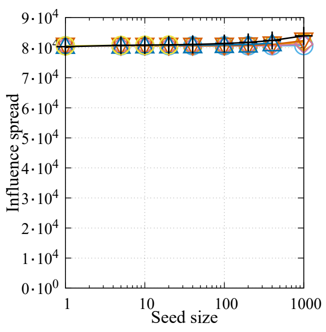
Figure B.14: Influence spread of each algorithm for web-BerkStan network.



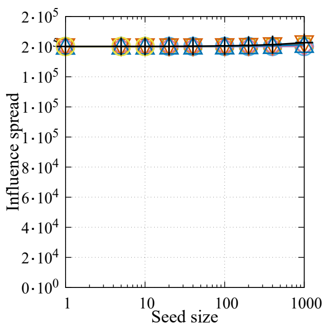
(a) higgs-twitter (UC$_{0.1}$)



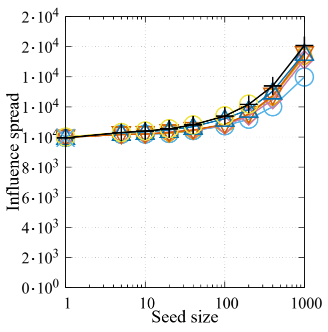
(b) higgs-twitter (UC$_{0.01}$)



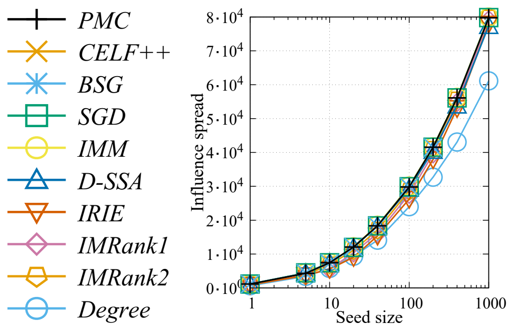
(c) higgs-twitter (TRI)



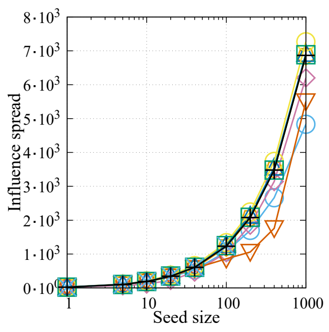
(d) higgs-twitter (EXP$_{0.1}$)



(e) higgs-twitter (EXP$_{0.01}$)

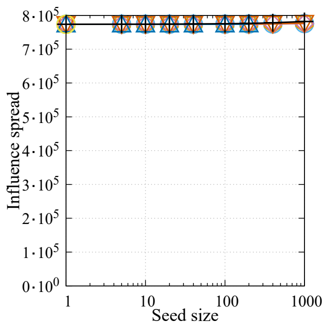


(f) higgs-twitter (IWC)

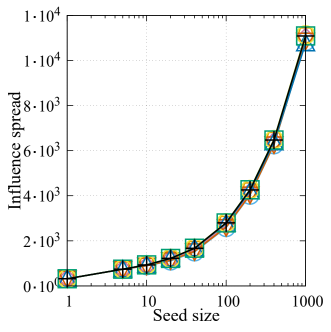


(g) higgs-twitter (OWC)

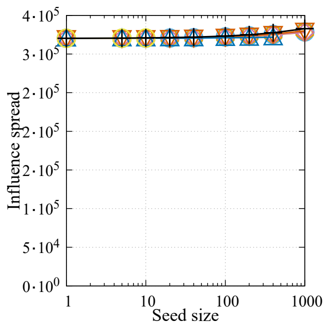
Figure B.15: Influence spread of each algorithm for higgs-twitter network.



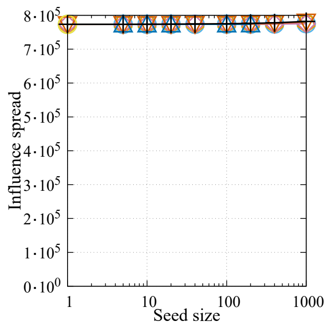
(a) soc-Pokec ($\mathrm{UC}_{0.1}$)



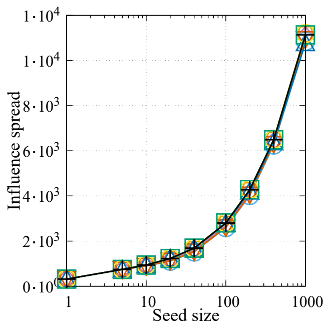
(b) soc-Pokec ($\mathrm{UC}_{0.01}$)



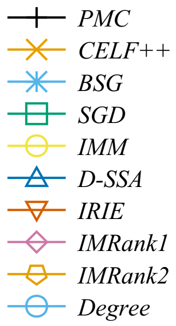
(c) soc-Pokec (TRI)

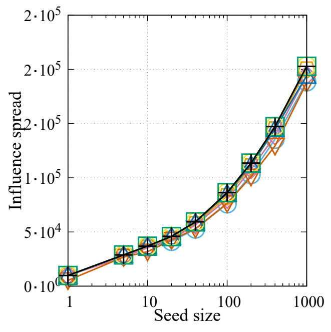


(d) soc-Pokec ($\mathrm{EXP}_{0.1}$)

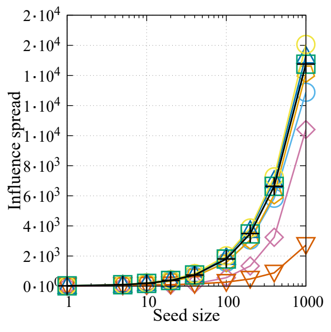


(e) soc-Pokec ($\mathrm{EXP}_{0.01}$)



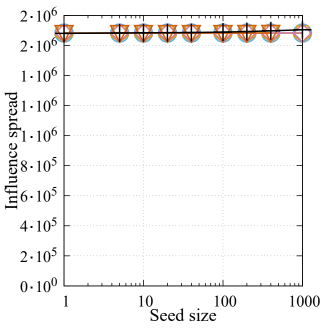


(f) soc-Pokec (IWC)

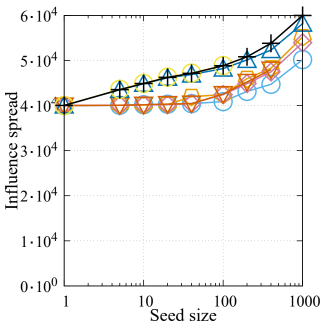


(g) soc-Pokec (OWC)

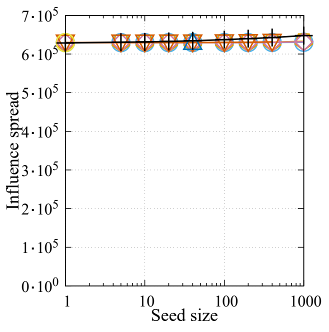
Figure B.16: Influence spread of each algorithm for soc-Pokec network.



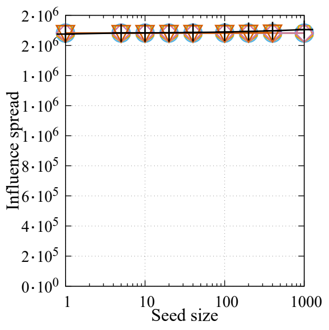
(a) soc-LiveJournal1 (UC$_{0.1}$)



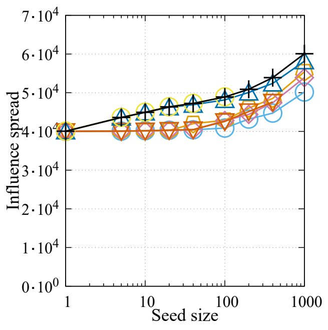
(b) soc-LiveJournal1 (UC$_{0.01}$)



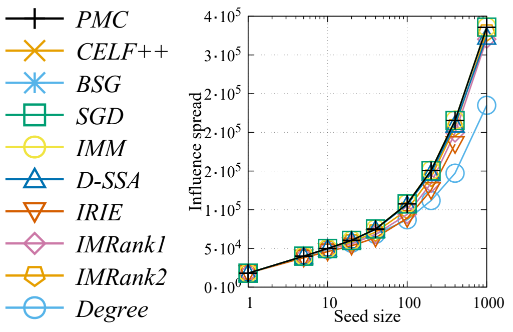
(c) soc-LiveJournal1 (TRI)



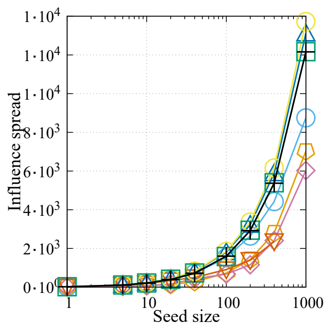
(d) soc-LiveJournal1 (EXP$_{0.1}$)



(e) soc-LiveJournal1 (EXP$_{0.01}$)

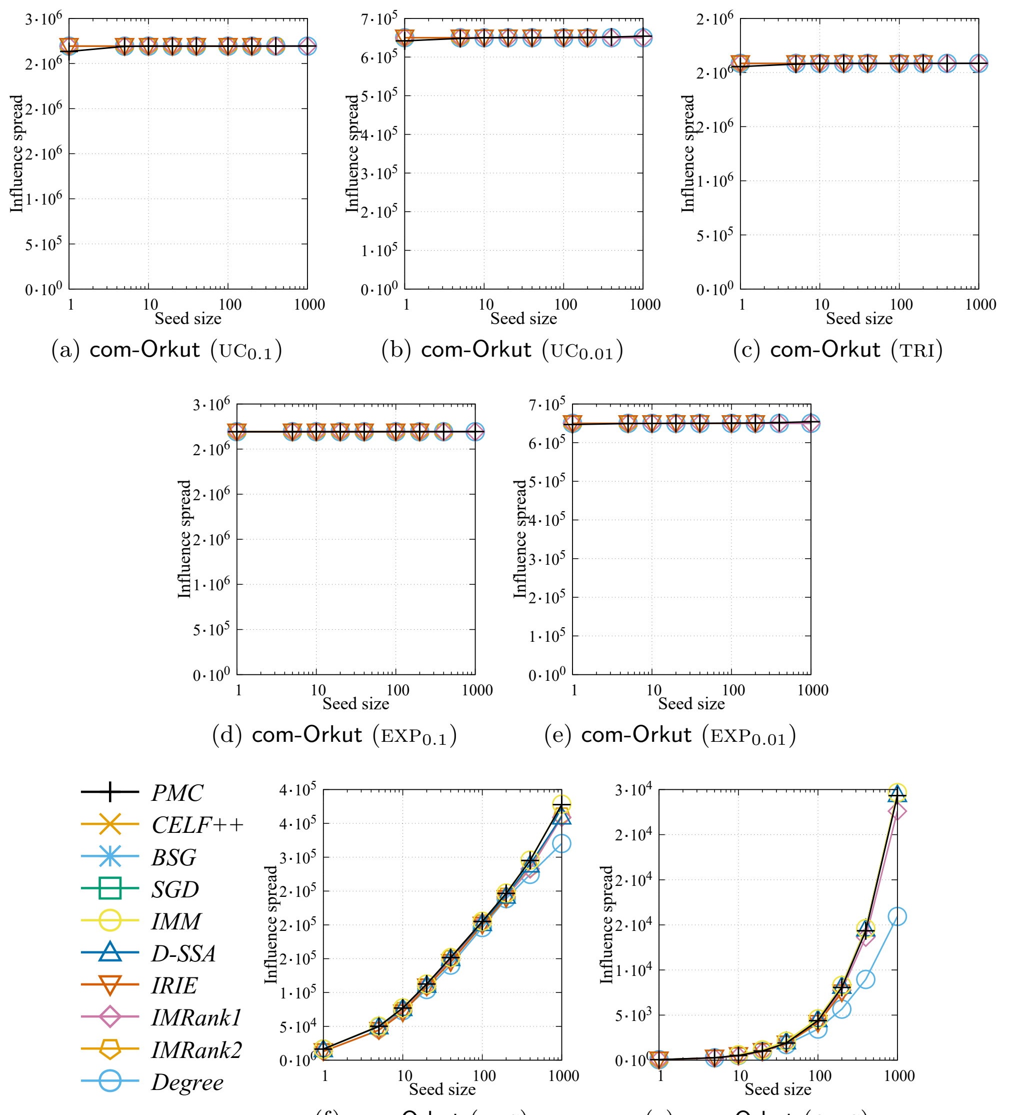


(f) soc-LiveJournal1 (IWC)



(g) soc-LiveJournal1 (OWC)

Figure B.17: Influence spread of each algorithm for soc-LiveJournal1 network.

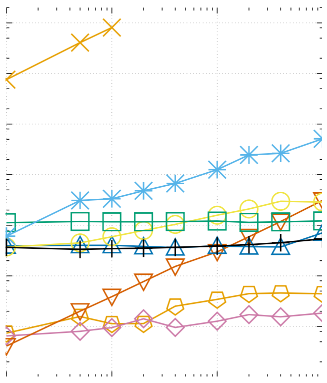


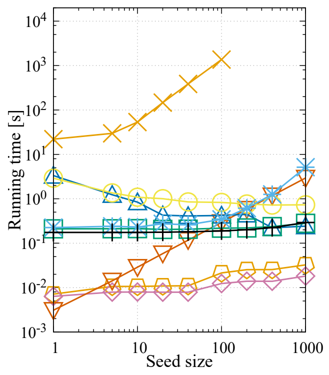
(a) com-Orkut (UC$_{0.1}$) (b) com-Orkut (UC$_{0.01}$) (c) com-Orkut (TRI)

(d) com-Orkut (EXP$_{0.1}$) (e) com-Orkut (EXP$_{0.01}$)

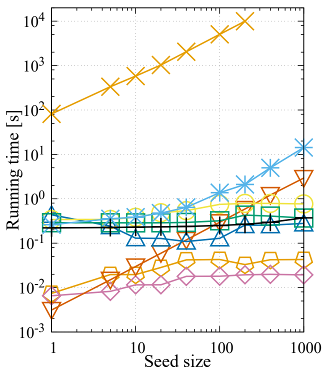
(f) com-Orkut (IWC) (g) com-Orkut (OWC)

Figure B.18: Influence spread of each algorithm for com-Orkut network.

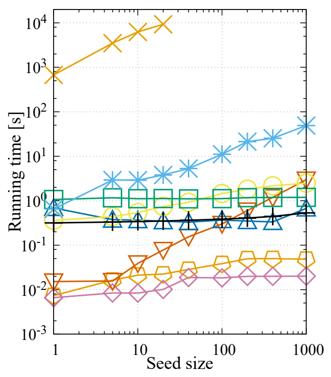

(a) ca-GrQc (UC$_{0.1}$)



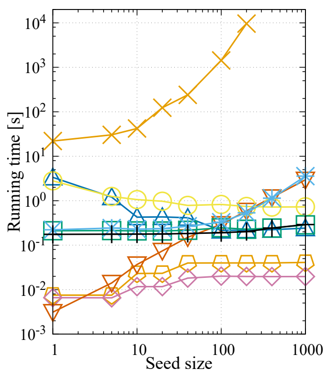
(b) ca-GrQc (UC$_{0.01}$)



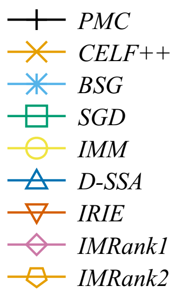
(c) ca-GrQc (TRI)

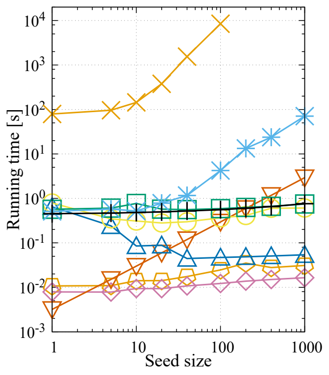


(d) ca-GrQc (EXP$_{0.1}$)

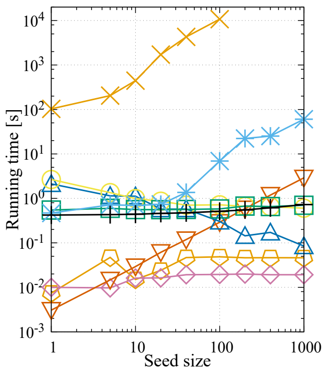


(e) ca-GrQc (EXP$_{0.01}$)



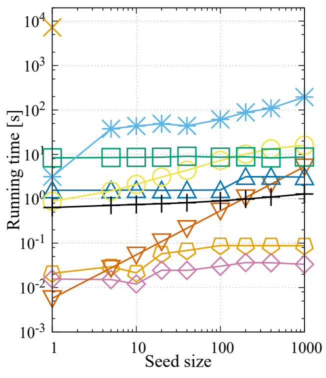


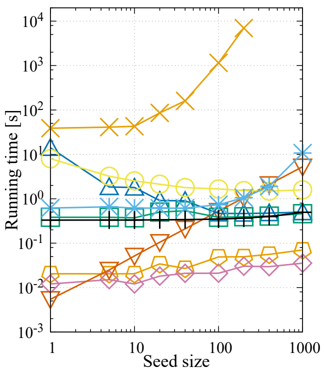
(f) ca-GrQc (IWC)

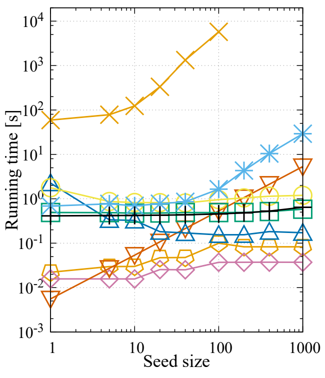


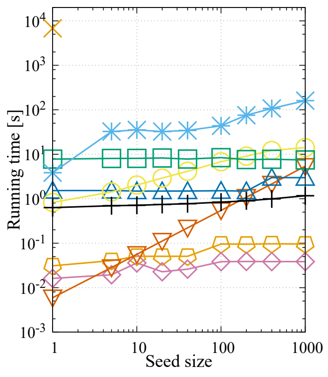
(g) ca-GrQc (OWC)

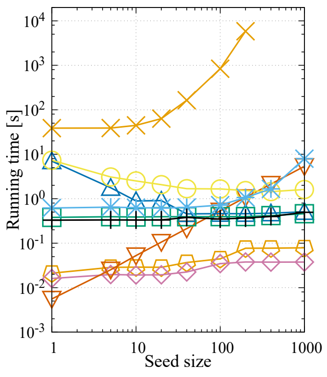
Figure B.19: Running time of each algorithm for ca-GrQc network.

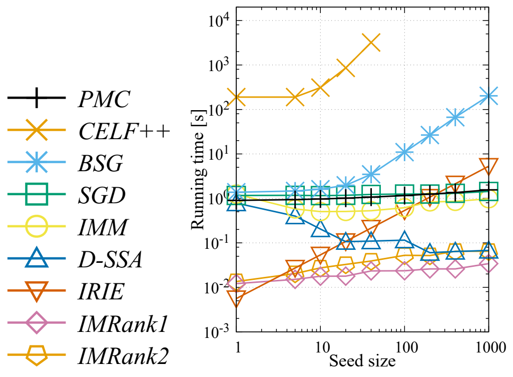

(a) ca-HepTh ($\text{UC}_{0.1}$)


(b) ca-HepTh ($\text{UC}_{0.01}$)


(c) ca-HepTh (TRI)


(d) ca-HepTh ($\text{EXP}_{0.1}$)


(e) ca-HepTh ($\text{EXP}_{0.01}$)



(f) ca-HepTh (IWC)

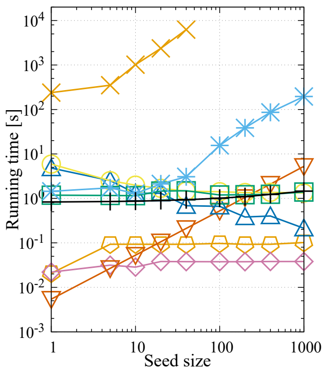
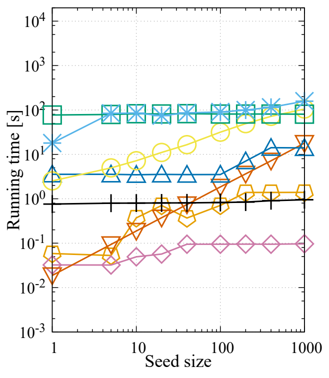
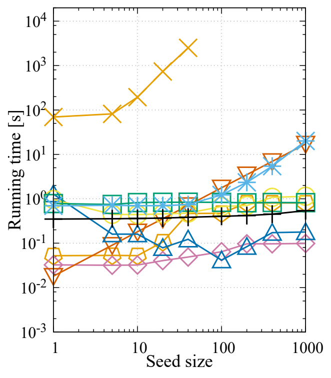

(g) ca-HepTh (OWC)

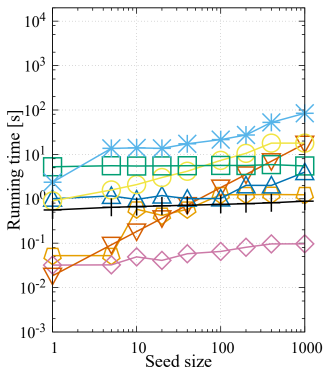
Figure B.20: Running time of each algorithm for ca-HepTh network.



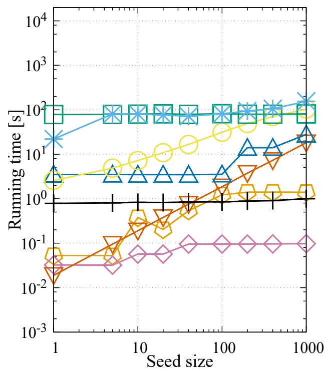
(a) wiki-Vote (UC$_{0.1}$)



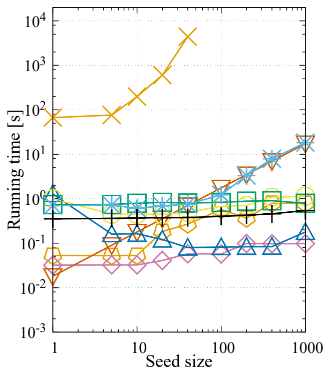
(b) wiki-Vote (UC$_{0.01}$)



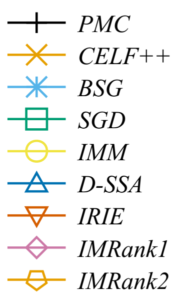
(c) wiki-Vote (TRI)

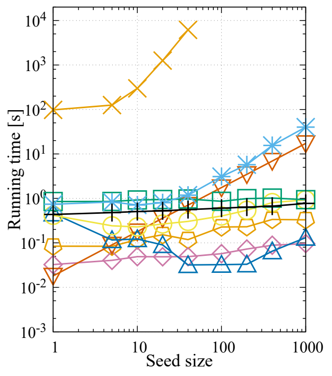


(d) wiki-Vote (EXP$_{0.1}$)

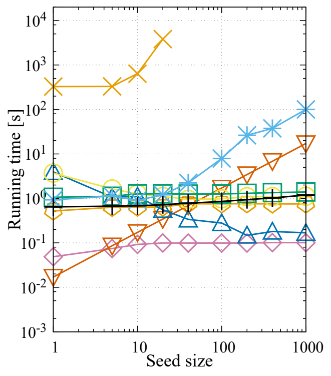


(e) wiki-Vote (EXP$_{0.01}$)



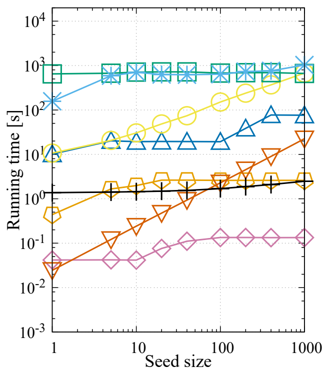


(f) wiki-Vote (IWC)

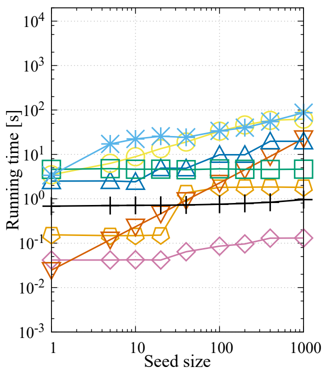


(g) wiki-Vote (OWC)

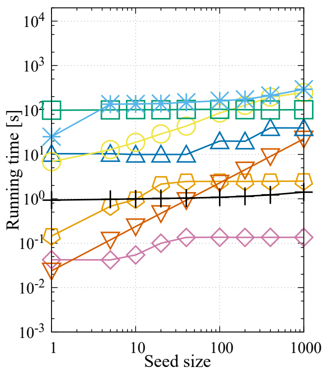
Figure B.21: Running time of each algorithm for wiki-Vote network.



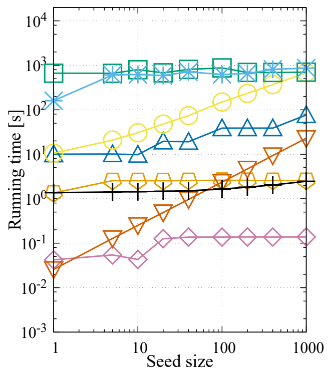
(a) ca-HepPh ($\text{UC}_{0.1}$)



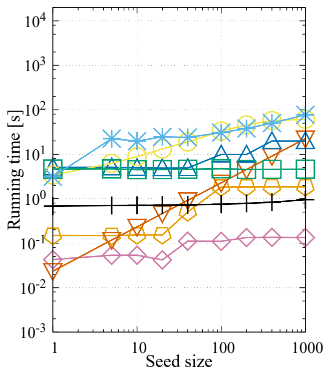
(b) ca-HepPh ($\text{UC}_{0.01}$)



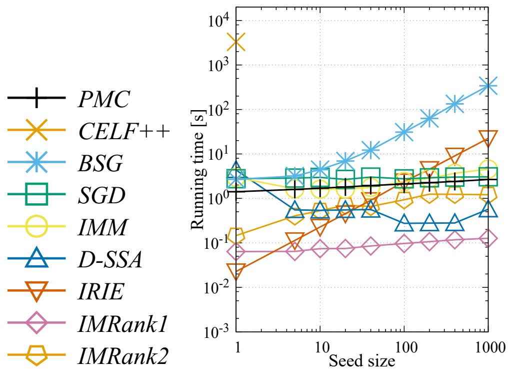
(c) ca-HepPh (TRI)



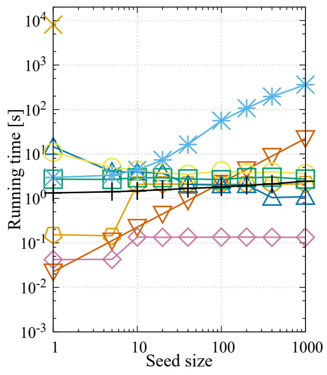
(d) ca-HepPh ($\text{EXP}_{0.1}$)



(e) ca-HepPh ($\text{EXP}_{0.01}$)

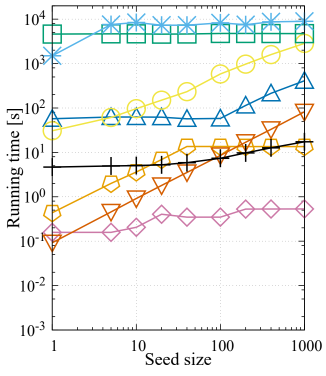


(f) ca-HepPh (IWC)

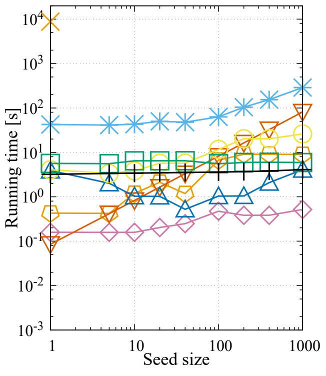


(g) ca-HepPh (OWC)

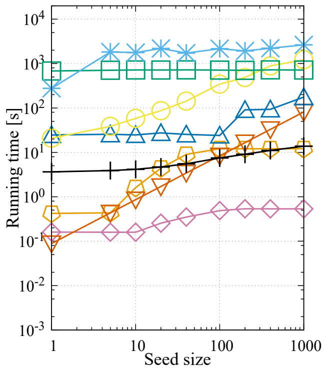
Figure B.22: Running time of each algorithm for ca-HepPh network.



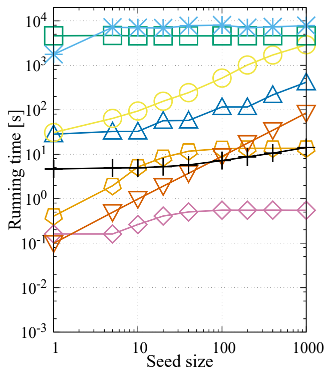
(a) soc-Epinions1 (UC$_{0.1}$)



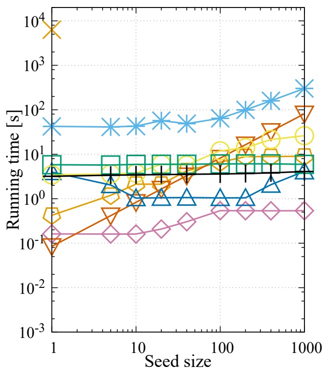
(b) soc-Epinions1 (UC$_{0.01}$)



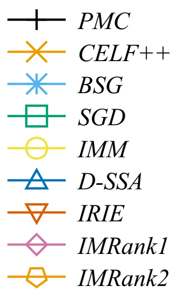
(c) soc-Epinions1 (TRI)

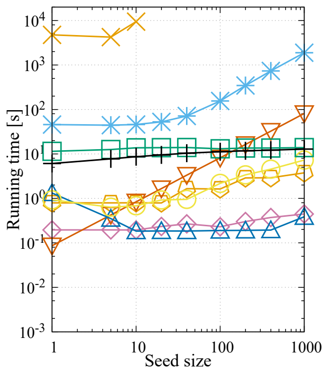


(d) soc-Epinions1 (EXP$_{0.1}$)

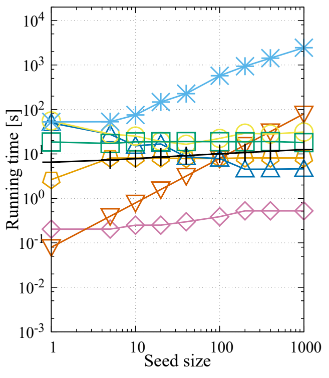


(e) soc-Epinions1 (EXP$_{0.01}$)



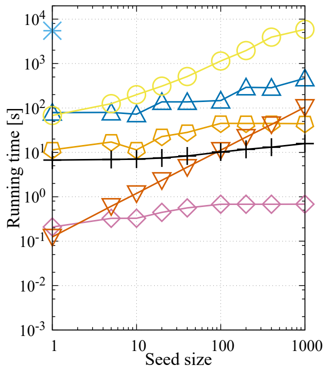


(f) soc-Epinions1 (IWC)

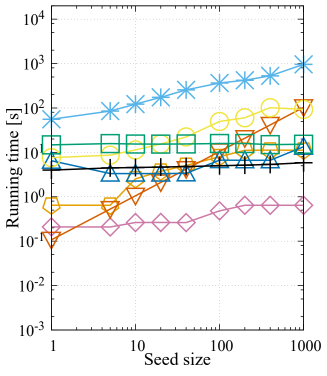


(g) soc-Epinions1 (OWC)

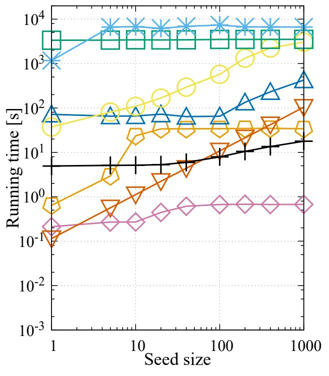
Figure B.23: Running time of each algorithm for soc-Epinions1 network.



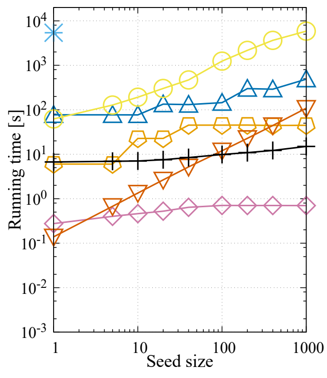
(a) soc-Slashdot0922 ($\mathrm{UC}_{0.1}$)



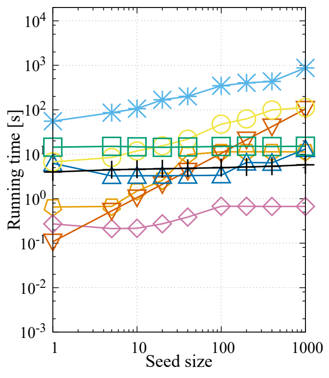
(b) soc-Slashdot0922 ($\mathrm{UC}_{0.01}$)



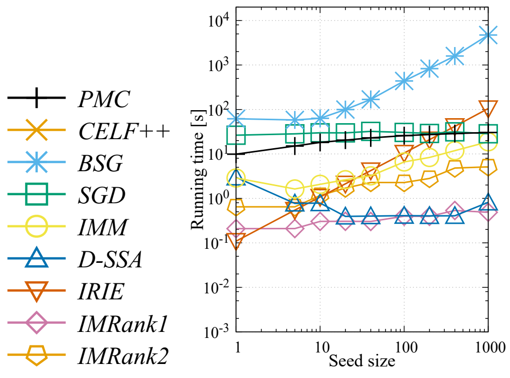
(c) soc-Slashdot0922 (TRI)



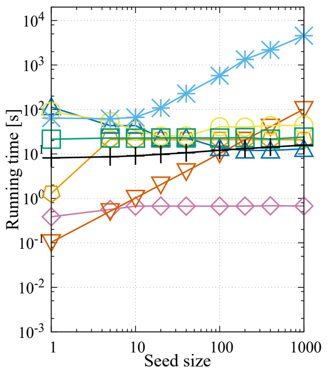
(d) soc-Slashdot0922 ($\mathrm{EXP}_{0.1}$)



(e) soc-Slashdot0922 ($\mathrm{EXP}_{0.01}$)

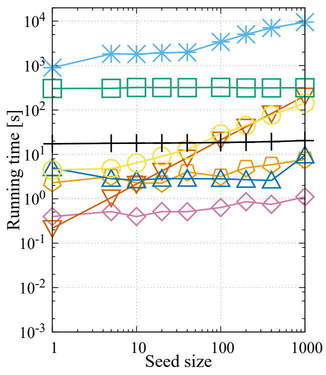


(f) soc-Slashdot0922 (IWC)

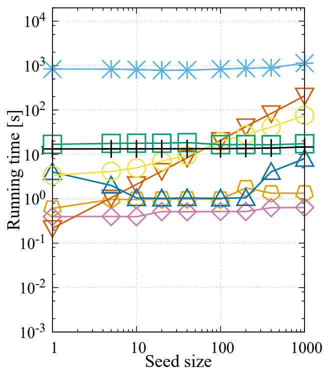


(g) soc-Slashdot0922 (OWC)

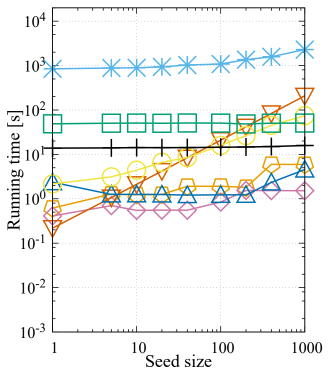
Figure B.24: Running time of each algorithm for soc-Slashdot0922 network.



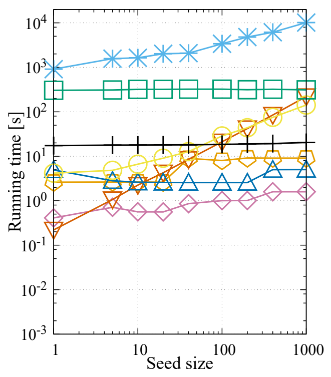
(a) web-NotreDame ($\mathrm{UC}_{0.1}$)



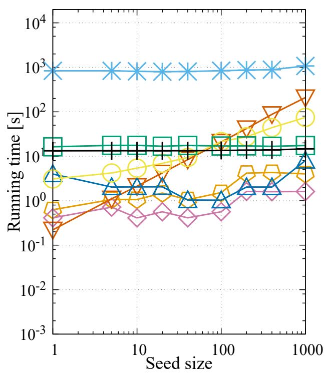
(b) web-NotreDame ($\mathrm{UC}_{0.01}$)



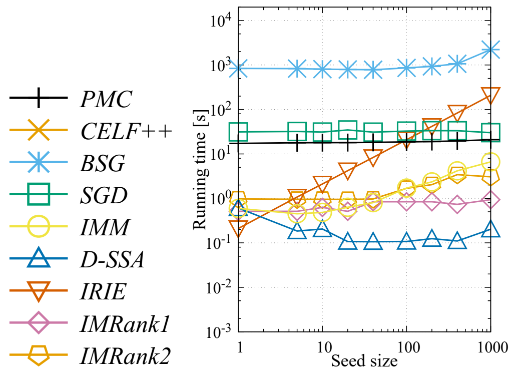
(c) web-NotreDame (TRI)



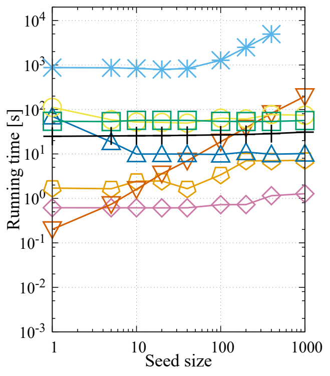
(d) web-NotreDame ($\mathrm{EXP}_{0.1}$)



(e) web-NotreDame ($\mathrm{EXP}_{0.01}$)

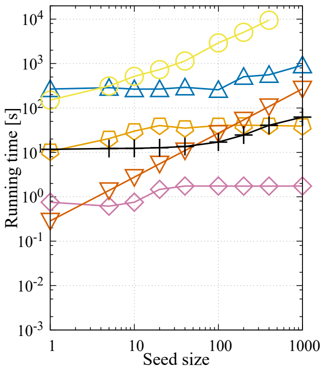


(f) web-NotreDame (IWC)

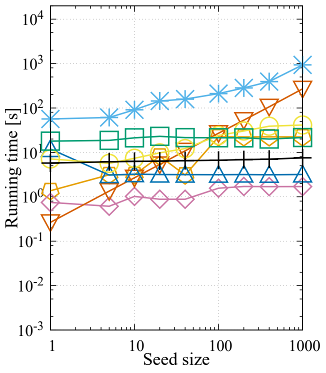


(g) web-NotreDame (OWC)

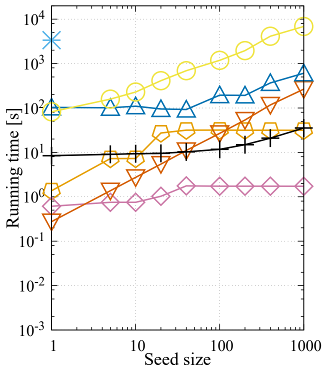
Figure B.25: Running time of each algorithm for web-NotreDame network.



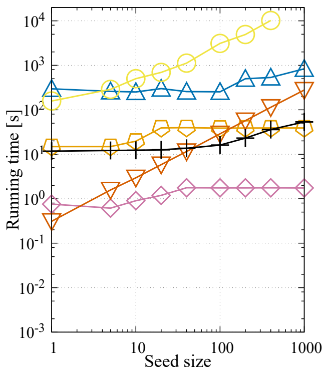
(a) ego-Twitter (UC$_{0.1}$)



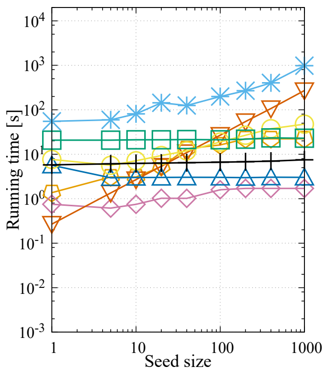
(b) ego-Twitter (UC$_{0.01}$)



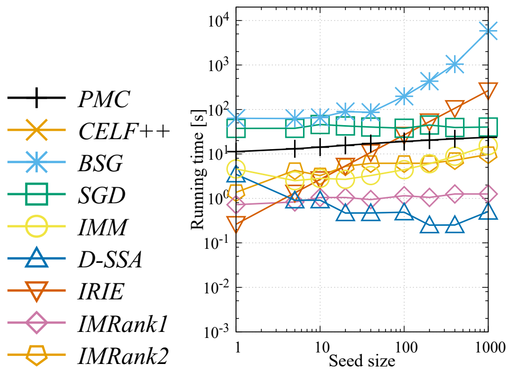
(c) ego-Twitter (TRI)



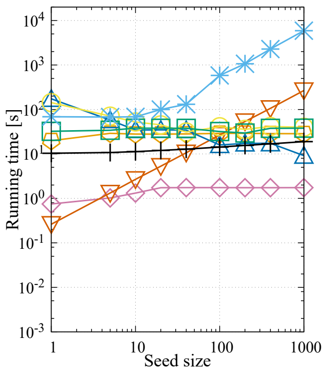
(d) ego-Twitter (EXP$_{0.1}$)



(e) ego-Twitter (EXP$_{0.01}$)

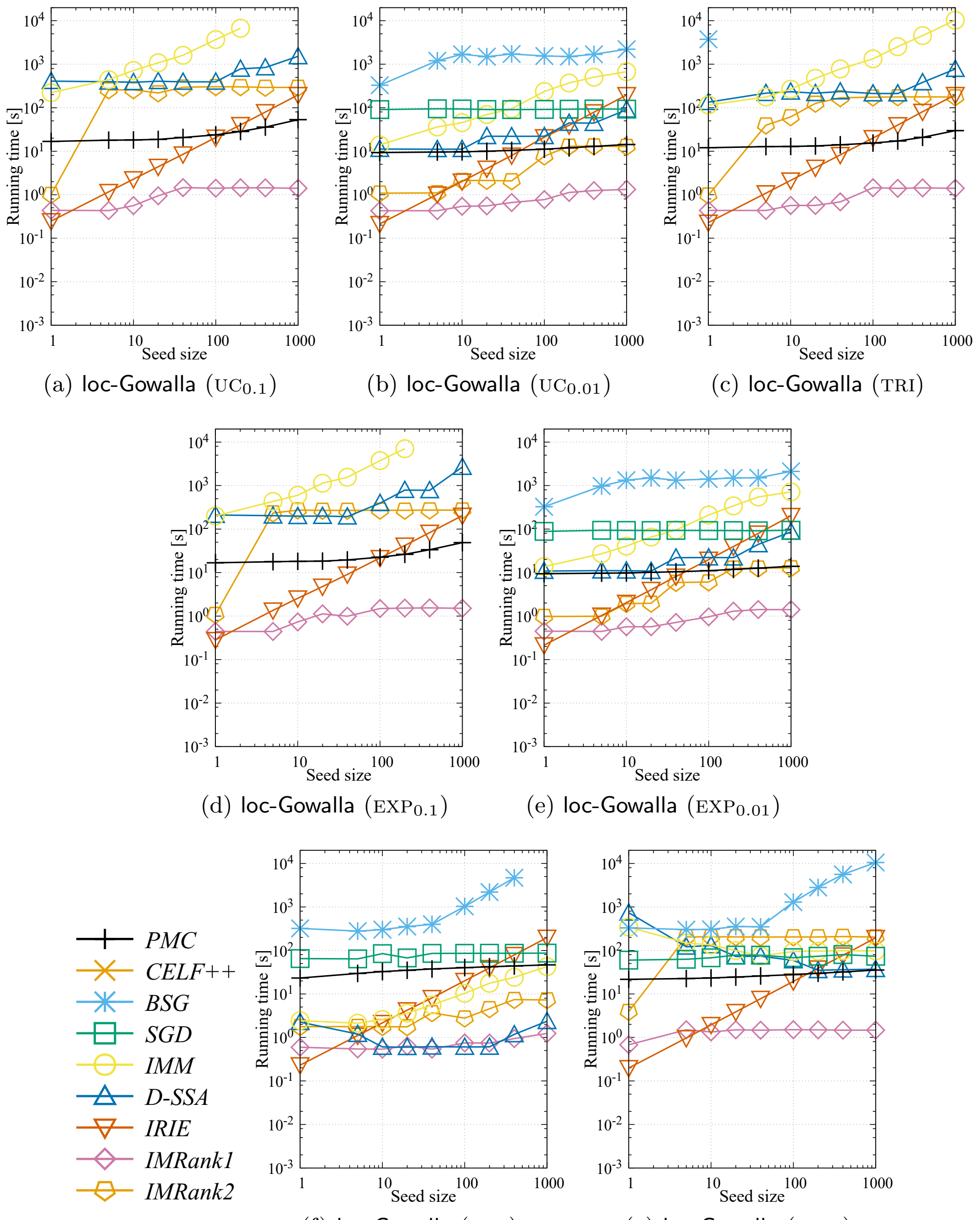


(f) ego-Twitter (IWC)



(g) ego-Twitter (OWC)

Figure B.26: Running time of each algorithm for ego-Twitter network.

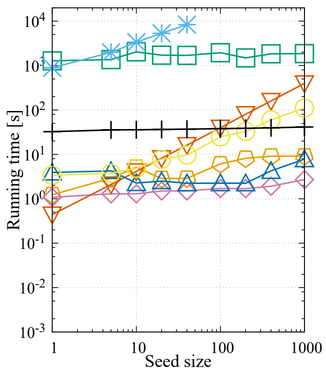


(a) loc-Gowalla ($\text{UC}_{0.1}$) (b) loc-Gowalla ($\text{UC}_{0.01}$) (c) loc-Gowalla (TRI)

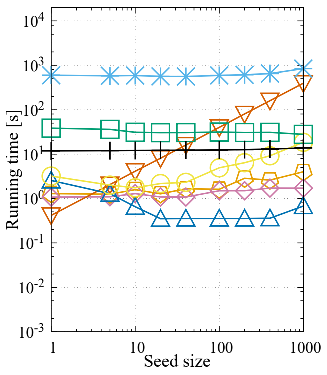
(d) loc-Gowalla ($\text{EXP}_{0.1}$) (e) loc-Gowalla ($\text{EXP}_{0.01}$)

(f) loc-Gowalla (IWC) (g) loc-Gowalla (OWC)

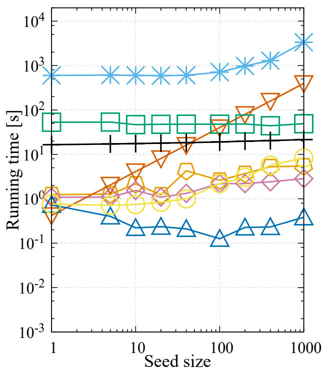
Figure B.27: Running time of each algorithm for loc-Gowalla network.



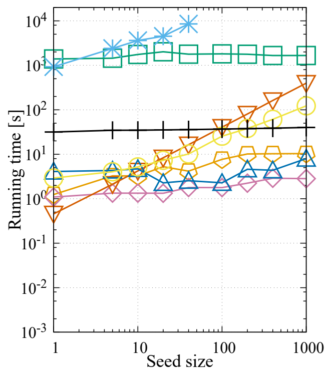
(a) web-Stanford ($\text{UC}_{0.1}$)



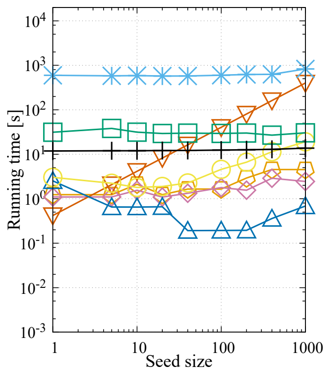
(b) web-Stanford ($\text{UC}_{0.01}$)



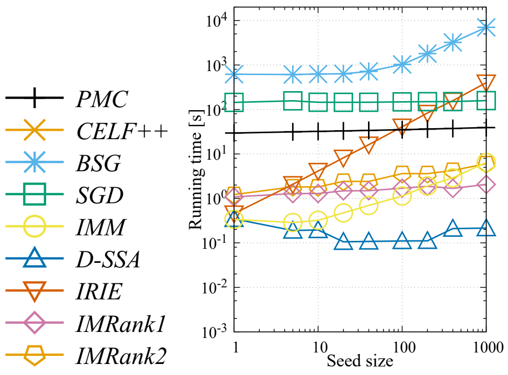
(c) web-Stanford (TRI)



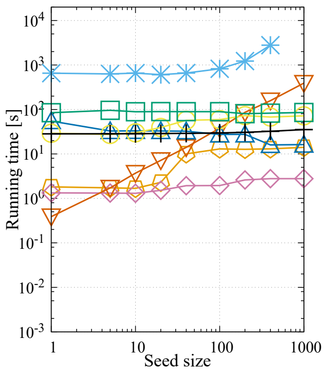
(d) web-Stanford ($\text{EXP}_{0.1}$)



(e) web-Stanford ($\text{EXP}_{0.01}$)

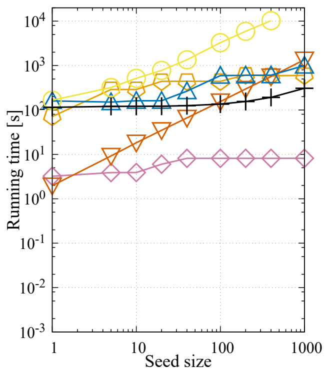


(f) web-Stanford (IWC)

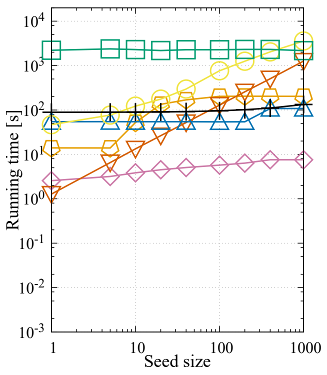


(g) web-Stanford (OWC)

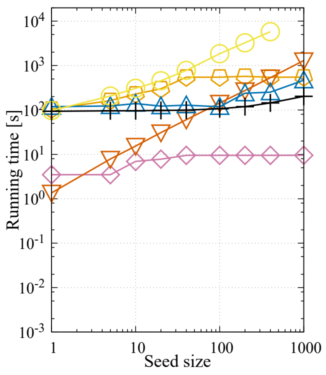
Figure B.28: Running time of each algorithm for web-Stanford network.



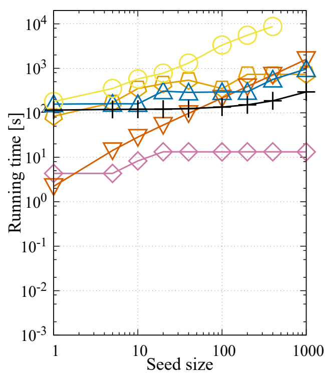
(a) wiki-Talk (UC$_{0.1}$)



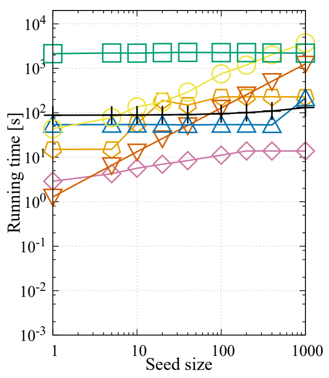
(b) wiki-Talk (UC$_{0.01}$)



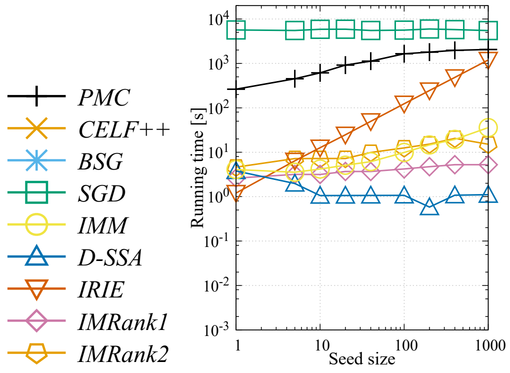
(c) wiki-Talk (TRI)



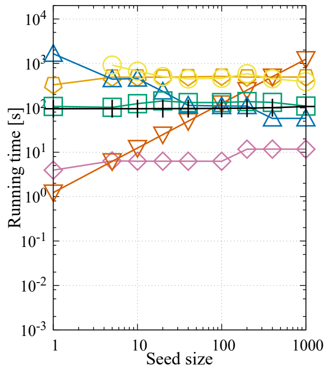
(d) wiki-Talk (EXP$_{0.1}$)



(e) wiki-Talk (EXP$_{0.01}$)

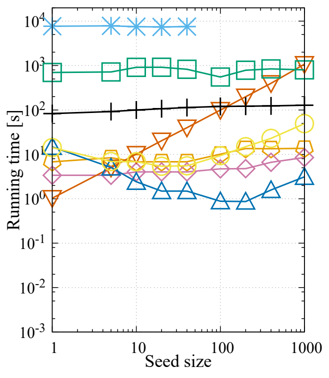


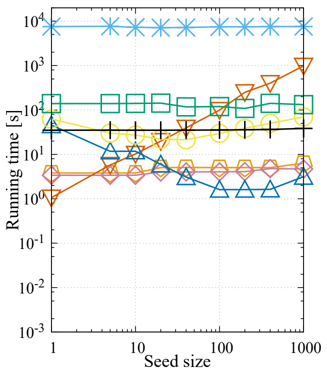
(f) wiki-Talk (IWC)

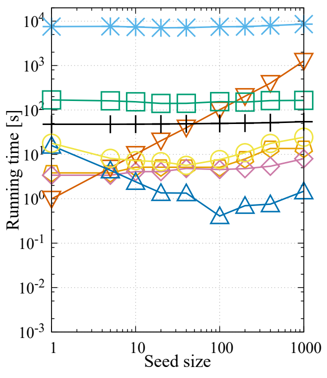


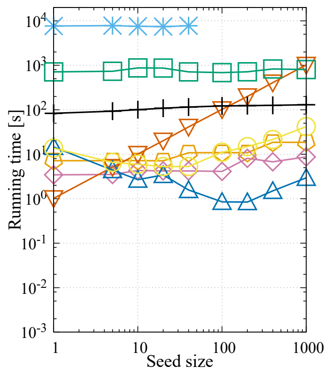
(g) wiki-Talk (OWC)

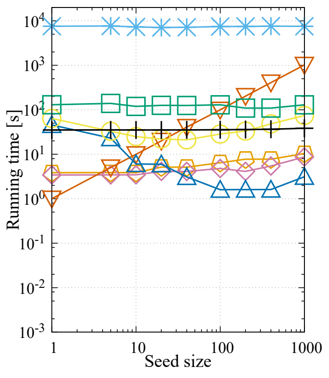
Figure B.29: Running time of each algorithm for wiki-Talk network.

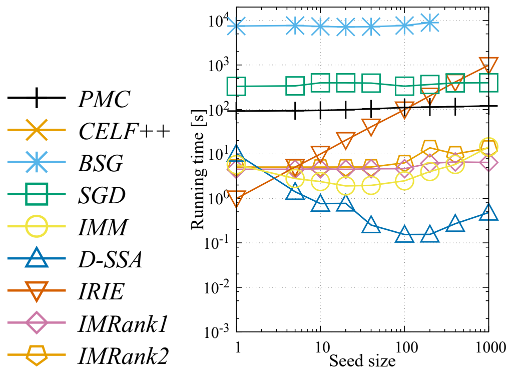

(a) web-Google (UC$_{0.1}$)


(b) web-Google (UC$_{0.01}$)


(c) web-Google (TRI)


(d) web-Google (EXP$_{0.1}$)


(e) web-Google (EXP$_{0.01}$)


(f) web-Google (IWC)

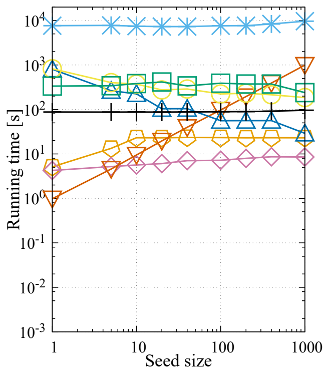
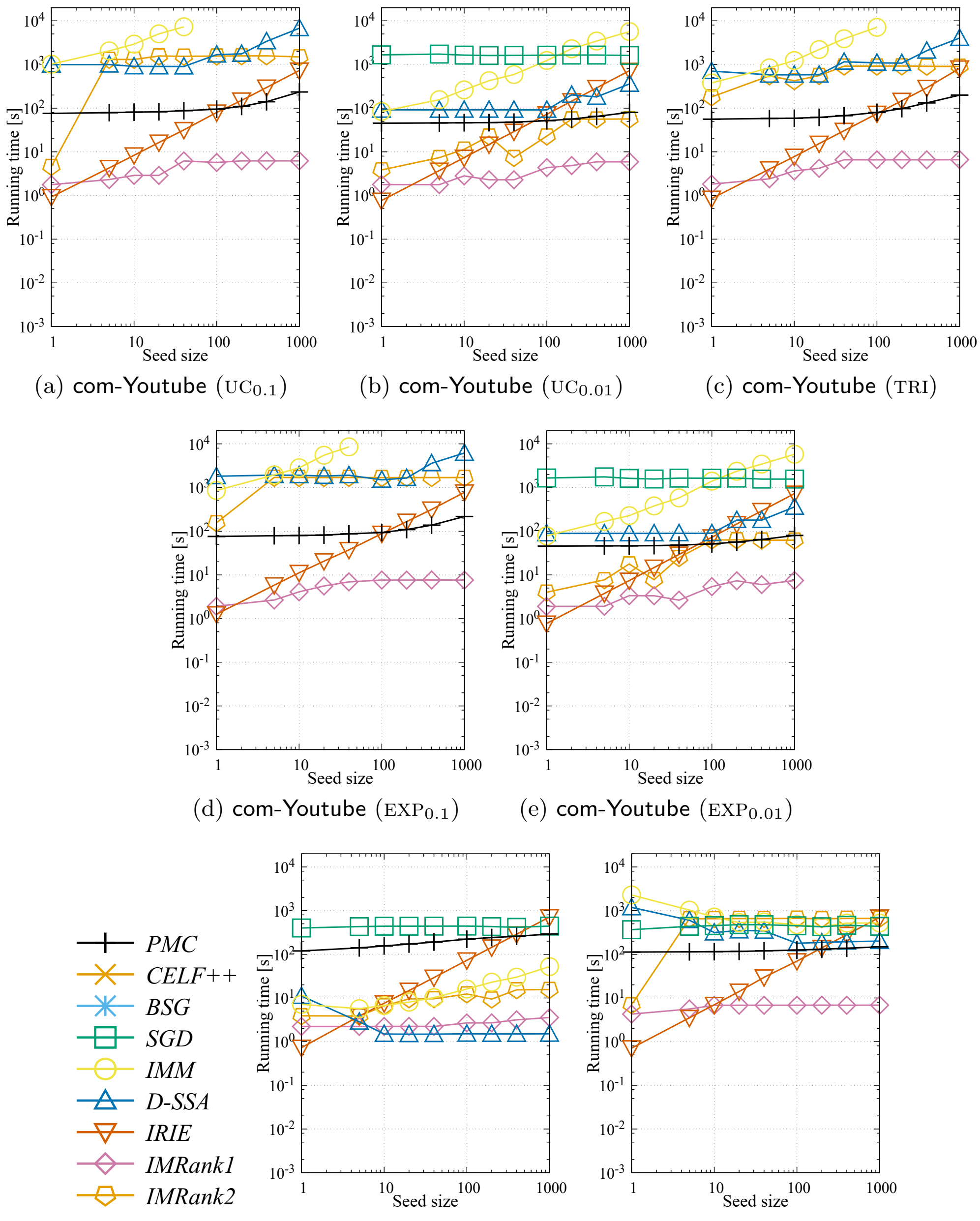
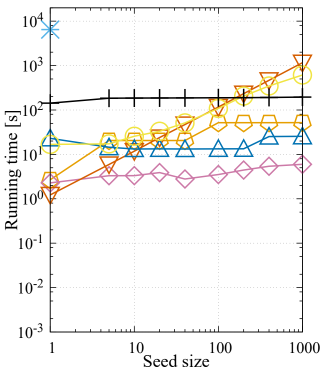

(g) web-Google (OWC)

Figure B.30: Running time of each algorithm for web-Google network.

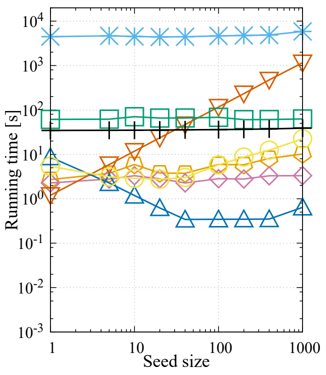


(a) com-Youtube ($\mathrm{UC}_{0.1}$)
(b) com-Youtube ($\mathrm{UC}_{0.01}$)
(c) com-Youtube (TRI)
(d) com-Youtube ($\mathrm{EXP}_{0.1}$)
(e) com-Youtube ($\mathrm{EXP}_{0.01}$)
(f) com-Youtube (IWC)
(g) com-Youtube (OWC)

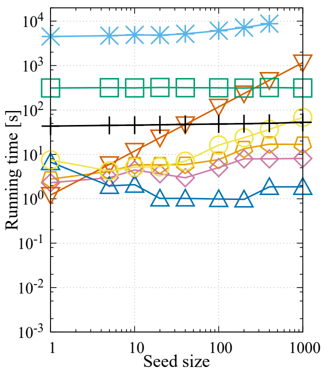
Figure B.31: Running time of each algorithm for com-Youtube network.



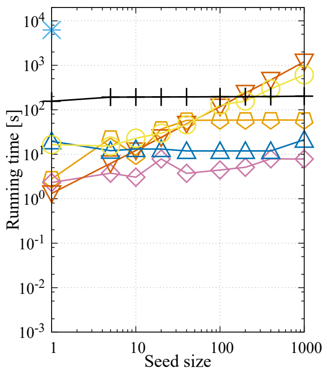
(a) web-BerkStan (UC$_{0.1}$)



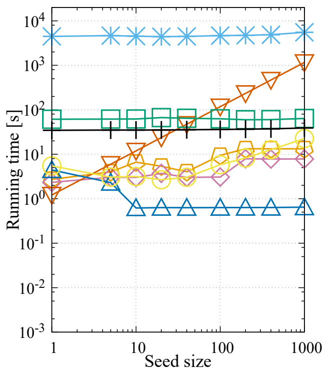
(b) web-BerkStan (UC$_{0.01}$)



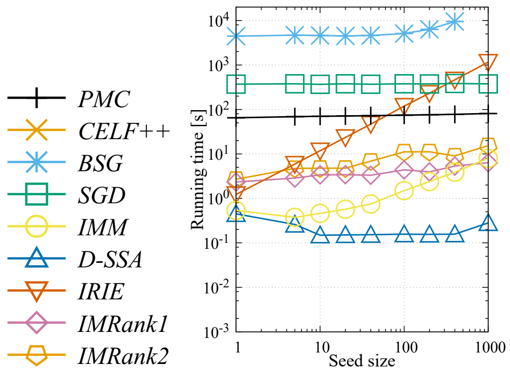
(c) web-BerkStan (TRI)



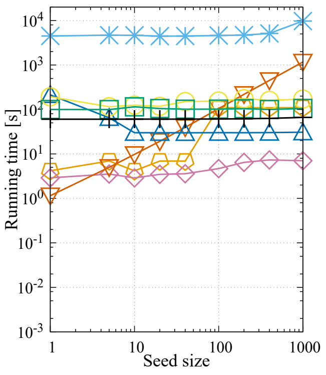
(d) web-BerkStan (EXP$_{0.1}$)



(e) web-BerkStan (EXP$_{0.01}$)

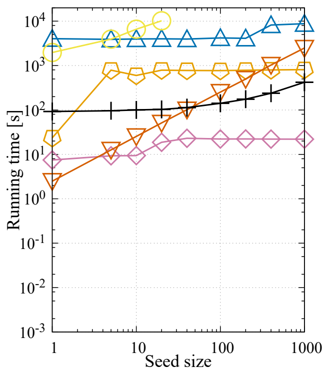


(f) web-BerkStan (IWC)

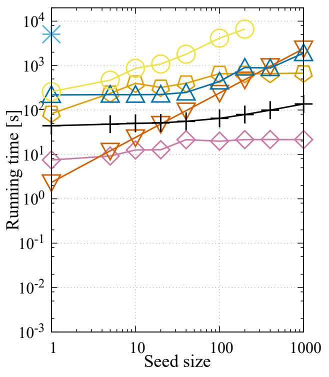


(g) web-BerkStan (OWC)

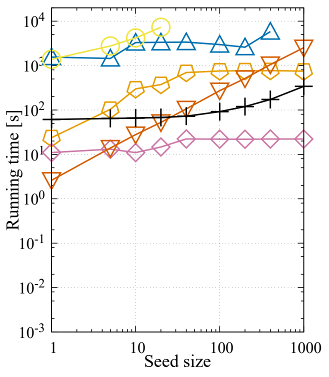
Figure B.32: Running time of each algorithm for web-BerkStan network.



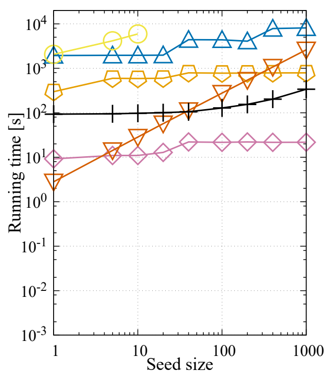
(a) higgs-twitter (UC$_{0.1}$)



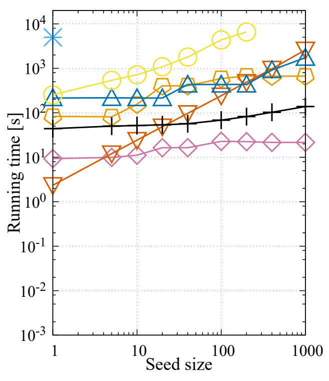
(b) higgs-twitter (UC$_{0.01}$)



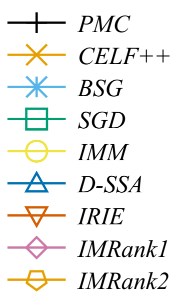
(c) higgs-twitter (TRI)

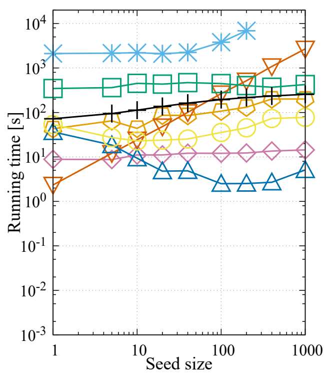


(d) higgs-twitter (EXP$_{0.1}$)

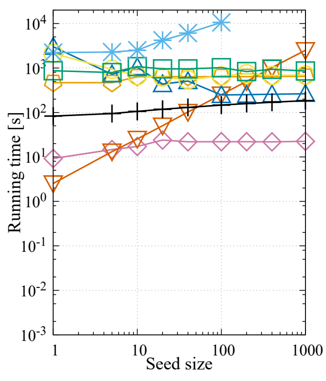


(e) higgs-twitter (EXP$_{0.01}$)



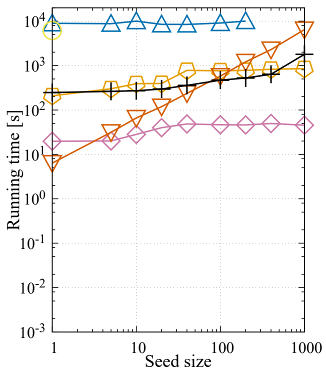


(f) higgs-twitter (IWC)

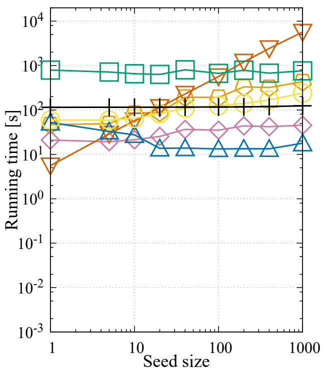


(g) higgs-twitter (OWC)

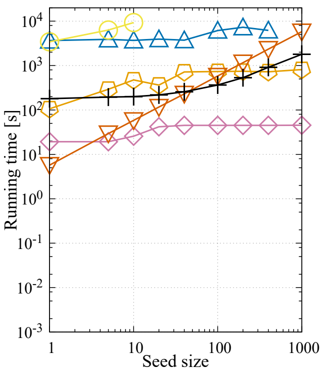
Figure B.33: Running time of each algorithm for higgs-twitter network.



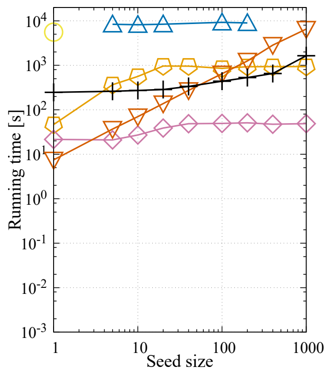
(a) soc-Pokec (UC$_{0.1}$)



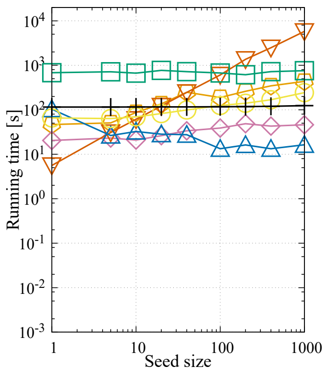
(b) soc-Pokec (UC$_{0.01}$)



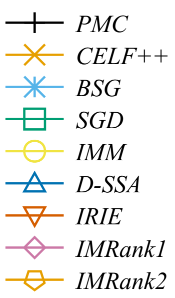
(c) soc-Pokec (TRI)

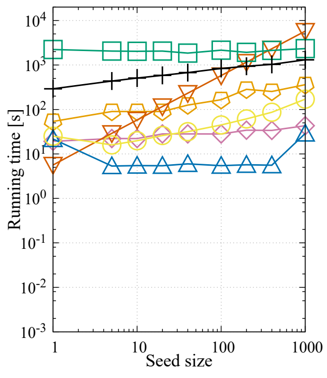


(d) soc-Pokec (EXP$_{0.1}$)

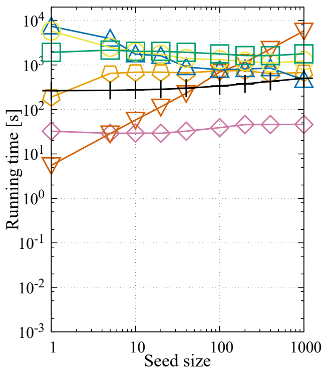


(e) soc-Pokec (EXP$_{0.01}$)



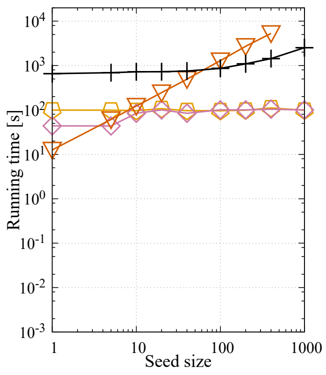


(f) soc-Pokec (IWC)

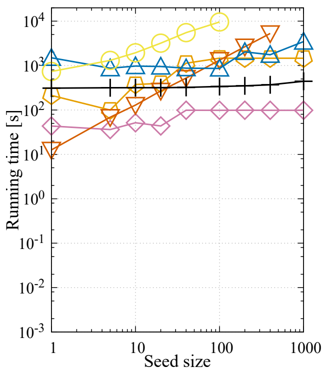


(g) soc-Pokec (OWC)

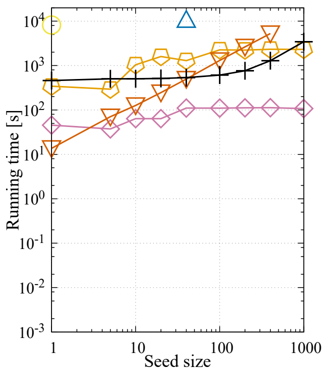
Figure B.34: Running time of each algorithm for soc-Pokec network.



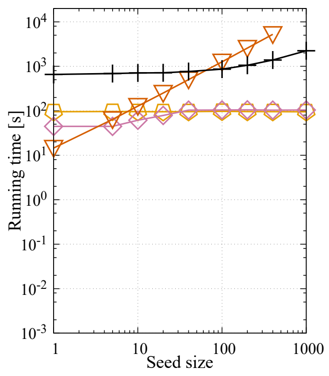
(a) soc-LiveJournal1 ($\mathrm{UC}_{0.1}$)



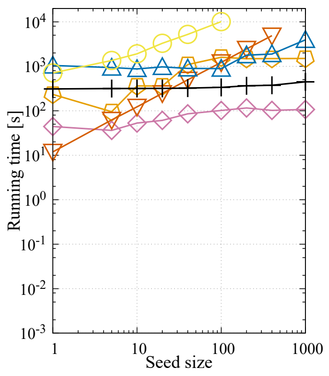
(b) soc-LiveJournal1 ($\mathrm{UC}_{0.01}$)



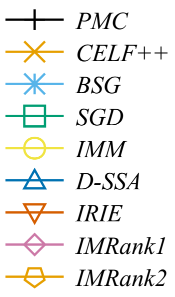
(c) soc-LiveJournal1 (TRI)

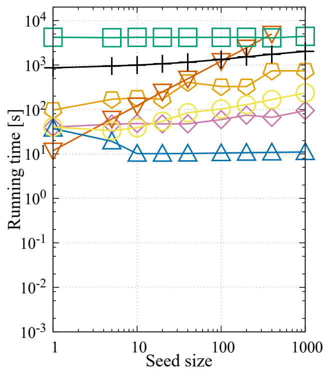


(d) soc-LiveJournal1 ($\mathrm{EXP}_{0.1}$)

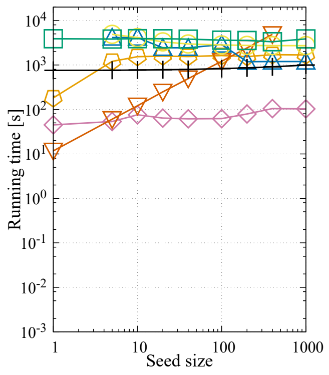


(e) soc-LiveJournal1 ($\mathrm{EXP}_{0.01}$)



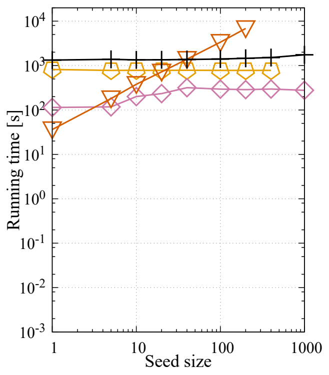


(f) soc-LiveJournal1 (IWC)

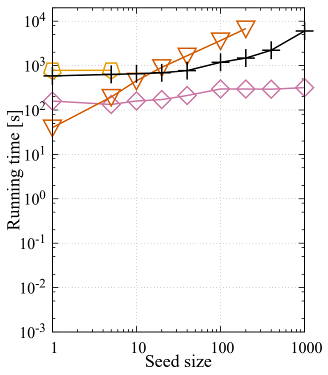


(g) soc-LiveJournal1 (OWC)

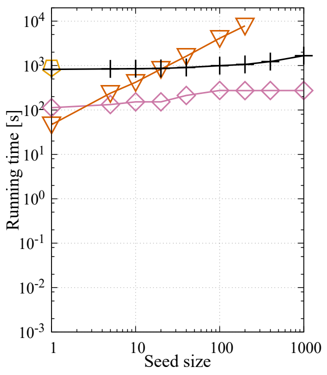
Figure B.35: Running time of each algorithm for soc-LiveJournal1 network.



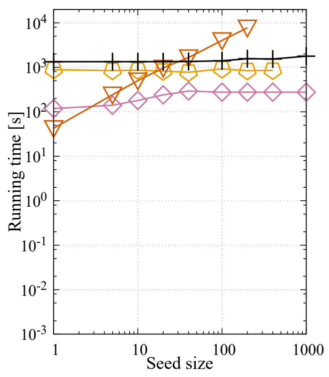
(a) com-Orkut (UC$_{0.1}$)



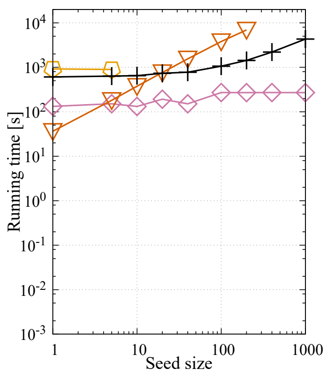
(b) com-Orkut (UC$_{0.01}$)



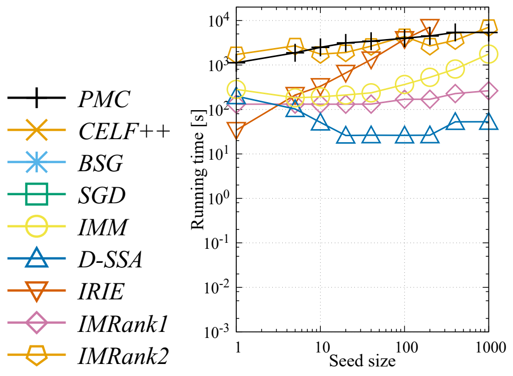
(c) com-Orkut (TRI)



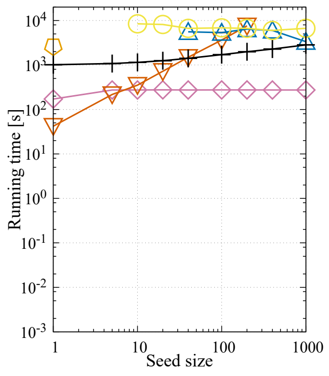
(d) com-Orkut (EXP$_{0.1}$)



(e) com-Orkut (EXP$_{0.01}$)



(f) com-Orkut (IWC)



(g) com-Orkut (OWC)

Figure B.36: Running time of each algorithm for com-Orkut network.

Appendix C

Additional Experimental Results in Chapter 7

This section provides additional experimental results under $\textsc{uc}_{0.1}$ and $\textsc{iwc}$. Table C.1 shows the run times and memory usages of both Algorithms 7.1 and 7.2. Table C.2 shows the numbers of vertices and edges, and the corresponding reduction ratio. Table C.3 shows the total run time and accuracy of each influence estimation method. Table C.4 shows the run time and solution quality of each influence maximization method. For the $\textsc{uc}_{0.1}$ setting, we observe similar trends to those for the $\textsc{exp}_{0.1}$ setting. On the other hand, our coarsening for $\textsc{iwc}$ is not much effective. It should be noted, however, that both influence estimation and influence maximization under $\textsc{iwc}$ can be solved quickly as shown in Tables C.3 and C.4.

Table C.1: Run time and memory usage of the proposed algorithm under UC$_{0.1}$ and IWC.

	uniform cascade (UC$_{0.1}$)				**in-degree weighted cascade** (IWC)			
	linear space (Alg. 7.1)		sublinear space (Alg. 7.2)		linear space (Alg. 7.1)		sublinear space (Alg. 7.2)	
dataset	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**
ca-GrQc	0.02 s	4 MB	0.45 s	2 MB	0.03 s	4 MB	0.21 s	2 MB
ca-HepTh	0.04 s	6 MB	0.32 s	2 MB	0.05 s	6 MB	0.33 s	2 MB
wiki-Vote	0.11 s	8 MB	0.93 s	2 MB	0.04 s	9 MB	0.58 s	2 MB
ca-HepPh	0.10 s	14 MB	1.36 s	3 MB	0.08 s	18 MB	1.71 s	3 MB
soc-Epinions1	0.44 s	38 MB	3.29 s	10 MB	0.37 s	44 MB	3.40 s	8 MB
soc-Slashdot0922	0.53 s	58 MB	4.86 s	13 MB	0.48 s	68 MB	5.27 s	9 MB
web-NotreDame	1.71 s	122 MB	9.54 s	28 MB	1.67 s	129 MB	10.00 s	28 MB
ego-Twitter	0.85 s	104 MB	10.51 s	14 MB	0.65 s	112 MB	10.77 s	9 MB
loc-Gowalla	1.29 s	139 MB	11.35 s	26 MB	1.26 s	130 MB	12.09 s	19 MB
web-Stanford	1.96 s	157 MB	15.54 s	26 MB	2.03 s	157 MB	14.83 s	26 MB
wiki-Talk	37.89 s	603 MB	60.78 s	270 MB	38.76 s	545 MB	56.35 s	217 MB
web-Google	9.55 s	344 MB	34.56 s	75 MB	10.18 s	344 MB	34.81 s	78 MB
com-Youtube	13.68 s	452 MB	39.18 s	147 MB	14.48 s	430 MB	40.10 s	100 MB
web-BerkStan	6.30 s	401 MB	49.23 s	70 MB	6.56 s	408 MB	46.47 s	63 MB
higgs-twitter	7.68 s	562 MB	87.83 s	85 MB	4.49 s	666 MB	87.11 s	41 MB
soc-Pokec	27.12 s	1,280 MB	188.94 s	237 MB	23.32 s	1,448 MB	201.95 s	148 MB
soc-LiveJournal1	89.33 s	2,965 MB	452.94 s	676 MB	84.60 s	4,265 MB	492.49 s	430 MB
com-Orkut	112.78 s	6,292 MB	1,326.51 s	527 MB	76.65 s	10,047 MB	1,469.62 s	284 MB
twitter-2010	1618.74 s	50,789 MB	10,419.08 s	5,629 MB	1,076.50 s	72,567 MB	9,955.61 s	3,403 MB
com-Friendster	3537.57 s	101,429 MB	22,659.64 s	7,711 MB	2,293.77 s	158,596 MB	28,065.90 s	5,203 MB
uk-2007-05	3103.58 s	136,572 MB	27,029.84 s	10,778 MB	2,491.07 s	165,428 MB	27,232.59 s	8,168 MB
ameblo	OOM	OOM	27,017.30 s	27,479 MB	OOM	OOM	6,860.45 s	20,488 MB

Table C.2: Effect of the proposed algorithm on graph size under UC$_{0.1}$ and IWC. V and E denote vertex and edge sets of an input graph, and W and F denote vertex and edge sets of a coarsened graph, respectively.

	uniform cascade (UC$_{0.1}$)				**in-degree weighted cascade** (IWC)			
dataset	$\lvert W\rvert$	$\lvert W\rvert/\lvert V\rvert$	$\lvert F\rvert$	$\lvert F\rvert/\lvert E\rvert$	$\lvert W\rvert$	$\lvert W\rvert/\lvert V\rvert$	$\lvert F\rvert$	$\lvert F\rvert/\lvert E\rvert$
ca-GrQc	5,178	98.8%	24,876	85.9%	5,065	96.6%	28,614	98.8%
ca-HepTh	9,868	99.9%	51,534	99.2%	9,613	97.3%	51,418	99.0%
wiki-Vote	6,973	98.0%	74,975	72.3%	7,115	100.0%	103,689	100.0%
ca-HepPh	10,730	89.4%	76,680	32.4%	11,848	98.7%	236,658	99.9%
soc-Epinions1	74,052	97.6%	229,389	45.1%	75,520	99.5%	508,066	99.8%
soc-Slashdot0922	78,360	95.4%	317,110	36.4%	82,168	100.0%	870,161	100.0%
web-NotreDame	321,484	98.7%	1,076,829	73.3%	325,628	100.0%	1,469,062	100.0%
ego-Twitter	73,257	90.1%	726,196	41.1%	81,306	100.0%	1,768,135	100.0%
loc-Gowalla	189,923	96.6%	1,039,774	54.7%	196,591	100.0%	1,900,654	100.0%
web-Stanford	281,675	99.9%	2,228,430	96.4%	281,498	99.9%	2,311,315	99.9%
wiki-Talk	2,389,283	99.8%	3,102,167	61.8%	2,394,284	100.0%	5,021,203	100.0%
web-Google	875,682	100.0%	5,100,102	99.9%	873,881	99.8%	5,095,502	99.8%
com-Youtube	1,121,282	98.8%	3,460,138	57.9%	1,134,888	100.0%	5,975,244	100.0%
web-BerkStan	679,808	99.2%	6,517,817	85.8%	684,441	99.9%	7,598,639	100.0%
higgs-twitter	409,250	89.6%	4,367,703	29.4%	453,623	99.3%	14,846,862	99.9%
soc-Pokec	1,464,493	89.7%	13,759,924	44.9%	1,632,048	100.0%	30,620,701	100.0%
soc-LiveJournal1	4,514,953	93.1%	29,562,747	43.2%	4,838,026	99.8%	68,453,373	100.0%
com-Orkut	1,359,950	44.3%	8,884,020	3.8%	3,072,441	100.0%	234,370,166	100.0%
twitter-2010	38,952,357	93.5%	359,675,952	24.5%	41,597,759	99.9%	1,468,218,916	100.0%
com-Friendster	47,022,949	71.7%	175,222,222	4.9%	65,608,362	100.0%	3,612,134,262	100.0%
uk-2007-05	102,506,082	97.4%	1,582,197,218	42.6%	105,197,183	100.0%	3,717,097,320	100.0%
ameblo	271,134,723	99.4%	5,489,262,364	79.4%	269,812,560	98.9%	6,837,663,021	98.9%

Table C.3: Average influence estimation time for plain *MC* and our framework with *MC* under $\textsc{uc}_{0.1}$ and IWC. MARE and RCC stand for “mean absolute relative error” and “rank correlation coefficient,” respectively.

	uniform cascade ($\textsc{uc}_{0.1}$)					**in-degree weighted cascade** (IWC)				
	run time			**accuracy**		**run time**			**accuracy**	
dataset	*MC*	Alg.7.4(*MC*)	$\frac{\text{Alg.7.4}(MC)}{MC}$	MARE	RCC	*MC*	Alg.7.4(*MC*)	$\frac{\text{Alg.7.4}(MC)}{MC}$	MARE	RCC
ca-HepPh	9.6 s	2.2 s	22.6%	0.0128	0.9992	84.6 ms	87.8 ms	103.8%	0.0100	0.9997
soc-Slashdot0922	35.2 s	8.7 s	24.6%	0.0106	0.9953	27.4 ms	25.9 ms	94.6%	0.0000	1.0000
web-NotreDame	207.4 ms	51.6 ms	24.9%	0.0136	0.9928	9.8 ms	7.7 ms	78.7%	0.0087	0.9985
wiki-Talk	10.1 s	6.1 s	60.5%	0.0006	0.9999	5.2 ms	4.5 ms	87.9%	0.0088	1.0000
com-Youtube	117.4 s	50.0 s	42.6%	0.0277	0.9907	12.9 ms	12.7 ms	98.9%	0.0626	0.8600
higgs-twitter	1,124.0 s	252.9 s	22.5%	0.0103	0.9951	104.3 ms	90.2 ms	86.5%	0.0302	0.9991
soc-Pokec	2,926.2 s	940.0 s	32.1%	0.0107	0.9974	86.9 ms	86.0 ms	99.0%	0.0279	0.9995
soc-LiveJournal1	4,293.8 s	1,627.2 s	37.9%	0.0125	0.9993	45.6 ms	52.5 ms	115.0%	0.0271	0.9988
com-Orkut	33,771.9 s	1,227.5 s	3.6%	0.0069	0.8915	720.2 ms	694.1 ms	96.4%	0.0000	1.0000
twitter-2010	105,743.0 s	21,889.5 s	20.7%	–	–	42.9 ms	44.7 ms	104.2%	–	–
com-Friendster	450,452.0 s	18,145.4 s	4.0%	–	–	583.2 ms	583.7 ms	100.1%	–	–
uk-2007-05	4,555.2 s	1,813.6 s	39.8%	–	–	56.4 ms	57.7 ms	102.2%	–	–

Table C.4: Run time for selecting a seed set of size 100 and solution quality for plain *D-SSA* and our framework with *D-SSA* under $\textsc{uc}_{0.1}$ and $\textsc{iwc}$. OOM denotes "out of memory."

	uniform cascade ($\textsc{uc}_{0.1}$)					**in-degree weighted cascade** ($\textsc{iwc}$)				
	run time			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\lvert V\rvert$)		**run time**			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\lvert V\rvert$)	
dataset	*D-SSA*	Alg.7.5(*D-SSA*)	$\frac{\text{Alg.7.5}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.7.5(*D-SSA*)	*D-SSA*	Alg.7.5(*D-SSA*)	$\frac{\text{Alg.7.5}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.7.5(*D-SSA*)
ca-HepPh	51.7 s	15.3 s	29.5%	0.3528	0.3525	1.0 s	1.0 s	104.6%	0.1826	0.1827
soc-Slashdot0922	291.9 s	84.3 s	28.9%	0.2958	0.2959	1.2 s	1.2 s	97.7%	0.2653	0.2653
web-NotreDame	3.3 s	1.1 s	33.0%	0.0827	0.0828	0.5 s	0.5 s	99.1%	0.1232	0.1233
wiki-Talk	266.1 s	94.9 s	35.6%	0.1394	0.1394	2.3 s	2.3 s	98.5%	0.4038	0.4041
com-Youtube	2,803.1 s	1,860.2 s	66.4%	0.1510	0.1511	2.4 s	2.4 s	99.4%	0.1727	0.1728
higgs-twitter	3,837.2 s	1,317.1 s	34.3%	0.3511	0.3511	6.7 s	6.9 s	102.8%	0.0654	0.0654
soc-Pokec	17,427.8 s	7,924.7 s	45.5%	0.4741	0.4741	7.5 s	7.5 s	99.9%	0.0528	0.0527
soc-LiveJournal1	OOM	OOM	–%	–	–	25.0 s	26.0 s	104.2%	0.0222	0.0222
com-Orkut	OOM	OOM	–%	–	–	57.6 s	58.3 s	101.3%	0.0667	0.0667
twitter-2010	OOM	OOM	–%	–	–	50.5 s	80.2 s	158.8%	0.2582	0.2586
com-Friendster	OOM	OOM	–%	–	–	5,297.2 s	5,220.4 s	98.5%	0.0040	0.0040
uk-2007-05	OOM	OOM	–%	–	–	81.9 s	92.7 s	113.2%	0.0350	0.0351