Real-Time Targeted-Influence Queries over Large Graphs

Alessandro Epasto★
Google
New York City, NY, USA
Email: aepasto@google.com

Ahmad Mahmoody★
Brown University
Providence, RI, USA
Email: ahmad@cs.brown.edu

Eli Upfal
Brown University
Providence, RI, USA
Email: eli@cs.brown.edu

***Abstract*—Social networks are important communication and information media. Individuals in a social network share information and influence each other through their social connections. Understanding social influence and information diffusion is a fundamental research endeavor and it has important applications in online social advertising and viral marketing.**

In this work, we introduce the *Targeted-Influence* problem (TIP): Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a model of influence, we want to be able to estimate in *real-time* (e.g. a few seconds per query) the influence of a subset of users S over another subset of users T, for any possible query $(S;T)$, $S, T \subseteq \mathcal{V}$. To do so, we allow an efficient preprocessing.

We provide the first scalable real-time algorithm for TIP. Our algorithm requires $\tilde{O}(|\mathcal{V}| + |\mathcal{E}|)$ space and preprocessing time, and it provides a provable approximation of the influence of S over T, for every subsets of nodes $S, T \subseteq \mathcal{V}$ in the query with large enough influence. The running time for answering each query (a.k.a query stage) is theoretically guaranteed to be $\tilde{O}(|S| + |T|)$ in general undirected and for directed graphs under certain assumptions, supported by experiments.

We also introduce the *Snapshot* model as our model of influence, which extends and includes as special case both the Independent Cascade and the Linear Threshold models. The analysis and the theoretical guarantees of our algorithms hold under the more general Snapshot model. Finally, we perform an extensive experimental analysis, demonstrating the accuracy, efficiency, and scalability of our methods.

I. Introduction

Online social networks allow their users to connect, share information, and interact with each other. These interactions between connected users can cause the members of the networks to be influenced by one another. For instance, a rumor can spread through Facebook as a result of being re-shared, a piece of news can reach a large audience by being re-tweeted multiple times, a user can adopt a new product after seeing it reviewed by his friends, etc. In these scenarios the members of the network are influenced (i.e. they adopted the product or they received the news) via a *cascade* of pairwise interactions which might extend far from the source of the influence. Understanding the phenomena of social influence and information propagation in networks is a fundamental research endeavor and as such it has been subject to many studies including works on identifying the influential members of a network [3], [9], [11], [12], [22], [23], [25]. Understanding the social influence has also important applications in the context of online advertisement and viral marketing.

In this paper we focus on the problem of estimating the influence exerted by a group of users over another target group of users. This is an important primitive in the context of online social advertisement. Social networks' advertisement campaigns, in fact, are often targeted: The advertiser has a specific group of users in mind that we refer to as the *target set*, such as users in a certain geographic area, in a social group, of a specific occupation, or at an income level. However, direct communication from the advertiser to the entire target set may not be possible and may not be as effective as a third person recommendation. For instance, the advertiser might only contact directly the users that already liked her page (or signed-up to a mailing list) and may want to influence a larger population. Therefore, in viral marketing, the goal of the advertiser is to spend her budget, e.g. by free samples or sending coupons, over some influential users of the network that may recommend her product to their friends/followers, with the hope that the product is adopted by a large number of users in her target set through the cascade of recommendations. We refer to the set of users over which the advertiser makes the initial spending as the *seed set*. The advertiser, however, can not spend on the all nodes, and she is restricted to a smaller group of users. Therefore, she needs to estimate the influence of potential seed sets over her target set, in order to decide which seed set to pick. Another potential application of this method is in the realm of online shopping. Here, one wants to estimate the likelihood that a customer visiting a set of products, to end up visiting another set of products by following the similar item recommendation links. This problem can also be modeled as an influence propagation instance.

Despite extensive research in modeling, computing, and optimizing the influence in social networks, most works have focused on the *global* influence, where the target set is the whole network and the advertiser can access any seed set. This is a significant limitation in the context of targeted advertisement where only a potentially small fraction of such users are of interest. Note that, in this context, the computation efficiency is the key since a great number of advertisers operate in on-line advertisement systems [14] and each requires to obtain estimates on multiple query sets in real-time to tune her campaign. For this reason, we design efficient preprocessing algorithms that help us serve these advertisers in reasonable time.

To address this issue more precisely, we introduce the *Targeted-Influence Problem* (TIP). In TIP we are given, in the *preprocessing* stage, an arbitrary graph $\mathcal{G}$ and a model of influence over $\mathcal{G}$. Then, in the *query* phase, a sequence of $(S_i; T_i)$ queries arrive, where S_i and T_i are the set of seed nodes and target nodes, respectively. Our goal in the query stage is to estimate the influence of the set S_i over the set T_i (i.e. the expected number of influenced nodes in T_i if the nodes in S_i spread the information) in (almost) linear time in size of the query, $\tilde{O}(|S_i| + |T_i|)$. To do so as efficiently as possible, we

★ Co-first authors. Sorted Alphabetically.

† This work was supported in part by NSF grant IIS-1247581 and NIH grant R01-CA180776.

ASONAM '17, July 31 - August 03, 2017, Sydney, Australia

ACM ISBN 978-1-4503-4993-2/17/07$15.00
http://dx.doi.org/10.1145/3110025.3110105

are allowed to preprocess the graph $\mathcal{G}$ in the preprocessing stage, *before* knowing which queries will be issued during the query stage. Also, note that having access to a TIP-solver oracle is sufficient to (approximately) find the most influential node among the accessible nodes as in [33]. However, the focus of this work is on *estimating* the targeted-influence and not maximizing it.

To the best of our knowledge, TIP is a new problem and has not been addressed before. Notice that this is a significant departure from recent works [13], [33] that proposed settings for *Influence Maximization* problem in the target context where an expensive computation is run to maximize influence over a given fixed target set. In practical settings however, the social network operator needs to optimize multiple campaigns with different targets and seed sets at the same time. For this reason our aim is instead, to allow high quality answers in the query stage, for many distinct $(S_i; T_i)$ queries after an efficient preprocessing.

Naive approach. In most influence propagation models, the TIP problem can be readily solved by standard Monte Carlo (MC) simulations: simply run (enough) MC simulations of the influence process, and measure how many nodes in T_i are influenced when the information starts propagating from the seed nodes in S_i for each query $(S_i; T_i)$. However, as we discussed above, in real settings numerous advertisers operate on the social network at any given time and each of them may have many ongoing advertisement campaigns with different target sets and potential seed sets to evaluate. Determining the best strategy even for a single campaign might involve performing numerous influence queries. As each influence query requires multiple MC simulations over a potentially huge graph, this naive approach is not able to handle many advertisers over large networks with billions of nodes and edges. For an online targeted-influence query system to be practically useful, we would like to be able to answer such queries in real-time (say in a few seconds per query).

In this paper we present the first real-time algorithm for TIP on large scale graphs. We present algorithms for both directed and undirected graphs: after a feasible preprocessing, our algorithms are guaranteed to answer each $(S; T)$-query in time $\tilde{O}(|S|+|T|)$, for general undirected graphs, and directed graphs under some assumptions, that are backed with experiments[1]. Note that in the naive approach, the running time at the query stage is $\Theta(|V|+|E|)$ required for just a single MC simulation. We also provide precise theoretical guarantees on the approximation error of this algorithm.

Our experiments show that our algorithms are able to answer influence queries quickly with high accuracy. We achieve several orders of magnitude speedups over the naive approach of Monte Carlo simulation while preserving good accuracy in both directed and undirected graphs.

A. Notations and Problem Definition

Throughout this paper, we assume that we are given an arbitrary graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input. To model the influence of a seed set $S \subseteq \mathcal{V}$ over a target set $T \subseteq \mathcal{V}$, we introduce the *Snapshot* model. In Sect. III-E we prove that the Snapshot model generalizes the Triggering model that in turn is a generalization of the classic and widely used Independent-Cascade and Linear-Threshold models [17]. Our analysis holds for this more general model, and thus, is valid for both Independent-Cascade and Linear-Threshold models. Informally, the Snapshot model is represented by an arbitrary distribution $\mathcal{M}$ over the subgraphs of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each sample $G = (\mathcal{V}, E)$ (subgraph of $\mathcal{G}$ with the same set of nodes) from the distribution is a realization of the diffusion process over $\mathcal{G}$, where the edges in E represent the active influence edges in that instance. Given one realization G, a node $u \in S$ influences all the other nodes to which it is connected to by a (directed) path in G, if S is the set of seeds:

[1]Here, $\tilde{O}(\cdot)$ notation hides the lower order terms.

Definition 1 (Snapshot & Influence)**.** *The **Snapshot** model for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is a distribution $\mathcal{M}$ over all subgraphs of $\mathcal{G}$, and by $G \sim \mathcal{M}$ we mean G is a subgraph of $\mathcal{G}$ that is selected randomly via $\mathcal{M}$. We say an edge $e \in \mathcal{E}$ is **activated** in $G = (\mathcal{V}, E) \sim \mathcal{M}$, if $e \in E$. The **influence** of a seed set S over a target set T, denoted by $\mathsf{Inf}(S; T)$, is defined as*

$$\mathsf{Inf}(S; T) = \mathbb{E}_{G\sim\mathcal{M}} \left[|I_G(S) \cap T|\right],$$

where G is a random subgraph of $\mathcal{G}$ sampled according to $\mathcal{M}$ and $I_G(S)$ is the set of all reachable nodes from S in graph G. Therefore, $\mathsf{Inf}(S; T)$ is the expected number of nodes in T that can be reached from S in a random graph sampled via $\mathcal{M}$.

In this paper, we study the problem of estimating the influence of seed sets over target sets in real-time, after a preprocessing of the data to expedite the estimation of the targeted-influence. Our methods for TIP has two stages: (1) preprocessing, and (2) query stages. Obviously, in order to work with large datasets, our solution has to be scalable at the preprocessing stage, and very fast at the query stage. Formally, our goal is as follows:

Definition 2 (Targeted-Influence Problem (TIP))**.** *By preprocessing the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in time $\tilde{O}(|\mathcal{V}| + |\mathcal{E}|)$ and using $\tilde{O}(|V|)$ space, provide an estimation of $\mathsf{Inf}(S; T)$ for every seed and target sets, in time $\tilde{O}(|S| + |T|)$. We call a query of a seed set S and a target set T, an $(S; T)$-**query**.*

Our Contributions. In this paper, (1) we introduce the Targeted-Influence problem, (2) we give efficient solutions to the TIP problem, both for undirected and directed graphs with theoretical guarantees, and (3) we provide extensive experiments on real world networks, and show the efficiency and accuracy of our methods.

Paper Organization. Sect. II reviews the related work. In Sect. III we provide our algorithms and theoretical results for (i) efficient preprocessing, and (ii) estimating the targeted-influence, both for undirected and directed graphs. Finally, in Sect. IV we conclude by applying our proposed algorithms on real world networks.

II. Related work

Kempe *et al.* [17] introduced the Influence Maximization (IM) problem. In the IM problem we want to find a subset of nodes, of a given size, that has the maximum influence over the graph. In our notation this can be defined as $\arg\max_{S:|S|=k} \mathsf{Inf}(S; \mathcal{V})$ for a fixed positive integer k. They provided a constant approximation for both *Independent Cascade* and *Linear Threshold models*. They also introduced the *Triggering* model which is a generalization of the Independent Cascade and Linear Threshold models. Here, we introduce the *Snapshot* model, that is a generalization of Triggering model, and for which our theoretical analysis holds (see Sect. III-E). Later, more efficient algorithms for the IM problem were provided based on sampling (random *hyper-edges*) [2], [29], [30]. These algorithms run in almost linear time. As mentioned before, our goal in this work is to efficiently *estimate* the targeted-influence and not maximizing it.

More related to our work, Zhou *et al.* [33] introduced the *constraint IM* problem, in which for a given positive integer k, a graph $G = (V, E)$, an accessible set $A \subseteq V$, and a target set $T \in V$, one has

to find a set $S \subseteq A$ of size k that has the maximum influence over the set T. The model in [33] generalized a previous model known as *personalized influence* [13], where T is just a single node and $A = V \setminus T$. The algorithm proposed for the constraint IM in [33], is essentially the standard Monte Carlo method followed by the classic greedy algorithm. Our model/problem substantially departs from this model since, (i) we focus on computing the (targeted) influence for several seed and target sets query, and (ii) we consider a real-time setting where we need to answer each query, in the query stage, as fast as possible (i.e $\tilde{O}(|S| + |T|)$). Thus, we cannot afford to run the Monte Carlo method over the entire graph, for each query.

In [21] Lucier *et al.* studied the problem of estimating the influence of a given seed set, and proposed a parallel algorithm for this task. In contrast, our model considers more general influence queries, where we have to find the influence of a seed set over any other given target set. Also, we aim at algorithms that answer each influence query in a time that is linear in the size of the query, not the size of the graph.

Cohen *et al.* in [7], introduced a sketch-based algorithm for computing and maximizing the influence. Although this method can be applied for estimating the influence of a seed set S in the network, i.e. $\mathsf{Inf}(S; \mathcal{V})$, it cannot handle online $(S;T)$-queries for general target set T. This is because such sketch-based method allows to estimate the cardinalities and not the set intersections (a significantly harder problem). A naive use of such sketches would requires to store (at the preprocessing time) a different sketch for each possible distinct target set T, where exponentially many of them are possible.

Other related work are *Topic-Aware model IM* [1], where the diffusion probabilities can be different for different items (for each item, an Independent Cascade model is assumed), and the *Adaptive Seeding*, in which the budget for seeding the nodes is partitioned in different stages. Recently, Subbian *et al.* [27], studied the problem of finding the top-k influential nodes in graph streams. In another work, Cohen *et al.* [6] considered a different influence model based on the distances in the network. The IM problem has been also studied in continuous time diffusion models [10], [24]. In [28], Tang *et al.* considered a slightly different objective function, which is a linear combination of the influence and a *diversity* function, with the goal of maximizing influence and diversity of the influenced nodes at the same time.

Variations of the problem of target influence propagation has been studied recently. Chang et al. [4] introduce a related formulation in a context where users can be activated multiple times. Lee et al cast the problem of maximizing influence over certain users as a query processing problem [19]. Finally Li et al. [20] model a target influence problem where activation probabilities depend on the similarity of the user interest to a topic.

Finally, our problem is closely related to the fundamental problems of *reachability* in directed graphs (RP) [8], [16], [26], [32] and uncertain graphs (RUP) [5], [15], [18]: In RP, each query is a (u, v) pair of nodes, and we have to decide whether u can reach v. The RUP problem, for each (u, v) query, asks the *probability* that u can reach v, where edges can fail (i.e., be removed) with some probability.

Note that each method for RP can be extended to an approximate method for RUP, by generating samples from the graph, and process them individually as *certain* graphs. Also, note that to answer each $(S;T)$-query in our problem, one can naively applies a reachability method on each (s, t) pair, where $s \in S$ and $t \in T$. However, this causes the processing time to be at least $\Omega(\alpha \cdot |S| \cdot |T|)$, where α is the time to answer a reachability query, since there are $|S| \cdot |T|$ number of (s, t) pairs. Another important point to note is that in most reachability methods, either α is at least $\Omega(m)$, or the construction time is $O(mn)$, where n and m are the number of nodes and edges, respectively. Our other hand, we seek a practical approach that would scale to large graphs, so we want the running time in the query stage to be (almost) linear in size of the query, i.e, $\tilde{O}(|S| + |T|)$.

In an independent work, similar to our method [26] applied the Bloom filter method for answering reachability queries. However, our Bloom filter method is equipped a *hub* technique that significantly improves the space needed by the Bloom filter to obtain good results as we show in the experiments.

III. Method

We start this section by discussing how we can estimate the targeted-influences in general settings, using Monte Carlo methods. We then, introduce our algorithms.

A. Estimating Targeted-Influence Using Monte Carlo Methods

Recall that we assume the Snapshot model and denote by $\mathcal{M}$ its distribution. Also, recall that by an $(S;T)$-query we mean the query for a pair of seed set S and a target set T. By definition, the influence of S over T is $\mathsf{Inf}(S;T) = \mathbb{E}_{G\sim\mathcal{M}}[|I_G(S) \cap T|]$ which is an expectation that can be approximated using Monte Carlo simulation: given a sample of ℓ graphs $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ sampled independently from $\mathcal{M}$, the (experimental) influence of S over T according to $\mathcal{S}$ is

$$\mathsf{Inf}_{\mathcal{S}}(S;T) = \frac{1}{\ell}\sum_{i=1}^{\ell} |I_{G_i}(S) \cap T|.$$

Obviously, we have $\mathbb{E}_{\mathcal{S}}[\mathsf{Inf}_{\mathcal{S}}(S;T)] = \mathsf{Inf}(S;T)$.

In order to quantify the accuracy of an estimation for an $(S;T)$-query for a given sample $\mathcal{S}$ we introduce the following definition.

Definition 3 (Impact)**.** *The **impact** of a seed set S over a target set T is the average fraction of the number of nodes in T that get influenced by S, i.e., $\frac{\mathsf{Inf}(S;T)}{|T|}$. We denote the impact of S over T by $\rho(S;T)$.*

Now, the following theorem states the sample complexity for estimating the influence of an $(S;T)$-query.

Theorem 1. *Suppose $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ is a sample of ℓ independent draws from $\mathcal{M}$. If $\ell \geq \frac{\ln(2/\delta)}{2\epsilon^2\rho^2}$, where $\epsilon, \delta \in (0,1)$ and $\rho = \rho(S;T)$, with probability at least $1-\delta$ we have*

$$|\mathit{Inf}(S;T) - \mathit{Inf}_{\mathcal{S}}(S;T)| < \epsilon \cdot \mathit{Inf}(S;T).$$

Proof. For $i \in \{1, \ldots, \ell\}$, let $X_i = |I_{G_i}(S) \cap T|$. Obviously , $\mathsf{Inf}_{\mathcal{S}}(S;T) = \frac{X_1+\ldots+X_\ell}{\ell}$, $0 \leq X_i \leq |T|$, and $\mathbb{E}[X_i] = \mathsf{Inf}(S;T)$. Therefore, by Hoeffding's inequality we have

$$\begin{aligned}\mathsf{Pr}[|\mathsf{Inf}_{\mathcal{S}}(S;T) - \mathsf{Inf}(S;T)| \geq \epsilon \cdot \mathsf{Inf}(S;T)] \\ \leq 2 \cdot \exp\left(-\frac{2\epsilon^2 \mathsf{Inf}(S;T)^2\,\ell}{|T|^2}\right),\end{aligned}$$

and by substituting for ℓ the proof is complete. □

Notice that in practice, the queried seed sets are expected to be close to the target set, if not directly connected. Therefore, if we restrict the queries to those with an impact greater than a constant threshold, the sample complexity for computing the influence will be $O(\ln(1/\delta)/\epsilon^2)$. Thus a **constant** number of samples suffices for any constant probability and error parameter.

B. Intuition of Our Method

We have shown that to answer any $(S;T)$-query, it is sufficient to draw a sample $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ according to $\mathcal{M}$ and then efficiently compute $|I_{G_i}(S) \cap T|$. In particular, for any graph we can obtain a good approximation for queries with at least constant impact with $O(\ln(1/\delta)/\epsilon^2)$ samples. Moreover, for $O(\ln(|V|)/\epsilon^2)$ samples we can get a $(1+\epsilon)$-approximation, with high probably, for all queries in a polynomially long sequence of queries. Notice that computing $|I_{G_i}(S) \cap T|$ at the query phase might take $\tilde{O}(|V| + |E|)$ time, as opposed to $\tilde{O}(|S| + |T|)$. To circumvent this issue, we show how one can preprocess each random graph $G_i \in \mathcal{S}$ during the preprocessing stage so that at the query time we can estimate $|I_{G_i}(S) \cap T|$ efficiently and accurately. To do so, we present different algorithms for preprocessing and answering the queries for undirected and directed graphs. Finally, note that even for queries with small impact, we have the following immediate corollary, using Hoeffding's inequality:

Corollary 2. *Suppose $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ is a sample of ℓ independent draws from $\mathcal{M}$. If $\ell \geq \frac{\ln(2/\delta)}{2\epsilon^2}$, where $\epsilon, \delta \in (0,1)$, with probability at least $1-\delta$ we have*

$$|\mathsf{Inf}(S;T) - \mathsf{Inf}_{\mathcal{S}}(S;T)| < \epsilon \cdot |T|.$$

C. Undirected Graphs

In this section, we assume $\mathcal{G}$ is an undirected graph [2], and $G \sim \mathcal{M}$ is one random subgraph of $\mathcal{M}$ that needs to be preprocessed. We first describe the algorithm, and then provide its analysis.

1) Algorithm: We preprocess G and compute $|I_G(S) \cap T|$ as following:

Preprocessing. We obtain the list of connected components of G and for each node u we store the id of its connected component that we denote by $\mathtt{cc}(u)$.

Computing $|I_G(S) \cap T|$. Suppose we are given the $(S;T)$-query to process. Let $\mathtt{cc}(S) = \{\mathtt{cc}(u) \mid u \in S\}$. Note that

$$|I_G(S) \cap T| = |\{v \in T \mid \mathtt{cc}(v) \in \mathtt{cc}(S)\}|.$$

This is because S reaches to a node $v \in T$ if and only if there is a node $u \in S$ such that u, v are in the same connected component. Therefore, $|I_G(S) \cap T|$ can be computed in time $O(\min|S|, |T|)$ by using hash-maps, or in time $O(\log(|S|)|S| + \log(|T|)|T|)$ by sorting and merging the lists).

2) Analysis: Now, we analyze the running time of our algorithm for undirected graphs, when we sample $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ of randomly chosen subgraphs $G_i \sim \mathcal{M}$.

Preprocessing: The preprocessing time is due to (i) the time to sample a random subgraph $G = (V, E)$ according to $\mathcal{M}$ (ii) finding the connected components in time $O(|V| + |E|)$. Therefore, the preprocessing algorithm can be implemented in total time $O(\ell[|\mathcal{V}| + |\mathcal{E}| + T_{\mathcal{M}}])$, where $T_{\mathcal{M}}$ is the time to generate a sample from $\mathcal{M}$. Note that our preprocessing stage requires $O(\ell \log(|\mathcal{V}|))$ bits per node, just to store the id of the connected component of the node in each sample. Also, if $\mathcal{M}$ is the Independent-Cascade model, $T_{\mathcal{M}} = |\mathcal{E}|$. Notice that the preprocessing algorithm can be easily parallelized in $O(diam(\mathcal{G}))$ rounds of Map-Reduce, in which the communication complexity of each round is $O(\ell(|\mathcal{V}| + |\mathcal{E}|))$, and each reducer receives an input of size $O(\ell\Delta)$, where Δ is the maximum degree in $\mathcal{G}$.

[2]Equivalently in Independent-Cascade model, each pair of nodes in an edge, have the same probability of activating it.

Answering queries: To answer each $(S;T)$-query, we need to compute the average of $|I_{G_i}(S) \cap T|$, over the sampled subgraphs, and thus, the running time for answering the queries is $O(\ell \cdot \min\{|S|, |T|\})$ by using hash-maps, or $O(\ell(\log(|S|)|S| + \log(|T|)|T|))$ by sorting and list merging as above.

Considering our analysis, our algorithm satisfies the requirements of the TIP problem, and is an accurate and efficient solution.

D. Directed Graphs

Our algorithms for directed graphs follow the same approach of the undirected one, but here the problem is complicated by the need to compute reachability in directed graphs. For this case we provide some heuristic algorithms that allow a good experimental tradeoff between the space and time required in the preprocessing phase and the precision at query time. We present our basic technique (Algorithms 1 and 2) for preprocessing and query. Later improve these algorithms with additional technique.

Here, we assume $\mathcal{G}$ is a directed graph, and we provide an efficient way to preprocess each randomly sampled subgraph $G = (\mathcal{V}, E) \sim \mathcal{M}$ and estimate $|I_G(S) \cap T|$ for any $(S;T)$-query.

1) Algorithm: Our method for both preprocessing and query stages are provided in Algorithms 1 and 2, and here we give an overview of these algorithms.

Preprocessing. We apply the *Bloom filter* technique (defined below), which assigns a *reachability vector* (defined below) $R_G(u)$ of length w to each node $u \in \mathcal{V}$ (see Algorithm 1).

Computing $|I_G(S) \cap T|$. Using these vectors, for each pair u, v we can estimate whether $v \in I_G(u)$, i.e., v is reachable from u or not (see Algorithm 2), which provides us a biased estimator for $\mathsf{Inf}(S;T)$.

Without loss of generality, assume $\mathcal{V} = [n]$[3]. For the directed graph G, let $\mathcal{C}(G) = \{C_1, \ldots, C_r\}$ be the partition of G into its strongly connected components (SCC)[4]. Also, for each node $u \in [n]$ we denote the strongly connected component that includes u by $\mathcal{C}(u)$. Note that by grouping the nodes in each C_i, we obtain a directed-acyclic-graph (DAG), denoted by $G[\mathcal{C}]$, whose nodes are the strongly connected components in $\mathcal{C}(G)$, and C_i has an edge to C_j if there is a node in C_i that connects to a node in C_j via an edge in E. Also, without loss of generality, we always assume $C_1, \ldots, C_r$ is the topological order of C_i's according to $G[\mathcal{C}]$ (i.e. if there is a direct path from C_i to C_j with $i \neq j$ then $i < j$). We also denote the in-neighbors of C_i in $G[\mathcal{C}]$ by $N^-(C_i)$.

Definition 4 (Reachability Vector)**.** *Suppose k and w are positive integers, and let $\mathcal{H}_{k,w} = \{h_1, \ldots, h_k\}$ be a family of k independent random hash functions, such that for $i \in \{1, \ldots, k\}$ we have $h_i : \mathcal{C}(G) \to [w]$. For $C \in \mathcal{C}(G)$ define $\mathcal{H}_{k,w}(C)$ as a binary vector of length w whose i-th entry is 1 if and only if for some j we have $h_j(C) = i$. Now, the **reachability vector** of node u, denoted by $R_G(u)$ is*

$$R_G(u) = \bigvee_{v \in I_G(u)} \mathcal{H}_{k,w}(\mathcal{C}(v)),$$

where $\vee$ is the bit-wise logical OR operation.

Note that reachability vectors are in fact Bloom filters that verify if a strongly connected component is reachable from a given node. We later describe the tradeoff between the values of k and w and the precision of the algorithm (See in Lemma 3 and Theorem 4). Finally,

[3]We denote the set $\{1, \ldots, n\}$ by $[n]$.

[4]Note that finding the partition $\mathcal{C}(G)$ can be done efficiently in time $O(|V| + |E|)$ using Tarjan's algorithm [31].

in Algorithm 2 we show how we can estimate $|I_G(S) \cap T|$ for an $(S;T)$-query. For notational convenience in Algorithm 2, we denote $\boldsymbol{x} \leq \boldsymbol{y}$ for two real vectors of the same length, if $\boldsymbol{x}$ is smaller than $\boldsymbol{y}$ entry-wise.

Algorithm 1: `PreD`(G, k, w)

input : Directed graph G, positive integers k and w.
output: Hash functions and reachability vectors.

begin
  $\mathcal{C}(G) = \{C_1, \ldots, C_r\}$ `//SCC, ordered topologically`
  $\mathcal{H} \leftarrow \mathcal{H}_{k,w}$ `//Sample random hash functions`
  for $i \leftarrow 1$ **to** r **do**
    $B(C_i) \leftarrow \mathcal{H}(C_i)$
  end
  for $j \leftarrow r$ **to** 1 **do**
    for $C_i \in N^-(C_j)$ **do**
      $B(C_i) \leftarrow B(C_i) \vee B(C_j)$
    end
  end
  for $u \leftarrow 1$ **to** n **do**
    $R_G(u) \leftarrow B(\mathcal{C}(u))$
  end
  return $\mathcal{H}$ *and* $(R_G(u))_{u\in[n]}$
end

2) Analysis: In this section we analyze our method for estimating $|I_G(S)\cap T|$ for an $(S;T)$-query. Let $\alpha(S)$ be the number of reachable strongly connected components from the nodes in S, i.e., $\alpha(S) = \{i \mid S \rightsquigarrow C_i\}$, where by $\rightsquigarrow$ we mean existence of a path from S to C_i in G. Similarly, we use $S \rightsquigarrow v$ for a node v if v is reachable from a node in S. We first have the following lemma:

Lemma 3. *Consider U and $\mathcal{H}$ as defined in Algorithm 2. If $w > \frac{\log(1/\delta)}{\log(1/0.7865)}\alpha(S)$ and $k = \frac{\log(2)w}{2\alpha(S)}$, for $\delta \in (0,1)$, and v is a node in G:*

(a) if $S \rightsquigarrow v$ then $\mathcal{H}(\mathcal{C}(v)) \leq U$; and
(b) if $S \not\rightsquigarrow v$, with probability at least $1-\delta$, $\mathcal{H}(\mathcal{C}(v)) \not\leq U$.

Proof. For part (a), note that if $S \rightsquigarrow v$, then there exists a node $u \in S$ such that $u \rightsquigarrow v$. First, $\mathcal{H}(\mathcal{C}(v)) \leq R_G(u)$, since $R_G(u)$ is the logical `OR` of some vectors including $\mathcal{H}(\mathcal{C}(v))$ in Algorithm 1. Secondly, $R_G(u) \leq U$ as U is the logical `OR` of some vectors including $R_G(u)$ in Algorithm 2. So, $\mathcal{H}(\mathcal{C}(v)) \leq U$.

For part (b), note that the probability that a single bit being zero in U is $(1-1/w)^{k\alpha(S)}$, since $\mathcal{H}$ is a family of k-independent functions. Therefore, the probability of a false positive (i.e. $\mathcal{H}(\mathcal{C}(v)) \leq U$) is

$$\begin{aligned}\mathsf{Pr}\left[\mathcal{H}(\mathcal{C}(v)) \leq U\right] &= \left(1 - \left(1 - \frac{1}{w}\right)^{k\alpha(S)}\right)^k \\ &\leq \left(1 - e^{-\frac{2k\alpha(S)}{w}}\right)^k \leq \left(1 - \frac{1}{2}\right)^k \leq (0.7865)^{\frac{w}{\alpha(S)}} < \delta,\end{aligned}$$

where we used the fact that $1 - x \geq e^{-2x}$ for $x \leq \frac{1}{2}$. □

Lemma 3, gives us the next Theorem:

Theorem 4. *Consider X and U as defined in Algorithm 2, and an $(S;T)$-query. If $k \geq \log_2(|T|/\delta)$ and $w \geq 2\alpha(S)$, for $\delta \in (0,1)$, we have*

(a) With probability at least $1-\delta$, $X = |I_G(S) \cap T|$; and
(b) Always

$$|I_G(S) \cap T| \leq \mathbb{E}\left[X\right] \leq \left(1 + \frac{\delta}{\rho(S;T)}\right) \cdot |I_G(S) \cap T|.$$

Recall that $\rho(S;T)$ is the impact of S over T.

Proof. Part (a) is directly obtained from Lemma 3, and applying the union bound. For part (b) notice that

$$|I_G(S) \cap T| \leq X \leq T,$$

since (i) for every $v \in |I_G(S) \cap T|$ we have $\mathcal{H}(\mathcal{C}(v)) \leq U$, and thus, X is increased at least $|I_G(S) \cap T|$ times; and (ii) X is increased at most T times. Therefore, $|I_G(S) \cap T| \leq \mathbb{E}\left[X\right]$, and

$$\begin{aligned}\mathbb{E}\left[X\right] &\leq \mathsf{Pr}\left[X = |I_G(S) \cap T|\right] \cdot |I_G(S) \cap T| \\ &\quad + \mathsf{Pr}\left[X > |I_G(S) \cap T|\right] \cdot |T| \\ &\leq |I_G(S) \cap T| + \delta \frac{|I_G(S) \cap T|}{\rho(S;T)} \\ &= \left(1 + \frac{\delta}{\rho(S;T)}\right) \cdot |I_G(S) \cap T|,\end{aligned}$$

where we used the fact that $\mathsf{Pr}\left[X = |I_G(S) \cap T|\right] \leq 1$, and the proof is complete. □

Algorithm 2: `QueD`$(S, T, \mathcal{H}, (R_G(u))_{u\in[n]})$

input : Seed set S, target set T, hash functions $\mathcal{H}$, and reachability vectors $(R_G(u))_{u\in[n]}$.
output: An estimate of $|I_G(S) \cap T|$.

begin
  $U \leftarrow \bigvee_{u\in S} R_G(u)$
  $X \leftarrow 0$
  for $v \in T$ **do**
    if $\mathcal{H}(\mathcal{C}(v)) \leq U$ **then**
      $X \leftarrow X + 1$
    end
  end
  return X
end

Theorem 4 shows that the number of bits for each node, sample pair needed to answer accurately a query is $w \geq 2\alpha(S)$. Notice that although the seed sets are in practice small sets, the $\alpha(S)$ can be a large quantity.

To reduce the memory required we introduce the following heuristic. In the preprocessing stage, we select λ strongly connected components (SCCs) with the highest out-degree in the $G[\mathcal{C}]$ DAG, i.e., SCCs with maximum number of SCCs they can reach with a directed edge. We let L to be the set of these SCCs, and call the elements of L, the **hubs** of the network. Thus, $|L| = \lambda$. We set λ to be a small constant, and in our experiments we used $\lambda = 10$.

Next, for each hub $l \in L$ (which is also a SCC) we execute two visits of the graph to obtain: 1) all the SCCs reached by l and 2) all the SCCs reaching l. The ids of the SCCs reached by each $l' \in L$ are stored in a distinct hash set for constant query time. For each node $v \in V$, instead, we store a bitmap of λ bits representing whether the node reaches any given hub.

We finish the preprocessing stage as before (similar to Algorithm 1) but the accumulations of $B(C_i)$'s vectors stop at the components in L (which are essentially ignored from the graph).

The query algorithm, instead, proceeds as before with the following difference: we first obtain the union of all hubs reached by any nodes in S. Then in order to see whether $S \rightsquigarrow v$, for a node $v \in T$, we both check (i) $\mathcal{H}(\mathcal{C}(v)) \leq U$, and (ii) if for any hub reached by S, v belong to the set reached by that hub.

It is easy to see that using the hub trick can only decrease the false positives caused by the Bloom filter as reachability through hubs is computed exactly. It is also clear that the running time for preprocessing and query algorithms become $\tilde{O}(\lambda(|\mathcal{V}| + |E|))$ and $\tilde{O}(\lambda(|S| + |T|))$, respectively, and since λ is a constant we obtain the same complexity as before.

Finally, it is worth noting that this trick of using hubs, significantly improves the space needed by the Bloom filters to obtain good results as we show in the experiments. This is due to the skewed reachability set size distribution of SCCs: not storing the descendants reachable from the top SSCs significantly reduce the maximum size that needs to be stored in any Bloom filter as our experiments show.

Therefore, for a general directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we preprocess the graph and answer the queries as follows:

- `Preprocess`: generate a sample of $\mathcal{S} = \{G_1, \ldots, G_\ell\}$, where $G \sim \mathcal{M}$. Then, preprocess each G_i using the hub heuristic (for better performance).
- `Query`: For an $(S;T)$-query, return $\frac{1}{\ell}\sum_{i=1}^{\ell} |I_{G_i}(S) \cap T|$, using the hub heuristic.

Finally we conclude this section, by computing the running time and the space complexity of our approach.

Running time and space complexity: The preprocessing algorithm can be implemented in total time $O(\ell(kw + \lambda) \cdot (|\mathcal{V}| + |E|))$. The space for each of ℓ samples is $O(wr + |V|(\log_2(r) + \lambda))$ where r is the number of SCCs in the sample graph. Finally the query time is $O(|S|\ell w d\lambda + \ell k\lambda(|T|))$.

E. On Snapshot Model

In this short section, we explain why the Snapshot model generalizes the Triggering model [17] and thus the independent Cascade and the Linear Threshold model. We first start by formally defining the Triggering model:

Definition 5 (Triggering model [17])**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, and for each node v let $N(v)$ be the set of v's neighbors. In the* Triggering *model, each node v is assigned with a distribution T_v over subsets of $N(v)$, and when v is influenced, it influences a subset of its neighbors sampled according to T_v, independent of other nodes. We denote a Triggering model by the vector $(T_v)_{v\in\mathcal{V}}$.*

Note that in the Triggering model, the process in which a node v chooses a subset N' of its neighbors to influence, is equivalent to sampling the edges that connect v to the nodes in N'. Therefore, the triggering model $(T_v)_{v\in\mathcal{V}}$ imposes a distribution $\mathcal{M}$ over the subsets of $\mathcal{G}$, where each subset is obtained from $\mathcal{G}$ by sampling the edges according to the joint distribution of $(T_v)_{v\in\mathcal{V}}$, independently. Therefore, clearly, our model is more general than the Triggering model as it allows for arbitrary (and hence non-independent) joint distributions for the neighbors of each node.

IV. Experiments

In this section we provide our experimental results. First, we introduce our datasets and the influence model that we use for the experiments and discuss some important properties for these datasets in the given model. Next, we evaluate the efficiency of the preprocessing algorithms and show the speed-up we obtain for query stage with respect to using the naive Monte Carlo method for each given $(S;T)$-query. Finally, we measure the accuracy of our estimation method. Each experiment was executed using a single machine with Intel Xeon cpu at 2.83GHz and with 12GB RAM.

A. Data and Model

For sake of experiments, we consider the widely used Independent Cascade model, which we showed is a special case of the Snapshot model. We assume that the activation probability of each edge equal to p. We show the results for $p \in \{0.05, 0.1, 0.2\}$.

In Sect. III, we presented 2 approaches: one for undirected, and one for directed graphs. We refer to $\mathcal{A}_U$ as the undirected graph algorithm of Sect. III-C. This algorithm is very efficient and all the experiments using $\mathcal{A}_U$ are run in the main memory.

We refer to $\mathcal{A}_D$ as the directed graph algorithm using hubs (Sect. III-D). This algorithm is more expensive compared to $\mathcal{A}_U$ for the complexity of handling reachability in directed graphs. In this case for our larger graphs (com-Youtube, wiki-Talk, soc-Pokec, soc-LiveJournal1) we store the preprocessed data output in files, and use them to answer the queries.

a) Datasets: We used publicly available datasets[5]. See Table I. The direction of each edge represents the flow of the influence. We also remove any self-loop or parallel-edge.

Datasets	Direction	#nodes	#edges
email-Enron	U	36692	183831
soc-Epinions1	D	75878	508837
email-EuAll	D	265006	418956
soc-Slashdot0902	D	82168	870161
com-dblp	U	317080	1049866
ego-twitter	D	81306	1768135
com-youtube	U	1134890	2987624
wiki-Talk	D	2394367	5021410
soc-Pokec	D	1632801	30622564
soc-LiveJournal1	D	4846606	68475391

TABLE I: The datasets, direction on their edges (**U**ndirected or **D**irected), number of nodes and edges.

b) Density of $G[\mathcal{C}]$: As we discussed in Sect. III-D, in $\mathcal{A}_D$, we sample several subgraphs of $G \sim \mathcal{M}$, and decompose each G into its strongly connected components (SCC). Note, each reachability question can be addressed in the graph $G[\mathcal{C}]$, the DAG induced by the strongly connected components of G. Therefore, a natural question to ask is that how dense (or sparse!) is the graph $G[\mathcal{C}]$, since sparser graphs are easier to store and to process for reachability questions. Density of a graph is defined as the number of edges divided by the number of nodes. We show the results in Table II. For each graph/model we sample 10 graphs $G \sim \mathcal{M}$, and the averages are reported. As shown, the graphs are highly sparse, which motivates our approach.

Datasets	$p = 0.05$	$p = 0.1$	$p = 0.2$
email-Enron	0.905	0.944	0.984
email-EuAll	0.955	0.961	0.964
com-dblp	0.813	0.943	1.033
com-youtube	0.849	0.89	0.934
soc-Epinions1	0.954	0.952	0.973
soc-Slashdot0902	0.987	0.999	1.012
ego-twitter	1.197	1.237	1.244
wiki-Talk	1.003	1.012	1.013
soc-Pokec	1.166	1.177	1.173
soc-LiveJournal1	1.03	1.055	1.048

TABLE II: The density of the $G[\mathcal{C}]$ graphs, for $G \sim \mathcal{M}$.

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

c) Maximum number of Descendants: As we discussed in Sect. III-D, the number of bits in the each reachability vector (denoted by w) necessary to obtain a multiplicative approximation is lower-bounded by a function of $\alpha(S)$, the number of SCCs that are reachable from S. The smaller the $\alpha(S)$ the smaller the memory required.

Here we evaluate our heuristic to reduce the amount of memory need using L hubs. we consider the following cases: (1) no hubs, i.e., $L = \emptyset$, and (2) when L is the t-top degree nodes in G for $t \in \{1, 2, 3, 4, 5, 10\}$. In Fig. 1 we present the maximum number of descendants of any node, using different number of hubs. Each number is the average for 10 sampled random subgraph $G \sim \mathcal{M}$. As shown, by using hubs, the number of direct descendants suddenly drops, which then, improves the space required dramatically. For space limitations, we provide the plot for $p = 0.2$. However for $p = 0.05$ and $p = 0.05$ we obtain very similar results.

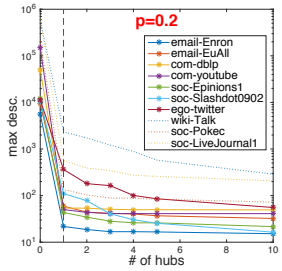


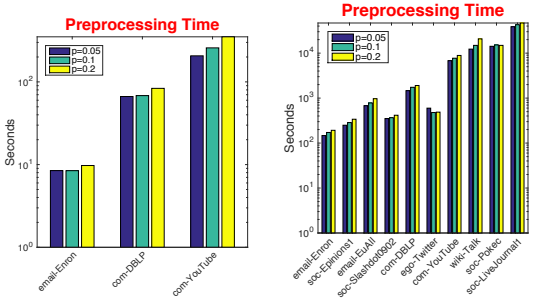
Fig. 1: Maximum number of descendants with/without using hubs.

B. Efficiency

In this section we provide the running time for preprocessing the graphs $G \sim \mathcal{M}$, and the speed-up we obtain for answer $(S;T)$-queries, i.e. estimating $\mathsf{Inf}(S;T)$, compared to running the standard Monte Carlo simulations to estimate $\mathsf{Inf}(S;T)$. For these experiments we set the parameters as follows:

- $|L| = 10$: number of hubs,
- $k = 5$: number of hash functions,
- $w = 2^{14} = 16384$: number of bits for reachability vectors,
- $\ell = 50$: number of sampled random subgraphs $G \sim \mathcal{M}$,

d) Preprocessing: In Figure 2 we show the running time for preprocessing the randomly sampled graphs $G \sim \mathcal{M}$: numbers are averaged over different samples of G.



Applying $\mathcal{A}_U$ approach Applying $\mathcal{A}_D$ approach

Fig. 2: Preprocessing time for both approaches.

e) Speed-ups in answering queries: Here, we compare the time to answer an $(S;T)$-query using our approach T_1, to the running time of Monte Carlo simulations, T_2. In both cases we use the same number $\ell = 50$ of simulations. Here, by speed-up we mean the $\frac{T_2}{T_1}$ ratio.

For our experiments, we report the average results over 50 $(S;T)$-queries that are obtained using three different methods, in each case we set $|S| = |T| = 1000$. In the **UNIF** case both S and T are subsets of nodes chosen uniformly at random, among the nodes with non-zero out-degree and non-zero in-degree, respectively. In the **2-Hops** experiment S is a subset of nodes with non-zero out-degree, chosen uniformly at random. T is obtained by randomly choosing from the nodes that can be reached from S by using at most 2 edges (i.e., from 2-neighborhood of S). Finally, in the **DEG** case S and T are obtained by sampling the nodes with probability proportional to their out-degree and in-degree, respectively.

In Table III the time (seconds) to answer the queries with pre-processed data, and in Fig. 3 the speed up we obtain is presented. For each graph, the speed-ups are the average of 50 random $(S;T)$-queries. As shown, we obtain tremendous speed-up by using our preprocessing techniques, in both $\mathcal{A}_U$ and $\mathcal{A}_D$ approaches reducing the running time for a given query up to a 10^3 factor. Again, for lack of space, in Fig. 3 we present the results for UNIF only, as the other two methods give us very similar results.

	UNIF			2-Hops			DEG		
Dataset $\backslash p =$	***0.05***	***0.1***	***0.2***	***0.05***	***0.1***	***0.2***	***0.05***	***0.1***	***0.2***
email-Enron(U)	0.03	0.03	0.03	0.04	0.03	0.03	0.03	0.02	0.02
com-DBLP(U)	0.41	0.05	0.03	0.17	0.05	0.05	0.05	0.04	0.03
com-YouTube(U)	1.51	0.06	0.05	0.05	0.05	0.05	1.19	0.05	0.06
soc-Epinions1(D)	2.87	2.94	3.02	2.87	2.93	3.01	3.24	2.9	2.78
email-EuAll(D)	2.78	2.65	2.78	2.83	2.7	2.77	3.31	2.95	3.06
soc-Slashdot0902(D)	2.93	2.84	2.88	3.02	2.86	2.91	3.72	2.73	2.62
ego-Twitter(D)	4.69	3.44	3.13	4.69	3.43	3.13	5.43	3.53	3
wiki-Talk(D)	10.08	10.61	11.21	10.19	10.62	10.35	11.62	13.1	11.32
soc-Pokec(D)	11.24	14.65	20.22	11.19	14.33	19.7	11.52	10.72	10.39
soc-LiveJournal1(D)	10.85	12.49	18.55	11.05	11.31	11.77	11.96	14.19	23.28

TABLE III: The time (seconds) consumed by our algorithm for answering queries. Letters **U** and **D** correspond to $\mathcal{A}_U$ and $\mathcal{A}_D$, respectively.

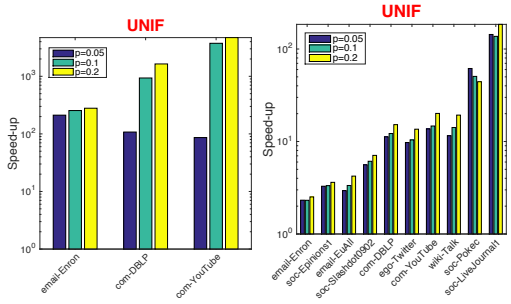


Fig. 3: Speed-ups: Applying $\mathcal{A}_U$ and $\mathcal{A}_D$ methods.

C. Relative Errors

Finally in this section, we discuss the accuracy of our estimated influence of an $(S;T)$-query, by computing its relative error: For each $(S;T)$-query, we run 300 Monte Carlo simulation to estimate $\mathsf{Inf}(S;T)$ as the ground truth, and we denote this estimate by I_1. If I_2 is our estimate of $\mathsf{Inf}(S;T)$, the relative error is $\frac{|I_1 - I_2|}{I_1}$.

The results are provided in Fig. 4. As shown, overall the relative errors are very small (true for all cases, show here only for UNIF, for space limitation), in particular, we observe that the higher the probability p, the lower the error observed. In almost all cases we obtain relative errors smaller than 10% and our relative errors can be as low as $< 0.1\%$

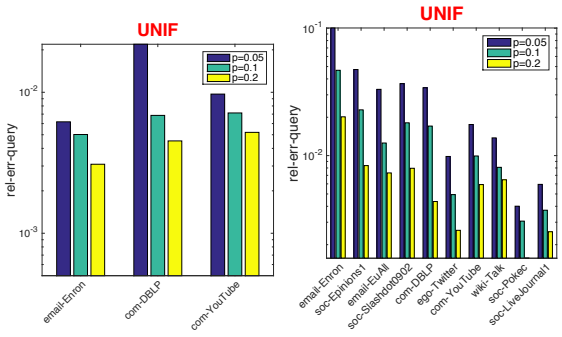


Fig. 4: Relative error for the influence estimation in $\mathcal{A}_U$ and $\mathcal{A}_D$.

V. Conclusion

In this work we introduced a new problem, Targeted-Influence problem, (TIP) whose goal is to answer the influence queries in an online fashion, efficiently and accurately, where each influence query is an $(S;T)$ pair of subsets of nodes, and the influence of S over T is desired. We presented algorithms for preprocessing the graphs and answering the influence queries online, both for directed and undirected graphs, and our algorithms are highly parallelizable. We also provided analysis and theoretical guarantees showing accuracy and efficiency of our approach. We also introduced the snapshot model, which generalizes the classic and widely used Independent Cascade and Linear Threshold models. We finally, perform an extensive experimental analysis, demonstrating the accuracy, efficiency, and scalability of our method.

References

[1] Ç. Aslay, N. Barbieri, F. Bonchi, and R. A. Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, pages 295–306, 2014.

[2] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 946–957. SIAM, 2014.

[3] J. J. Brown and P. H. Reingen. Social ties and word-of-mouth referral behavior. *Journal of Consumer research*, pages 350–362, 1987.

[4] C.-W. Chang, M.-Y. Yeh, and K.-T. Chuang. On influence maximization to target users in the presence of multiple acceptances. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1592–1593. ACM, 2015.

[5] Y. Cheng, Y. Yuan, L. Chen, and G. Wang. The reachability query over distributed uncertain graphs. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 786–787. IEEE, 2015.

[6] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Distance-based influence in networks: Computation and maximization. *arXiv preprint*.

[7] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 629–638. ACM, 2014.

[8] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[9] P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001.

[10] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion networks. In *Advances in neural information processing systems*, pages 3147–3155, 2013.

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

[12] J. Goldenberg, B. Libai, and E. Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 2001:1, 2001.

[13] J. Guo, P. Zhang, C. Zhou, Y. Cao, and L. Guo. Personalized influence maximization on social networks. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 199–208. ACM, 2013.

[14] M. HELFT. *Google: 1 Million Advertisers in 2007, More Now*, Jan. 2009. http://bits.blogs.nytimes.com/2009/01/08/google-1-million-advertisers-in-2007-more-now/?_r=0.

[15] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment*, 4(9):551–562, 2011.

[16] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 595–608. ACM, 2008.

[17] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[18] A. Khan, F. Bonchi, A. Gionis, and F. Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014.

[19] J.-R. Lee and C.-W. Chung. A query approach for influence maximization on specific users in social networks. *IEEE Transactions on knowledge and data engineering*, 27(2):340–353, 2015.

[20] Y. Li, D. Zhang, and K.-L. Tan. Real-time targeted influence maximization for online advertisements. *Proceedings of the VLDB Endowment*, 8(10):1070–1081, 2015.

[21] B. Lucier, J. Oren, and Y. Singer. Influence at scale: Distributed computation of complex contagion in networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744. ACM, 2015.

[22] V. Mahajan, E. Muller, and F. M. Bass. New product diffusion models in marketing: A review and directions for research. *The journal of marketing*, pages 1–26, 1990.

[23] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70. ACM, 2002.

[24] M. G. Rodriguez and B. Schölkopf. Influence maximization in continuous time diffusion networks. *arXiv preprint arXiv:1205.1682*, 2012.

[25] E. M. Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[26] J. Su, Q. Zhu, H. Wei, and J. X. Yu. Reachability querying: Can it be even faster? *IEEE Transactions on Knowledge and Data Engineering*, 2016.

[27] K. Subbian, C. C. Aggarwal, and J. Srivastava. Querying and tracking influencers in social streams. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 493–502. ACM, 2016.

[28] F. Tang, Q. Liu, H. Zhu, E. Chen, and F. Zhu. Diversified social influence maximization. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 455–459. IEEE, 2014.

[29] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: a martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1539–1554. ACM, 2015.

[30] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86. ACM, 2014.

[31] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[32] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1601–1606. ACM, 2013.

[33] C. Zhou and L. Guo. A note on influence maximization in social networks from local to global and beyond. *Procedia Computer Science*, 30:81–87, 2014.