Boosting Information Spread: An Algorithmic Approach

Yishi Lin, Wei Chen *Member, IEEE*, John C.S. Lui *Fellow, IEEE*

***Abstract*—The majority of influence maximization (IM) studies focus on targeting influential *seeders* to trigger substantial information spread in social networks. Motivated by the observation that incentives could "boost" users so that they are more likely to be influenced by friends, we consider a new and complementary k-*boosting problem* which aims at finding k users to boost so to trigger a maximized "boosted" influence spread. The k-boosting problem is *different* from the IM problem because boosted users behave differently from seeders: *boosted* users are initially uninfluenced and we only increase their probability to be influenced. Our work also *complements* the IM studies because we focus on triggering larger influence spread on the basis of given seeders. Both the NP-hardness of the problem and the non-submodularity of the objective function pose challenges to the k-boosting problem. To tackle the problem on general graphs, we devise two efficient algorithms with the *data-dependent* approximation ratio. To tackle the problem on bidirected trees, we present an efficient greedy algorithm and a dynamic programming that is a fully polynomial-time approximation scheme. Experiments using real social networks and synthetic bidirected trees verify the efficiency and effectiveness of the proposed algorithms. In particular, on general graphs, boosting solutions returned by our algorithms achieves boosts of influence that are up to several times higher than those achieved by boosting intuitive solutions with no approximation guarantee. We also explore the "budget allocation" problem experimentally, demonstrating the benefits of allocating the budget to *both* seeders and boosted users.**

***Index Terms*—Influence maximization, Information boosting, Social networks, Viral marketing**

I. Introduction

WITH the popularity of online social networks, *viral marketing*has become a powerful tool for companies to promote sales. In viral marketing campaigns, companies target influential users by offering free products or services with the hope of triggering a chain reaction of adoption. These targeted users are often called *Initial adopters* or *seeds*. Motivated by the need for effective viral marketing strategies, *influence maximization* has become a fundamental research problem in the past decade. The goal of influence maximization is usually to identify influential initial adopters [1–8].

In practical marketing campaigns, companies often consider *a mixture of promotion strategies*. Besides targeting influential users as initial adopters, we list some others as follows.

- *Incentive programs*: Companies offer incentives such as coupons or free trials to attract potential customers. Targeted customers are more likely to be influenced by their friends.
- *Social media advertising*: Companies reach intended audiences via advertising. According to the "Global Trust in Advertising" survey [9], owned online channels are the second most trusted advertising formats, second only to recommendations from family and friends. We believe that customers targeted by ads are more likely to follow their acquaintances' purchases.
- *Referral marketing*: Companies encourage customers to refer others to use the product by offering rewards such as cash back. In this case, targeted customers are more likely to influence their friends.

These marketing strategies are able to "*boost*" the influence transferring through customers. Furthermore, for companies, the cost of "boosting" a customer (e.g., the average redemption and distribution cost per coupon, or the advertising cost per customer) is much lower than the cost of nurturing an influential user as an initial adopter and a product evangelist. Although identifying influential initial adopters have been actively studied, very little attention has been devoted to studying how to utilize incentive programs or other strategies to further increase the influence spread of initial adopters.

In this paper, we study the problem of finding k boosted users so that when their friends adopt a product, they are more likely to make the purchase and continue to influence others. Motivated by the need for modeling boosted customers, we propose a novel *influence boosting model*. In our model, *seed users* generate influence same as in the classical *Independent Cascade* (IC) model. In addition, we introduce the *boosted user* as a new user type. They represent customers with incentives such as coupons. They are uninfluenced at the beginning of the influence propagation process. However, they are more likely to be influenced by their friends and further spread the influence to others. In other words, they "boost" the influence transferring through them. Under the influence boosting model, we study *how to boost the influence spread given initial adopters*. More precisely, given initial adopters, we are interested in identifying k users among other users, so that the expected influence spread upon "boosting" them is maximized. Because of the differences in behaviors between seed users and boosted users, our work is very different from influence maximization studies focusing on selecting seeds.

Our work also complements the studies of influence maximization problems. First, compared with nurturing an initial adopter, *boosting* a potential customer usually incurs a lower cost. For example, companies may need to offer free products to initial adopters, but only need to offer coupons to boost

Manuscript received July 10, 2017. Yishi Lin and John C.S. Liu are partially supported by the GRF #14630815. Wei Chen is partially supported by the National Natural Science Foundation of China (Grant No. 61433014).

Yishi Lin and John C.S. Lui are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. (Emails: yishilin14@gmail.com, cslui@@cse.cuhk.edu.hk)

Wei Chen is with Microsoft Research, Beijing, China. (Email: weic@microsoft.com)

potential customers. With both our methods that identify users to *boost* and algorithms that select initial adopters, companies have more flexibility in allocating their marketing budgets. Second, initial adopters are sometimes predetermined. For example, they may be advocates of a particular brand or prominent bloggers in the area. In this case, our study suggests how to effectively utilize incentive programs or similar marketing strategies to take the influence spread to the next level.

Contributions. We summarize our contributions as follows.

- We formulate a k*-boosting* problem that asks how to maximize the boost of influence spread under a novel *influence boosting model*. The k-boosting problem is NP-hard. Computing the boost of influence spread is #P-hard. Moreover, the boost of influence spread does not possess the submodularity, meaning that the greedy algorithm does not provide performance guarantee.
- We present approximation algorithms `PRR-Boost` and `PRR-Boost-LB` for the k-boosting problem. For the k-boosting problem on bidirected trees, we present a greedy algorithm `Greedy-Boost` based on a linear-time exact computation of the boost of influence spread and a fully polynomial-time approximation scheme (FPTAS) `DP-Boost` that returns near-optimal solutions. [1] `DP-Boost` provides a benchmark for the greedy algorithm, at least on bi-directed trees, since it is very hard to find near optimal solutions in general cases. Moreover, the algorithms on bidirected trees may be applicable to situations where information cascades more or less follow a fixed tree architecture.
- We conduct extensive experiments using real social networks and synthetic bidirected trees. Experimental results show the efficiency and effectiveness of our proposed algorithms, and their superiority over intuitive baselines.

Paper organization. Section II provides background. We describe the *influence boosting model* and the k-boosting problem in Section III. We present building blocks of `PRR-Boost` and `PRR-Boost-LB` for the k-boosting problem in Section IV, and the detailed algorithm design in Section V. We present `Greedy-Boost` and `DP-Boost` for the k-boosting problem on bidirected trees in Section VI. We show experimental results in Sections VII-VIII. Section IX concludes the paper. Due to space limit, we omit most of the proofs in this paper. The full analysis can be found in our technical report [10].

II. Background and Related Work

In this section, we provide backgrounds about influence maximization problems and related works.

Classical influence maximization problems. Kempe et al. [1] first formulated the *influence maximization* problem that asks to select a set S of k nodes so that the expected influence spread is maximized under a predetermined influence propagation model. The *Independent Cascade* (IC) model is one classical model that describes the influence diffusion process [1]. Under the IC model, given a graph $G = (V, E)$, influence probabilities on edges and a set $S \subseteq V$ of *seeds*, the influence propagates as follows. Initially, nodes in S are activated. Each newly activated node u influences its neighbor v with probability p_{uv}. The influence spread of S is the expected number of nodes activated at the end of the influence diffusion process. Under the IC model, the influence maximization problem is NP-hard [1] and computing the expected influence spread for a given S is #P-hard [4]. A series of studies have been done to approximate the influence maximization problem under the IC model and other models [3, 4, 6–8, 11–14].

Influence maximization on trees. Under the IC model, tree structure makes the influence computation tractable. To devise greedy "seed-selection" algorithms on trees, several studies presented methods to compute the "marginal gain" of influence spread on trees [4, 15]. Our computation of "marginal gain of boosts" on trees is more advanced than the previous methods: It runs in linear-time, it considers the behavior of "boosting", and we assume that the benefits of "boosting" can be transmitted in both directions of an edge. On bidirected trees, Bharathi et al. [16] described an FPTAS for the classical influence maximization problem. Our FPTAS on bidirected trees is different from theirs because "boosting" a node and targeting a node as a "seed" have significantly different effects.

Boost the influence spread. Several works studied how to recommend friends or inject links into social networks in order to boost the influence spread [17–22]. Lu et al. [23] studied how to maximize the expected number of adoptions by targeting initial adopters of a complementing product. Chen et al. [24] considered how to select a subset of seed content providers and a subset of seed customers so that the expected number of influenced customers is maximized. Their model differs from ours in that they only consider influence originators selected from content providers, which are separated from the social network, and influence boost is only from content providers to consumers in the social network. Yang et al. [22] studied how to offer discounts assuming that the probability of a customer being an initial adopter is a function of the discounts. Different from the above studies, we study how to *boost the spread of influence* when seeds are given. This article is an extended version of our conference paper [25] that formulated the k-boosting problem and presented algorithms for it. We add two new algorithms that tackle the k-boosting problem in bidirected trees, and report new experimental results.

III. Model and Problem Definition

In this section, we first define the *influence boosting model* and the k-boosting problem. Then, we highlight the challenges.

A. Model and Problem Definition

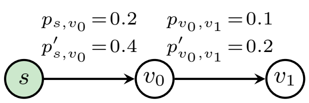
Traditional studies of the influence maximization problem focus on how to identify a set of k influential users (or *seeds*) who can trigger the largest influence diffusion. In this paper, we aim to *boost* the influence propagation assuming that *seeds* are given. We first define the *influence boosting model.*

Definition 1 (Influence Boosting Model)**.** *Suppose we are given a directed graph $G = (V, E)$ with n nodes and m edges, two influence probabilities p_{uv} and p'_{uv} (with $p'_{uv} > p_{uv}$) on each edge e_{uv}, a set $S \subseteq V$ of seeds, and a set $B \subseteq V$*

[1] An FPTAS for a maximization problem is an algorithm that given any $\epsilon > 0$, it can approximate the optimal solution with a factor $1 - \epsilon$, with running time polynomial to the input size and $1/\epsilon$.

of boosted nodes. Influence propagates in discrete time steps as follows. If v is not boosted (resp. is boosted), each of its newly-activated in-neighbor u influences v with probability p_{uv} (resp. p'_{uv}).

In Definition 1, we assume that "boosted" users are more likely to be influenced. Our study can also be adapted to the case where boosted users are more influential: if a newly-activated user u is boosted, she influences her neighbor v with probability p'_{uv} instead of p_{uv}. To simplify the presentation, we focus on the influence boosting model in Definition 1.



B	$\sigma_S(B)$	$\Delta_S(B)$
$\emptyset$	1.22	0.00
$\{v_0\}$	1.44	0.22
$\{v_1\}$	1.24	0.02
$\{v_0, v_1\}$	1.48	0.26

Fig. 1: Example of the influence boosting model ($S=\{s\}$).

Let $\sigma_S(B)$ be the expected influence of S upon boosting nodes in B. We refer to $\sigma_S(B)$ as the *boosted influence spread*. Let $\Delta_S(B) = \sigma_S(B) - \sigma_S(\emptyset)$. We refer to $\Delta_S(B)$ as the *boost of influence spread of* B, or simply the *boost of* B. Consider the example in Figure 1. We have $\sigma_S(\emptyset) = 1.22$, which is essentially the influence of S in the IC model. When we boost node v_0, we have $\sigma_S(\{v_0\}) = 1 + 0.4 + 0.04 = 1.44$, and $\Delta_S(\{v_0\}) = 0.22$. We now formulate the *k-boosting problem*.

Definition 2 (k-Boosting Problem)**.** *Given a directed graph $G = (V, E)$, influence probabilities p_{uv} and p'_{uv} on every edges e_{uv}, and a set $S \subseteq V$ of seed nodes, find a boost set $B \subseteq V$ with k nodes, such that the boost of influence spread of B is maximized. That is, find $B^* = \arg\max_{B \subseteq V, |B| \leq k} \Delta_S(B)$.*

By definition, the k-boosting problem is very different from the classical influence maximization problem. Moreover, boosting nodes that significantly increase the influence spread when used as additional seeds could be extremely inefficient. For example, in Figure 1, if we are allowed to select one more seed, we should select v_1. However, if we can boost a node, boosting v_0 is much better than boosting v_1. Section VII provides more experimental results.

B. Challenges of the Boosting Problem

We now analyze the k-boosting problem and show the challenges. Theorem 1 summarizes the hardness results.

Theorem 1 (Hardness)**.** *The k-boosting problem is NP-hard. Computing $\Delta_S(B)$ given S and B is #P-hard.*

Proof. The NP-hardness is proved by a reduction from the NP-complete *Set Cover* problem [26]. The #P-hardness of the computation is proved by a reduction from the #P-complete counting problem of *s-t connectedness* in directed graphs [27]. The full analysis is in our technical report [10]. ■

Non-submodularity of the boost of influence. Because of the above hardness results, we explore approximation algorithms to tackle the problem. In most influence maximization problems, the influence of the seed set S (i.e., the objective function) is a monotone and submodular function of S.[2] Thus, a natural greedy algorithm provides an approximation guarantee [1, 6–8, 14, 28]. However, the objective function $\Delta_S(B)$ in our problem is *neither submodular nor supermodular* on the set B of boosted nodes. On one hand, when we boost several nodes on different parallel paths from seeds, their overall boosting effect exhibits a *submodular behavior*. On the other hand, when we boost several nodes on a path starting from a seed, their boosting effects can be cumulated, generating a larger overall effect than the sum of their individual boosting effect. This is in fact a *supermodular behavior*. The non-submodularity of $\Delta_S(\cdot)$ indicates that the boosting set returned by the greedy algorithm may not have the $(1-1/e)$-approximation guarantee. Therefore, the non-submodularity of the objective function poses an additional challenge.

[2] A set function f is monotone if $f(S) \leq f(T)$ for all $S \subseteq T$; it is submodular if $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ for all $S \subseteq T$ and $v \notin T$, and it is supermodular if $-f$ is submodular.

IV. Boosting on General Graphs: Building Blocks

In this section, we present three building blocks for solving the k-boosting problem: (1) a state-of-the-art influence maximization framework, (2) the *Potentially Reverse Reachable Graph* for estimating the boost of influence spread, and (3) the *Sandwich Approximation* strategy [23] for maximizing non-submodular functions. Our algorithms `PRR-Boost` and `PRR-Boost-LB` integrate the three building blocks. We will present their detailed algorithm design in Section V.

A. State-of-the-art influence maximization techniques

One state-of-the-art influence maximization framework is the *Influence Maximization via Martingale* (`IMM`) method [8] based on the idea of *Reverse-Reachable Sets* (RR-sets) [6]. We utilize the `IMM` method in this paper. But other RR-set-based frameworks such as the *Stop-and-Stare Algorithm* (SSA) and the *Dynamic Stop-and-Stare Algorithm* (D-SSA) [14] could also be applied.

RR-sets. An *RR-set* for a node r is a random set R of nodes, such that for any seed set S, the probability that $R \cap S \neq \emptyset$ equals the probability that r can be activated by S in a random diffusion process. Node r may also be selected uniformly at random from V, and the RR-set will be generated accordingly with r. One key property of RR-sets is that the expected influence of S equals to $n \cdot \mathbb{E}[\mathbb{I}(R \cap S \neq \emptyset)]$ for all $S \subseteq V$, where $\mathbb{I}(\cdot)$ is the indicator function and the expectation is taken over the randomness of R.

General `IMM` algorithm. The `IMM` algorithm has two phases. The *sampling* phase generates a sufficiently large number of random RR-sets such that the estimation of the influence spread is "accurate enough". The *node selection* phase greedily selects k seed nodes based on their estimated influence spread. If generating a random RR-set takes time $O(EPT)$, `IMM` returns a $(1-1/e-\epsilon)$-approximate solution with probability at least $1-n^{-\ell}$, and runs in $O(\frac{EPT}{OPT} \cdot (k+\ell)(n+m)\log n/\epsilon^2)$ expected time, where OPT is the optimal expected influence.

B. Potentially Reverse Reachable Graphs

We now describe how we estimate the boost of influence. The estimation is based on the concept of the *Potentially Reverse Reachable Graph* (PRR-graph) defined as follows.

Definition 3 (Potentially Reverse Reachable Graph)**.** *Let r be a node in G. A Potentially Reverse Reachable Graph (PRR-graph) R for a node r is a random graph generated as follows. We first sample a deterministic copy g of G: each edge e_{uv} is "live" in g with probability p_{uv}, "live-upon-boost" with probability $p'_{uv} - p_{uv}$, and "blocked" with probability $1 - p'_{uv}$. The PRR-graph R is the minimum subgraph of g containing all paths from seed nodes to r through non-blocked edges in g. We refer to r as the "root node". When r is also selected from V uniformly at random, we simply refer to the generated PRR-graph as a random PRR-graph (for a random root).*

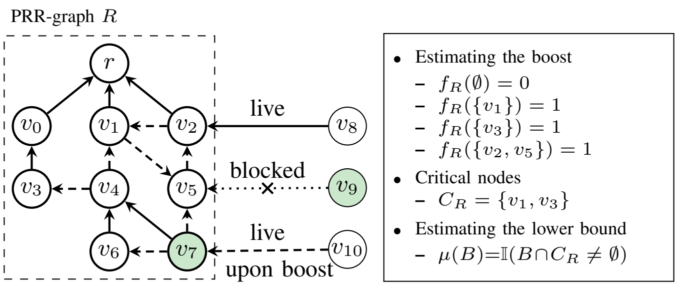


Fig. 2: Example of a Potentially Reverse Reachable Graph.

Figure 2 shows an example of a PRR-graph R. Node r is the *root* node. Shaded nodes are seed nodes. Solid, dashed and dotted arrows with crosses represent live, live-upon-boost and blocked edges, respectively. The PRR-graph R is the subgraph in the dashed box. Nodes and edges outside the dashed box do not belong to the PRR-graph because they are not on any paths from seed nodes to r that only contain non-blocked edges. By definition, a PRR-graph may contain loops. For example, in Figure 2, R contains a loop among nodes v_1, v_5, and v_2.

Estimating the boost of influence. Let R be a given PRR-graph with root r. By definition, every edge in R is either *live* or *live-upon-boost*. We say a path in R is *live* if and only if it contains only live edges. We say that a path is *live upon boosting* B if and only if the path is not a *live* one, but every edge e_{uv} on it is either *live* or *live-upon-boost* with $v \in B$. For example, in Figure 2, the path from v_3 to r is *live*, and the path from v_7 to r via v_4 and v_1 is *live upon boosting* $\{v_1\}$. Define $f_R(B) : 2^V \to \{0, 1\}$ as: $f_R(B) = 1$ if and only if, in R, (1) there is no *live* path from seed nodes to r; and (2) a path from a seed node to r is *live upon boosting* B. In Figure 2, if $B = \emptyset$, there is no live path from the seed node v_7 to r upon boosting B. Therefore, we have $f_R(\emptyset) = 0$. There is a live path from the seed node v_7 to r if we boost v_1, thus we have $f_R(\{v_1\}) = 1$. Similarly, we have $f_R(\{v_3\}) = f_R(\{v_2, v_5\}) = 1$. Based on the above definition of $f_R(\cdot)$, we have the following lemma.

Lemma 1. *For any $B \subseteq V$, we have $n \cdot \mathbb{E}[f_R(B)] = \Delta_S(B)$, where the expectation is taken over the randomness of R.*

Proof. For a random PRR-graph R, $\Pr[f_R(B) = 1]$ equals the difference between probabilities that a random node in G is activated given that we boost B and $\emptyset$. ∎

Let $\mathcal{R}$ be a set of independent random PRR-graphs, define

$$\hat{\Delta}_{\mathcal{R}}(B) = \frac{n}{|\mathcal{R}|} \cdot \sum_{R \in \mathcal{R}} f_R(B), \forall B \subseteq V. \tag{1}$$

By Chernoff bound, $\hat{\Delta}_{\mathcal{R}}(B)$ closely estimates $\Delta_S(B)$ for any $B \subseteq V$ if $|\mathcal{R}|$ is sufficiently large.

C. Sandwich Approximation Strategy

To tackle the non-submodularity of function $\Delta_S(\cdot)$, we apply the *Sandwich Approximation* (SA) strategy [23]. First, we find submodular lower and upper bound functions of Δ_S, denoted by μ and ν. Then, we select node sets B_Δ, B_μ and B_ν by greedily maximizing Δ_S, μ and ν under the cardinality constraint of k. Ideally, we return $B_{\text{sa}} = \arg\max_{B \in \{B_\mu, B_\nu, B_\Delta\}} \Delta_S(B)$ as the final solution. Let the optimal solution of the k-boosting problem be B^* and let $OPT = \Delta_S(B^*)$. Suppose B_μ and B_ν are $(1 - 1/e - \epsilon)$-approximate solutions for maximizing μ and ν, we have

$$\Delta_S(B_{\text{sa}}) \geq \frac{\mu(B^*)}{\Delta_S(B^*)} \cdot (1 - 1/e - \epsilon) \cdot OPT, \tag{2}$$

$$\Delta_S(B_{\text{sa}}) \geq \frac{\Delta_S(B_\nu)}{\nu(B_\nu)} \cdot (1 - 1/e - \epsilon) \cdot OPT. \tag{3}$$

Thus, to obtain a good approximation guarantee, at least one of μ and ν should be close to Δ_S. In this work, we derive a submodular lower bound μ of Δ_S using the definition of PRR-graphs. Because μ is significantly closer to Δ_S than any submodular upper bound we have tested, we only use the lower bound function μ and the "lower-bound side" of the SA strategy with approximation guarantee in Inequality (2).

Submodular lower bound of Δ_S. Let R be a PRR-graph with the root node r. Let $C_R = \{v | f_R(\{v\}) = 1\}$. We refer to nodes in C_R as *critical nodes* of R. Intuitively, the root node r becomes activated if we boost any node in C_R. For any $B \subseteq V$, define $f_R^-(B) = \mathbb{I}(B \cap C_R \neq \emptyset)$ and $\mu(B) = n \cdot \mathbb{E}[f_R^-(B)]$ where the expectation is taken over the randomness of R. Lemma 2 shows the properties of the function μ.

Lemma 2. *We have $\mu(B) \leq \Delta_S(B)$ for all $B \subseteq V$. Moreover, $\mu(B)$ is a submodular function of B.*

Proof. For all $B \subseteq V$, we have $\mu(B) \leq \Delta_S(B)$ because we have $f_R^-(B) \leq f_R(B), \forall R$. Moreover, $\mu(B)$ is submodular on B because $f_R^-(B)$ is submodular on B for any PRR-graph R. ∎

Our experiments show that μ is close to Δ_S especially for small k (e.g., less than a thousand). Define

$$\hat{\mu}_{\mathcal{R}}(B) = \frac{n}{|\mathcal{R}|} \cdot \sum_{R \in \mathcal{R}} f_R^-(B), \forall B \subseteq V.$$

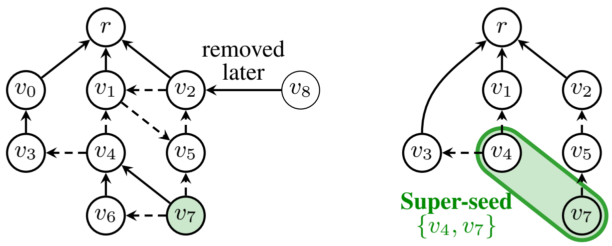
Because $f_R^-(B)$ is submodular on B for any PRR-graph R, $\hat{\mu}_{\mathcal{R}}(B)$ is submodular on B. Moreover, by Chernoff bound, $\hat{\mu}_{\mathcal{R}}(B)$ is close to $\mu(B)$ when $|\mathcal{R}|$ is sufficiently large.

V. Boosting On General Graphs: Algorithm Design

In this section, we first present how we generate random PRR-graphs. Then we obtain the overall algorithms by integrating all building blocks.

A. Generating random PRR-graphs

We classify PRR-graphs into three categories. Let R be a PRR-graph with root node r. (1) *Activated*: If there is a *live* path from a seed node to r; (2) *Hopeless*: There is no path from seeds to r with at most k *non-live* edges; (3) *"Boostable"*: not the above two categories. If R is not *boostable* (i.e. case (1) or (2)), we have $f_R(B) = f_R^-(B) = 0$ for all $B \subseteq V$. Therefore, for "non-boostable" PRR-graphs, we only count their occurrences and we terminate the generation of them once we know they are not *boostable*. Algorithm 1 depicts the generation of a random PRR-graph in two phases. The first phase (Lines 1-19) generates a PRR-graph R. If R is boostable, the second phase compresses R to reduce its size. Figure 3 shows the results of two phases, given that the status sampled for every edge is same as that in Figure 2.



(a) Results of phase I (b) Results of phase II

Fig. 3: Generation of a PRR-Graph. (Solid and dashed arrows represent live and live-upon-boost edges respectively.)

Algorithm 1: Generating a random PRR-graph (G, S, k)

---

1 Select a random node r as the root node
2 **if** $r \in S$ **then return** *R is activated*
3 Create a graph R with a singleton node r
4 Create a double-ended queue Q with $(r, 0)$
5 Initialize $d_r[r] \leftarrow 0$ and $d_r[v] \leftarrow +\infty, \forall v \neq r$
6 **while** *Q is not empty* **do**
7 $\quad (u, d_{ur}) \leftarrow Q$.dequeue_front()
8 $\quad$ **if** $d_{ur} > d_r[u]$ **then** continue // we've processed u
9 $\quad$ **for** *each non-blocked incoming edge e_{vu} of u* **do**
10 $\quad\quad d_{vr} \leftarrow \mathbb{I}(e_{vu}$ is *live-upon-boost*$) + d_{ur}$
11 $\quad\quad$ **if** $d_{vr} > k$ **then** continue // pruning
12 $\quad\quad$ Add e_{vu} to R
13 $\quad\quad$ **if** $d_{vr} < d_r[v]$ **then**
14 $\quad\quad\quad d_r[v] \leftarrow d_{vr}$
15 $\quad\quad\quad$ **if** $v \in S$ **then**
16 $\quad\quad\quad\quad$ **if** $d_r[v] = 0$ **then return** *R is activated*
17 $\quad\quad\quad$ **else if** $d_{vr} = d_{ur}$ **then** Q.enqueue_front((v, d_{vr}))
18 $\quad\quad\quad$ **else** Q.enqueue_back((v, d_{vr}))
19 **if** *there is no seed in R* **then return** *R is hopeless*
20 Compress the boostable R to reduce its size
21 **return** *a compressed boostable R*

Phase I: Generating a PRR-graph. Let r be a random node. We include into R all *non-blocked* paths from seed nodes to r with at most k *live-upon-boost* edges via a a backward Breadth-First Search (BFS) from r. The status of each edge (i.e., *live, live-upon-boost, blocked*) is sampled when we first process it. The detailed backward BFS is as follows. Define the *distance* from u to v as the minimum number of nodes we have to boost so that at least a path from u to v becomes live. For example, in Figure 3a, the distance from v_7 to r is one. We use $d_r[\cdot]$ to maintain the distances from nodes to the root node r. Initially, we have $d_r[r] = 0$ and we enqueue $(r, 0)$ into a double-ended queue Q. We repeatedly dequeue and process a node-distance pair (u, d_{ur}) from the head of Q, until the queue is empty. Note that the distance d_{ur} in a pair (u, d_{ur}) is the shortest known distance from u to r when the pair was enqueued. Thus we may find $d_{ur} > d_r[u]$ in Line 8. Pairs (u, d_{ur}) in Q are in the ascending order of the distance d_{ur} and there are at most two different values of distance in Q. Therefore, we process nodes in the ascending order of their shortest distances to r. When we process a node u, for each of its non-blocked incoming edge e_{vu}, we let d_{vr} be the shortest distance from v to r via u. If $d_{vr} > k$, all paths from v to r via u are impossible to become live upon boosting at most k nodes, therefore we ignore e_{vu} in Line 11. This is in fact a pruning strategy, which is effective especially for small values of k. If $d_{vr} \leq k$, we insert e_{vu} into R, update $d_r[v]$ and enqueue (v, d_{vr}) if necessary. If we find out that the distance from a seed node to r is zero, we know R is *activated* and we terminate the generation (Line 16). If we do not visit any seed node during the backward BFS, R *is hopeless* and we terminate the generation (Line 19).

Remarks. At the end of phase I, R may include extra nodes and edges (e.g., *non-blocked* edges not on any non-blocked paths from seeds to the root). For example, Figure 3a shows the results of the first phase, given that we are constructing a PRR-graph according to the root node and sampled edge status in Figure 2. There is an extra edge from v_8 to v_2. All extra nodes and edges will be removed in the compression phase.

Phase II: Compressing the PRR-graph. When we reach Line 20, R is *boostable*. In practice, we observe that we can remove and merge a significant fraction of nodes and edges from R while keeping values of $f_R(B)$ and $f_R^-(B)$ for all $|B| \leq k$ same as before. Therefore, we compress boostable PRR-graphs to prevent the memory usage from becoming a bottleneck. Figure 3b shows the compressed result. The compression phase contains two steps. *First*, observing that nodes v_4 and v_7 are activated without boosting any node, we merge them into a single "super-seed" node. Then, we remove nodes v_6, v_8 and their incident edges because they are not on any paths from the super-seed node to the root node r. *Next*, observing that there are live paths from nodes v_0, v_1, v_2 and v_3 to root r, we remove their outgoing edges and directly link them to r. After doing so, we remove node v_0 because it is not on any path from the super-seed node to r. It is easy to see that the compression phase could be done by several passes of forward BFS from seeds or backward BFS from the root node. Thus, the compression phase runs in time linear to the number of uncompressed edges. Due to space limit, we omit the detailed description of the compression phase here, and refer interested readers to our previous conference paper [25] or our technical report [10].

B. PRR-Boost Algorithm

Algorithm 2 depicts `PRR-Boost`. It integrates PRR-graphs, the `IMM` algorithm and the *Sandwich Approximation* strategy.

Lines 1-3 utilize the `IMM` algorithm [8] with the PRR-graph generation to maximize the lower bound μ of Δ_S under the cardinality constraint of k. Line 4 greedily selects a set B_Δ of nodes with the goal of maximizing Δ_S, and we reuse PRR-graphs in $\mathcal{R}$ to estimate $\Delta_S(\cdot)$. Ideally, we should return $B_{\text{sa}} = \arg\max_{B \in \{B_\mu, B_\Delta\}} \Delta_S(B)$. Because evaluating $\Delta_S(B)$ is #P-hard, we select B_{sa} between B_μ and B_Δ with the larger *estimated* boost of influence in Line 5.

Algorithm 2: PRR-Boost$(G, S, k, \epsilon, \ell)$

$\ell' = \ell \cdot (1 + \log 3 / \log n)$
$\mathcal{R} \leftarrow$ `SamplingLB`$(G, S, k, \epsilon, \ell')$ // sampling in `IMM` [8] using the PRR-graph generation of Algo. 1
$B_\mu \leftarrow$ `NodeSelectionLB`$(\mathcal{R}, k)$ // maximize μ
$B_\Delta \leftarrow$ `NodeSelection`$(\mathcal{R}, k)$ // maximize Δ_S
$B_{\text{sa}} = \arg\max_{B \in \{B_\Delta, B_\mu\}} \hat{\Delta}_{\mathcal{R}}(B)$
return B_{sa}

Theorem 2 summarizes the theoretical result of `PRR-Boost`.

Theorem 2. *With a probability of at least $1 - n^{-\ell}$, `PRR-Boost` returns a $(1 - 1/e - \epsilon) \cdot \frac{\mu(B^*)}{\Delta_S(B^*)}$-approximate solution. Moreover, it runs in $O(\frac{EPT}{OPT_\mu} \cdot k \cdot (k + \ell)(n + m) \log n / \epsilon^2)$ expected time, where EPT is the expected number of edges explored for generating a random PRR-graph.*

Due to space limit, we omit the detailed analysis, and we refer interested readers to our previous paper [25] or our technical report [10]. The approximation ratio in Theorem 2 depends on $\frac{\mu(B^*)}{\Delta_S(B^*)}$, which should be close to one if the lower bound function $\mu(B)$ is close to the boost of influence $\Delta_S(B)$, when $\Delta_S(B)$ is large. Section VII demonstrates that $\mu(B)$ is indeed close to $\Delta_S(B)$ in real datasets.

C. The PRR-Boost-LB Algorithm

`PRR-Boost-LB` is a simplification of `PRR-Boost` where we return the node set B_μ as the final solution. Recall that the estimation of μ only relies on the critical node set C_R of each boostable PRR-graph R. In the first phase of the PRR-graph generation, if we only need to obtain C_R, there is no need to explore incoming edges of a node v if $d_r[v] > 1$. Moreover, in the compression phase, we can obtain C_R right after computing $d_S[\cdot]$ and we can terminate the compression earlier. The sampling phase of `PRR-Boost-LB` usually runs fasterbecause we only need to generate C_R for each boostable PRR-graph R. In addition, the memory usage is significantly lower than that for `PRR-Boost`, because the averaged number of "critical nodes" in a random boostable PRR-graph is small in practice. In summary, compared with `PRR-Boost`, `PRR-Boost-LB` has the same approximation factor but runs faster than `PRR-Boost`. We will compare `PRR-Boost` and `PRR-Boost-LB` by experiments in Section VII.

D. Discussion: The Budget Allocation Problem

A question one may raise is what is the best strategy if companies could freely decide how to allocation budget on both seeding and boosting. A heuristic method combing influence maximization algorithms and `PRR-Boost` is as follows. We could test different budget allocation strategy. For each allocation, we first identify seeds using any influence maximization algorithm, then we find boosted user by `PRR-Boost`. Finally, we could choose the budget allocation strategy leading to the largest boosted influence spread among all tested ones. In fact, the budget allocation problem could be much harder than the k-boosting problem itself, and its full treatment is beyond the scope of this study and is left as a future work.

VI. Boosting on Bidirected Trees

In this section, we study the k-boosting problem where influence propagates on *bidirected trees*.

On bidirected trees, the computation of the boost of influence spread becomes tractable. We are able to devise an efficient greedy algorithm and an approximation algorithm with a near-optimal approximation ratio. This demonstrates that the hardness of the k-boosting problem is partly due to the graph structure, and when we restrict to tree structures, we are able to find near-optimal solutions. Moreover, using near-optimal solutions as benchmarks enables us to verify that a greedy node selection method on trees in fact returns near-optimal solutions in practice. Besides, our efforts on trees will help to designing heuristics for the k-boosting problem on general graphs, or for other related problems in the future.

Bidirected trees. A directed graph G is a *bidirected tree* if and only if its underlying undirected graph (with directions and duplicated edges removed) is a tree. For simplicity of notation, we assume that every two adjacent nodes are connected by two edges, one in each direction. We also assume that nodes are activated with probability less than one, because nodes that will be activated for sure could be identified in linear time and they could be treated as seeds. Figure 4 shows an example of a bidirected tree. The existence of bidirected edges brings challenges to the algorithm design, because the influence may flow from either direction between a pair of neighboring nodes.

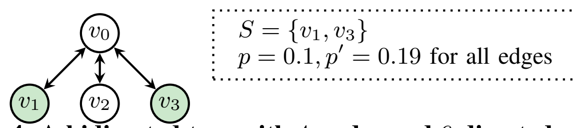


Fig. 4: A bidirected tree with 4 nodes and 6 directed edges.

In this section, we first present how to compute the exact boosted influence spread on bidirected trees, and a greedy algorithm `Greedy-Boost` based on it. Then, we present a rounded dynamic programming `DP-Boost`, which is a *fully polynomial-time approximation scheme*. `Greedy-Boost` is efficient but does not provide the approximation guarantee. `DP-Boost` is more computationally expensive but guarantees a near-optimal approximation ratio.

A. Computing the boosted influence spread

We first present how to compute the boosted influence spread in a bidirected tree. It serves as a building block for the greedy algorithm that iteratively selects nodes with the maximum marginal gain of the boosted influence spread.

We separate the computation into *three steps*. (1) We refer to the probability that a node gets activated (i.e., influenced) as its "*activation probability*". For every node u, we compute the increase of its activation probability when it is inserted into B. (2) If we regard a node u as the root of the tree, the remaining nodes could be categorized into multiple "subtrees", one for each neighbor of u. For every node u, we compute intermediate results that help us to determine the increase of influence spread in each such "subtree" if we insert u into B. (3) Based on the previous results, we compute $\sigma_S(B)$ and

$\sigma_S(B \cup \{u\})$ for every node u. If necessary, we are able to obtain $\Delta_S(B \cup \{u\})$ from $\sigma_S(B \cup \{u\}) - \sigma_S(\emptyset)$.

Notations. Let $p^B_{u,v}$ be the influence probability of an edge e_{uv} given that we boost nodes in B. Similarly, let $p^b_{u,v}$ be the influence probability of e_{uv}, where $b \in \{0,1\}$ indicates whether v is boosted. We use $N(u)$ to denote the set of neighbors of node u. Given neighboring nodes u and v, we use $G_{u\backslash v}$ to denote the subtree of G obtained by first removing node v and then removing all nodes not connected to u. To avoid cluttered notation, we slightly abuse the notation and keep using S and B to denote seed users and boosted users in $G_{u\backslash v}$, although some nodes in S or B may not be in $G_{u\backslash v}$.

Step I: Activation probabilities. For a node u, let $ap_B(u)$ be the activation probability of u when we boost B. For $v \in N(u)$, let $ap_B(u\backslash v)$ be the activation probability of node u in $G_{u\backslash v}$ when we boost B. For example, in Figure 4, suppose $B = \emptyset$, we have $ap_B(v_0) = 1 - (1-p)^2 = 0.19$ and $ap_B(v_0\backslash v_1) = p = 0.1$. We have the following lemma.

Lemma 3. *Given a node u, if u is a seed node (i.e., $u \in S$), we have $ap_B(u) = 1$ and $ap_B(u\backslash v) = 1$ for all $v \in N(u)$. Otherwise, we have*

$$ap_B(u) = 1 - \prod_{v \in N(u)} \left(1 - ap_B(v\backslash u) \cdot p^B_{v,u}\right), \tag{4}$$

$$ap_B(u\backslash v) = 1 - \prod_{w \in N(u)\backslash\{v\}} \left(1 - ap_B(w\backslash u) \cdot p^B_{w,u}\right), \forall v \in N(u), \tag{5}$$

$$ap_B(u\backslash v) = 1 - \big(1 - ap_B(u\backslash w)\big) \cdot \frac{1 - ap_B(w\backslash u) \cdot p^B_{w,u}}{1 - ap_B(v\backslash u) \cdot p^B_{v,u}}, \quad \forall v, w \in N(u), v \neq w. \tag{6}$$

Algorithm 3: Computing Activation Probabilities

```
1  Initialize ap_B(u\v), ∀u, v ∈ N(u) as "not computed"
2  foreach u ∈ V do
3      foreach v ∈ N(u) do
4          ComputeAP(u, v)                    // compute ap_B(u\v)
5  foreach u ∈ V do
6      if u ∈ S then  ap_B(u) ← 1
7      else  ap_B(u) ← 1 − ∏_{v∈N(u)} (1 − ap_B(v\u) · p^B_{v,u})
8  Procedure ComputeAP(u, v)
9      if we have not computed ap_B(u\v) then
10         if u ∈ S then ap_B(u\v) ← 1
11         else if we have not computed ap_B(u\w) for any
               w ∈ N(u)\{v} then
12             foreach w ∈ N(u)\{v} do  ComputeAP(w, u)
13             ap_B(u\v) ← 1 − ∏_{w∈N(u)\{v}} (1 − ap_B(w\u) p^B_{w,u})
14         else
15             Suppose we know ap_B(u\w) for w ∈ N(u)\{v}
16             ComputeAP(w, u)
17             ap_B(u\v) ← 1 − (1 − ap_B(u\w)) · (1 − ap_B(w\u) · p^B_{w,u}) / (1 − ap_B(v\u) · p^B_{v,u})
```

Algorithm 3 depicts how we compute activation probabilities. Lines 1-4 initialize and compute $ap_B(u\backslash v)$ for all neighboring nodes u and v. Lines 5-7 compute $ap_B(u)$ for all nodes u. The recursive procedure ComputeAP(u, v) for computing $ap_B(u\backslash v)$ works as follows. Line 9 avoids the recomputation. Line 10 handles the trivial case where node u is a seed. Lines 11-13 compute the value of $ap_B(u\backslash v)$ using Equation (5). Lines 14-17 compute $ap_B(u\backslash v)$ more efficiently using Equation (6), taking advantages of the known $ap_B(u\backslash w)$ and $ap_B(v\backslash u)$. Note that in Line 17, the value of $ap_B(v\backslash u)$ must have been computed, because we have computed $ap_B(u\backslash w)$, which relies on the value of $ap_B(v\backslash u)$. For a node u, given the values of $ap_B(w\backslash u)$ for all $w \in N(u)$, we can compute $ap_B(u\backslash v)$ for all $v \in N(u)$ in $O(|N(u)|)$. Then, for a node u, given values of $ap_B(w\backslash u)$ for all $w \in N(u)$, we can compute $ap_B(u)$ in $O(|N(u)|)$. Therefore, the time complexity of Algorithm 3 is $O(\sum_u |N(u)|) = O(n)$, where n is the number of nodes in the bidirected tree.

Step II: More intermediate results. Given that we boost B, we define $g_B(u\backslash v)$ as the gain of the influence spread in $G_{u\backslash v}$ when we add node u into the current seed set S. Formally, $g_B(u\backslash v)$ is defined as $g_B(u\backslash v) = \sigma^{G_{u\backslash v}}_{S\cup\{u\}}(B) - \sigma^{G_{u\backslash v}}_S(B)$, where $\sigma^{G_{u\backslash v}}_S(B)$ is the boosted influence spread in $G_{u\backslash v}$ when the seed set is S and we boost B. In Figure 4, we have $G_{v_0\backslash v_1} = G\backslash\{e_{01}, e_{10}\}$. Suppose $B = \emptyset$, when we insert v_0 into S, the boosted influence spread in $G_{v_0\backslash v_1}$ increases from 1.11 to 2.1, thus $g_B(v_0, v_1) = 0.99$. We compute $g_B(u\backslash v)$ for all neighboring nodes u and v by the following lemma.

Lemma 4. *Given a node u, if it is a seed node, we have $g_B(u\backslash v) = 0$. Otherwise, for any $v \in N(u)$, we have*

$$g_B(u\backslash v) = \Big(1 - ap_B(u\backslash v)\Big) \cdot \Big(1 + \sum_{w \in N(u)\backslash\{v\}} \frac{p^B_{u,w} \cdot g_B(w\backslash u)}{1 - ap_B(w\backslash u) \cdot p^B_{w,u}}\Big). \tag{7}$$

Moreover, for $v, w \in N(u)$ and $v \neq w$, we have

$$g_B(u\backslash v) = \big(1 - ap_B(u\backslash v)\big) \cdot \Big(\frac{g_B(u\backslash w)}{1 - ap_B(u\backslash w)} + \frac{p^B_{u,w} \cdot g_B(w\backslash u)}{1 - ap_B(w\backslash u) \cdot p^B_{w,u}} - \frac{p^B_{u,v} \cdot g_B(v\backslash u)}{1 - ap_B(v\backslash u) \cdot p^B_{v,u}}\Big). \tag{8}$$

Equation (7) shows how to compute $g_B(u\backslash v)$ by definition. Equation (8) provides a faster way to compute $g_B(u\backslash v)$, taking advantages of the previously computed values. Using similar algorithm in Algorithm 3, we are able to compute $g_B(u\backslash v)$ for all u and $v \in N(u)$ in $O(n)$.

Step III: The final computation. Recall that $\sigma_S(B)$ is the boosted influence spread, we have $\sigma_S(B) = \sum_{v \in V} ap_B(v)$. The following lemma shows how we compute $\sigma_S(B \cup \{u\})$.

Lemma 5. *Given a node u, if it is a seed node or a boosted node, we have $\sigma_S(B \cup \{u\}) = \sigma_S(B)$. Otherwise, we have*

$$\sigma_S(B \cup \{u\}) = \sigma_S(B) + \Delta ap_B(u) + \sum_{v \in N(u)} p^B_{u,v} \cdot \Delta ap_B(u\backslash v) \cdot g_B(v\backslash u), \tag{9}$$

where $\Delta ap_B(u) := ap_{B\cup\{u\}}(u) - ap_B(u) = 1 - \prod_{v \in N(u)} \left(1 - ap_B(v\backslash u) \cdot p'_{v,u}\right) - ap_B(u)$ *and* $\Delta ap_B(u\backslash v) := ap_{B\cup\{u\}}(u\backslash v) - ap_B(u\backslash v) = 1 - \prod_{w \in N(u)\backslash\{v\}} \big(1 - ap_B(w\backslash u) \cdot p'_{w,u}\big) - ap_B(u\backslash v)$.

The intuition behind Equation (9) is as follows. Let $V_{v\backslash u} \subseteq V$ be the set of nodes in $G_{v\backslash u} \subseteq G$. When we insert a node u into B, $\Delta ap_B(u)$ is the increase of the activation probability of u itself, and $p^B_{u,v} \cdot \Delta ap_B(u\backslash v) \cdot g_B(v\backslash u)$ is the increase of the number of influenced nodes in $V_{v\backslash u}$. The final step computes $\sigma_S(B)$ and $\sigma_S(B \cup \{u\})$ for all nodes u in $O(n)$.

Putting it together. Given a bidirected tree G and a set of boosted nodes B, we can compute $\sigma_S(B)$ and $\sigma_S(B \cup \{u\})$ for all nodes u in three steps. The total time complexity of all three steps is $O(n)$, where n is the number of nodes.

Greedy-Boost. Based on the linear-time computation of $\sigma_S(B \cup \{u\})$ for all nodes u, `Greedy-Boost` iteratively inserts into set B a node u that maximizes $\sigma_S(B \cup \{u\})$, until $|B| = k$. `Greedy-Boost` runs in $O(kn)$.

B. A Rounded Dynamic Programming

In this part, we present a rounded dynamic programming `DP-Boost`, which is a *fully polynomial-time approximation scheme*. `DP-Boost` requires that the tree has a root node. Any node could be assigned as the root node. Denote the root node by r. For ease of presentation, we assume here that every node has at most two children. We leave details about `DP-Boost` for general bidirected trees in our technical report [10].

Bottom-up exact dynamic programming. For notational convenience, we assume that r has a *virtual parent* r' and $p_{r'r} = p'_{r'r} = 0$. Given a node v, let V_{T_v} be the set of nodes in its subtree. Define $g(v, \kappa, c, f)$ as the maximum expected boost of nodes in V_{T_v} under the following conditions. (1) *Assumption:* The parent of node v is activated with probability f if we remove V_{T_v} from G. (2) *Requirement:* We boost at most κ nodes in V_{T_v}, and node v is activated with probability c if we remove nodes not in V_{T_v}. It is possible that for some node v, the second condition could never be satisfied (e.g., v is a seed but $c < 1$). In that case, we define $g(v, \kappa, c, f) := -\infty$.

By definition, $\max_c g(r, k, c, 0)$ is the maximum boost of the influence upon boosting at most k nodes. However, the exact dynamic programming is infeasible in practice because we may have to calculate $g(v, \kappa, c, f)$ for exponentially many choices of c and f. To tackle this problem, we propose a *rounded dynamic programming* and call it `DP-Boost`.

High level ideas. Let $\delta \in (0, 1)$ be a *rounding parameter*. We use $\lfloor x \rfloor_\delta$ to denote the value of x rounded down to the nearest multiple of δ. We say x is *rounded* if and only if it is a multiple of δ. For simplicity, we consider 1 as a *rounded* value. In `DP-Boost`, we compute a *rounded version* of $g(v, \kappa, c, f)$ only for *rounded values* of c and f. Then, the number of calculated entries would be polynomial in n and $1/\delta$. Let $g'(v, \kappa, c, f)$ be the rounded version of $g(v, \kappa, c, f)$, `DP-Boost` guarantees that $g'(v, \kappa, c, f) \le g(v, \kappa, c, f)$ and $g'(v, \kappa, c, f)$ gets closer to $g(v, \kappa, c, f)$ when δ decreases.

Definition 4 defines `DP-Boost`. An important remark is that $g'(\cdot)$ is equivalent to the definition of $g(\cdot)$ if we ignore all the rounding (i.e., assuming $\lfloor x \rfloor_\delta = x, \forall x$).

Definition 4 (`DP-Boost`). *Let v be a node. Denote the parent node of v by u.*

- *Base case. Suppose v is a leaf node. If $c \neq \mathbb{I}(v \in S)$, let $g'(v, \kappa, c, f) = -\infty$; otherwise, let $g'(v, \kappa, c, f) =$* $\max\left\{1-(1-c)(1-f \cdot p_{u,v}^{\mathbb{I}(\kappa>0)})-ap_\emptyset(v), 0\right\}$.
- *Recurrence formula. Suppose v is an internal node. If v is a seed node, we let $g'(v, \kappa, c, f) = -\infty$ for $c \neq 1$, and otherwise let*
$$g'(v, \kappa, 1, f) = \max_{\kappa=\sum \kappa_{v_i}} \sum_i g'(v_i, \kappa_{v_i}, c_{v_i}, 1).$$
If v is a non-seed node, we use $C'(v, \kappa, c, f)$ to denote the set of consistent subproblems of $g'(v, \kappa, c, f)$. Subproblems $(\kappa_{v_i}, c_{v_i}, f_{v_i}, \forall i)$ are consistent with $g'(v, \kappa, c, f)$ if they satisfy the following conditions:
$$b = \kappa - \sum_i \kappa_{v_i} \in \{0, 1\}, c = \Big\lfloor 1 - \prod_i \left(1 - c_{v_i} \cdot p_{v_i,v}^b\right) \Big\rfloor_\delta,$$
$$f_{v_i} = \Big\lfloor 1 - (1 - f \cdot p_{u,v}^b) \prod_{j \neq i} \left(1 - c_{v_j} \cdot p_{v_j,v}^b\right) \Big\rfloor_\delta, \forall i.$$
If $C'(v, \kappa, c, f) = \emptyset$, let $g'(v, \kappa, c, f) = -\infty$; otherwise, let
$$g'(v, \kappa, c, f) = \max_{\substack{(\kappa_{v_i}, f_{v_i}, c_{v_i}, \forall i) \\ \in C'(v, \kappa, c, f), \\ b = k - \sum_i \kappa_{v_i}}} \begin{pmatrix} \sum_i g'(v_i, \kappa_{v_i}, c_{v_i}, f_{v_i}) + \\ \max\{1-(1-c)(1-f \cdot p_{u,v}^b) - ap_\emptyset(v), 0\} \end{pmatrix}.$$

Rounding and relaxation. Because we compute $g'(v, \kappa, c, f)$ only for rounded c and f, in order to find consistent subproblems of $g'(v, \kappa, c, f)$ for an internal node v, we slightly relax the requirements of c and f_{v_i} as shown in Definition 4. Our relaxation guarantees that $g'(v, \kappa, c, f)$ is at most $g(v, \kappa, c, f)$. The rounding and relaxation may result in a loss of the boosted influence spread of the returned boosting set. However, as we shall show later, the loss is bounded.

`DP-Boost`. We first determine the rounding parameter δ by
$$\delta = \frac{\epsilon \cdot \max(LB, 1)}{\sum_{u \in V} \sum_{v \in V} p^{(k)}(u \rightsquigarrow v)}, \tag{10}$$
where LB is a lower bound of the optimal boost of influence, and $p^{(k)}(u \rightsquigarrow v)$ is defined as the probability that node u can influence node v given that we boost edges with top-k influence probability along the path. The value of LB could be obtained by `Greedy-Boost` in $O(kn)$. The denominator of Equation (10) could be computed via depth-first search starting from every node, each takes time $O(kn)$. Thus, we can obtain δ in $O(kn + kn^2) = O(kn^2)$. With the rounding parameter δ, `DP-Boost` computes the values of $g'(\cdot)$ bottom-up. For a leaf node v, it takes $O(k/\delta^2)$ to compute entries $g'(v, \kappa, c, f)$ for all κ, rounded c and rounded f. For an internal node v, we enumerate over all combinations of f, $b \in \{0, 1\}$, and κ_{v_i}, c_{v_i} for children v_i. For each combination, we can uniquely determine the values for κ, c and f_{v_i} for all children v_i, and update $g'(v, \kappa, c, f)$ accordingly. For an internal node v, the number of enumerated combinations is $O(k^2/\delta^3)$, hence we can compute all k/δ^2 entries $g'(v, \dots)$ in $O(k^2/\delta^3)$. The total complexity of `DP-Boost` is $O(kn^2 + n \cdot k^2/\delta^3) = O(k^2 n^7/\epsilon^3)$. To conclude, we have the following theorem about `DP-Boost`. The approximation guarantee is proved in our technical report.

Theorem 3. *Assuming the optimal boost is at least one, `DP-Boost` is a fully-polynomial time approximation scheme, it returns a $(1-\epsilon)$-approximate solution in $O(k^2 n^7/\epsilon^3)$.*

Refinements. We compute possible ranges of c and f for every node v. And we only compute $g'(v, k, c, f)$ for c and f within those ranges. For each node, the lower bound (resp. upper bound) of possible values of c and f are computed assuming that we do not boost any node (resp. we boost all nodes).

General `DP-Boost`. Computing $g'(v, \dots)$ for general bidirected trees is far more complicated. We list the main results here, and refer to interested readers to our technical report [10]. Given that the optimal boost of influence is at least one, `DP-Boost` returns a $(1-\epsilon)$-approximate solution in $O(k^2 n^9/\epsilon^3)$. In addition, if the number of children of every node is $O(1)$ DP-Boost runs in $O(k^2n^7/\epsilon^3)$.

VII. Experiments on General Graphs

We conduct extensive experiments using real social networks to evaluate PRR-Boost and PRR-Boost-LB, and show their superiority over intuitive baselines. Experiments were conduct on a Linux machine with an Intel Xeon E5620@2.4GHz CPU and 30 GB memory. The generation of PRR-graphs and the estimation of objective functions are parallelized with OpenMP and executed using eight threads.

Table 1: Statistics of datasets and seeds (all directed)

Description	Digg	Flixster	Twitter	Flickr
number of nodes (n)	28 K	96 K	323 K	1.45 M
number of edges (m)	200 K	485 K	2.14 M	2.15 M
average influence probability	0.239	0.228	0.608	0.013
influence of 50 influential seeds	2.5 K	20.4 K	85.3 K	2.3 K

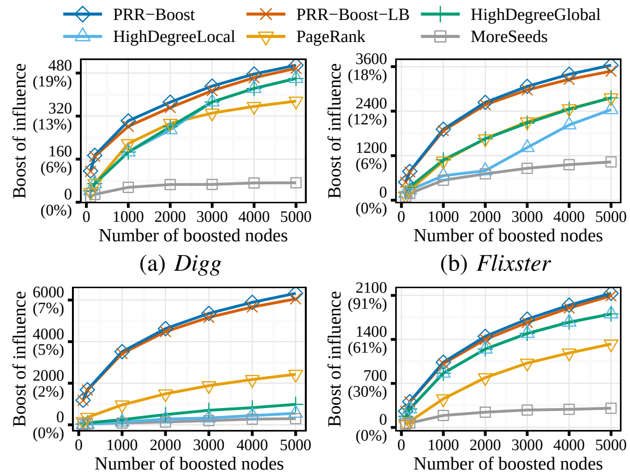
Datasets. We use four social networks *Flixster* [29], *Digg* [30], *Twitter* [31] and *Flickr* [32]. All dataset have directed social connections, and actions of users with timestamps (e.g., rating movies, voting for stories, re-tweeting URLs, marking favorite photos). We learn influence probabilities on edges using a widely accepted method by Goyal et al. [33]. We remove edges with zero influence probability and keep the largest weakly connected component. Table 1 summaries our datasets.

Boosted influence probabilities. To the best of our knowledge, no existing work quantitatively studies how influence among people changes respect to different kinds of "boosting strategies". For every edge e_{uv} we let the boosted influence probability p'_{uv} be $1-(1-p_{uv})^{\beta}$ (β>1). We refer to β as the *boosting parameter*. Due to the large number of combinations of parameters, we fix $\beta = 2$ unless otherwise specified. Intuitively, $\beta = 2$ indicates that every activated neighbor of a boosted node v has two independent chances to activate v. We also provide experiments showing the impacts of β.

Seed selection. We use the IMM method [8] to select 50 influential nodes. Table 1 summaries the influence spread of selected seeds. We also conduct experiments with randomly selected seeds. The setting maps to the situation where some users become seeds spontaneously. The experimental results given influential seeds and random seeds provide similar insights. Due to space limits, we leave the detailed results where seed users are randomly selected in our technical report [10].

Baselines. Because there is no existing algorithm applicable to the k-boosting problem, we compare our proposed algorithms with several heuristic baselines listed below.

- *HighDegreeGlobal*: Starting from an empty node set B, we iteratively adds a node with the highest *weighted degree* to B, until k nodes are selected. We use four definitions of the *weighted degree*. For a node $u \notin (S \cup B)$, they are: $\sum_{e_{uv}} p_{uv}$, $\sum_{e_{uv}, v\notin B} p_{uv}$, $\sum_{e_{vu}} [p'_{vu} - p_{vu}]$ and $\sum_{e_{vu}, v\notin B} [p'_{vu} - p_{vu}]$. Each definition outperforms others in some experiments, and we report the best result.
- *HighDegreeLocal*: *HighDegreeLocal* differs from *HighDegreeGlobal* in that we first consider nodes close to seeds. We first try to select k nodes among neighbors of seeds. If we can boost more nodes, we continue to select from nodes that are two-hops away from seeds. We repeat until k nodes are selected. We also report the best solution selected using four definitions of the *weighted degree*.
- *PageRank*: We use the *PageRank* baseline for the influence maximization problems [4]. When a node u has influence on v, it implies that node v "votes" for the rank of u. The transition probability on edge e_{uv} is $p_{vu}/\sum_{e_{wu}} p_{wu}$. The restart probability is 0.15. We compute the *PageRank* until two consecutive iteration differ for at most 10^{-4} in L_1 norm.
- *MoreSeeds*: We adapt the IMM method to select k more seeds with the goal of maximizing the final expected influence spread. We return the selected k seeds as the boosted nodes.



(a) *Digg* (b) *Flixster* (c) *Twitter* (d) *Flickr*

Fig. 5: Boost of the influence versus k.

We do not compare our algorithms to the greedy algorithm with Monte-Carlo simulations. Because it is extremely computationally expensive even for the classical influence maximization [1, 7].

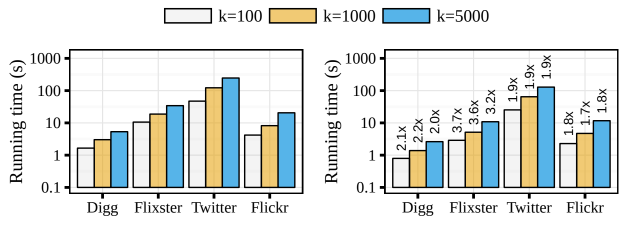
Settings. For PRR-Boost and PRR-Boost-LB, we let $\epsilon = 0.5$ and $\ell = 1$ so that both algorithms return $(1-1/e-\epsilon)\cdot\frac{\mu(B^*)}{\Delta_S(B^*)}$-approximate solution with probability at least $1 - 1/n$. To enforce fair comparison, for all algorithms, we evaluate the boost of influence spread by $20,000$ Monte-Carlo simulations.

A. Performance evaluation

In this part, we evaluate the performance of our algorithms. We report results where the seeds are 50 influential nodes. We run each experiment five times and report the average results.

Quality of solution. Figure 5 compares the solutions returned by different algorithms. Both PRR-Boost and PRR-Boost-LB outperform other baselines. PRR-Boost always return the best solution, and PRR-Boost-LB returns solutions with slightly lower but comparable quality. In addition, *MoreSeeds* returns solutions with the lowest quality. This is because nodes selected by *MoreSeeds* are typically in the part of graph not covered by the existing seeds so that they could generate larger marginal influence. In contrast, boosting nodes are typically close to seeds to make the boosting result more effective. Thus, our empirical result further shows that k-boosting problem differs significantly from the influence maximization problem.

Running time. Figure 6 shows the running time. The running time of both PRR-Boost and PRR-Boost increases when k increases. This is mainly because the number of random



(a) PRR-Boost (b) PRR-Boost-LB

Fig. 6: Running time.

Table 2: Memory usage and compression ratio. Numbers in parentheses are memory usage for storing PRR-graphs.

k	Dataset	PRR-Boost		PRR-Boost-LB
		Compression Ratio	Memory (GB)	Memory (GB)
100	*Digg*	1810.32 / 2.41 = 751.79	0.07 (0.01)	0.06 (0.00)
	Flixster	3254.91 / 3.67 = 886.90	0.23 (0.05)	0.19 (0.01)
	Twitter	14343.31 / 4.62 = 3104.61	0.74 (0.07)	0.69 (0.02)
	Flickr	189.61 / 6.86 = 27.66	0.54 (0.07)	0.48 (0.01)
5000	*Digg*	1821.21 / 2.41 = 755.06	0.09 (0.03)	0.07 (0.01)
	Flixster	3255.42 / 3.67 = 886.07	0.32 (0.14)	0.21 (0.03)
	Twitter	14420.47 / 4.61 = 3125.37	0.89 (0.22)	0.73 (0.06)
	Flickr	189.08 / 6.84 = 27.64	0.65 (0.18)	0.50 (0.03)

PRR-graphs required increases. Figure 6 also shows that the running time is in general proportional to the number of nodes and edges for *Digg*, *Flixster* and *Twitter*, *but not for Flickr*. This is mainly because of the significantly smaller average influence probabilities on *Flickr*, and the accordingly lower cost for generating a random PRR-graph (i.e., EPT) as we will show shortly in Table 2. In Figure 6, we also label the speedup of `PRR-Boost-LB` compared with `PRR-Boost`. Together with Figure 5, we can see that `PRR-Boost-LB` returns solutions with quality comparable to `PRR-Boost` but runs faster. Because our algorithms consistently outperform heuristic methods with no performance guarantee in all tested cases, we do not compare the running time of our algorithms with heuristic methods to avoid cluttering the results.

Effectiveness of the compression phase. Table 2 shows the *compression ratio* of PRR-graphs and memory usages of our algorithms, demonstrating the importance of compressing PRR-graphs. The compression ratio is the ratio between the average number of uncompressed edges and average number of edges after compression in boostable PRR-graphs. Besides the total memory usage, we also show in parenthesis the memory usage for storing boostable PRR-graphs. It is measured as the additional memory usage starting from the generation of the first PRR-graph. The compression ratio is high in practice for two reasons. *First*, many nodes visited in the first phase cannot be reached by seeds. *Second*, among the remaining nodes, many of them can be merged into the super-seed node, and most non-super-seed nodes are be removed because they are not on paths from the super-seed node to the root node. The high compression ratio and the memory used for storing compressed PRR-graphs show that the compression phase is indispensable. For `PRR-Boost-LB`, the memory usage is *much lower* because we only store critical nodes of boostable PRR-graphs and most boostable PRR-graph only has a few critical nodes in our experiments with $\beta = 2$.

Approximation factors. Recall that the approximate ratio of `PRR-Boost` and `PRR-Boost-LB` depends on $\frac{\mu(B^*)}{\Delta_S(B^*)}$. The closer to one the ratio is, the better the approximation guarantee is. With B^* being unknown due to the NP-hardness of the

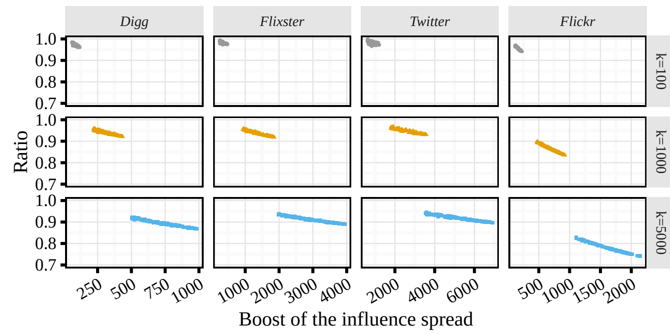
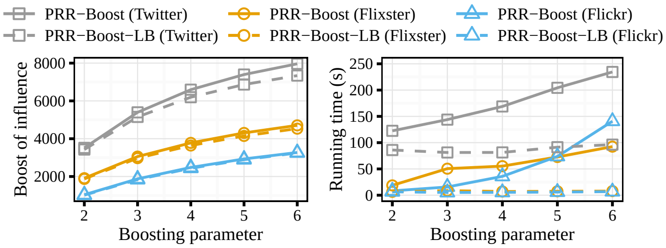


Fig. 7: Sandwich Approximation: $\frac{\mu(B)}{\Delta_S(B)}$.

problem, we show the ratio when the boost is relatively large. We obtain 300 sets of k boosted nodes by replacing a random number of nodes in B_{sa} by other non-seed nodes, where B_{sa} is the solution returned by `PRR-Boost`. For a set B, we use PRR-graphs generated for finding B_{sa} to estimate $\frac{\mu(B)}{\Delta_S(B)}$. Figure 7 shows the ratios for generated sets B as a function of $\Delta_S(B)$ for varying k. Because we intend to show the ratio when the boost of influence is large, we ignore points corresponding to sets whose boost of influence is less than 50% of $\Delta_S(B_{sa})$. For all datasets, the ratio is above 0.94, 0.83 and 0.74 for $k = 100, 1000, 5000$, respectively. The ratio is closer to one when k is smaller. In practice, most boostable PRR-graphs have critical nodes. When k is smaller, a node set B that could result in a large boost of influence tends to contain more nodes that are critical in many boostable PRR-graphs. For a given PRR-graph R, if B contains its critical nodes, we have $f_R^-(B) = f_R(B)$. Therefore, when k is smaller, $\frac{\mu(B)}{\Delta_S(B)} = \frac{\mathbb{E}[f_R^-(B)]}{\mathbb{E}[f_R(B)]}$ tends to be closer to one.

Effects of the boosted influence probabilities. The larger the *boosting parameter* β is, the larger the optimal boost is. Figure 8 shows the effects of β on the boost of influence and the running time. Figure 8a shows that `PRR-Boost` and `PRR-Boost-LB` return comparable solutions with varying β for *Flixster* and *Flickr*. For *Twitter*, we consider the slightly degenerated performance of `PRR-Boost-LB` acceptable because `PRR-Boost-LB` runs significantly faster. Figure 8b shows that the running time of `PRR-Boost` increases when β increases, but the running time of `PRR-Boost-LB` remains almost unchanged. Thus, `PRR-Boost-LB` is more scalable to *larger boosted influence probabilities on edges*. In fact, when β increases, a random PRR-graph tends to be larger. The running time of `PRR-Boost` increases mainly because the cost for PRR-graph generation increases. However, when β increases, we observe that the cost for obtaining "critical nodes" for a random PRR-graph does not change much, thus the running time of `PRR-Boost-LB` remains almost unchanged. We also check the approximation ratio of the sandwich approximation strategy with varying boosting parameters. For every dataset, when the boosting parameter increases, the ratio of $\frac{\mu(B)}{\Delta_S(B)}$ for large $\Delta_S(B)$ remains almost the same. This suggests that both our proposed algorithms remain effective when we increase the boosted influence probabilities on edges.

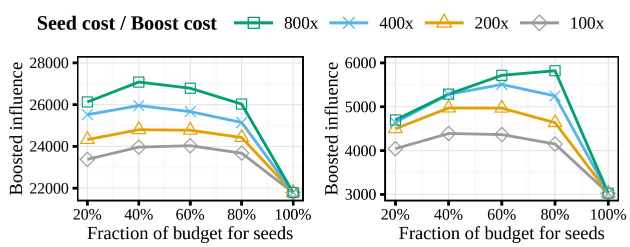


(a) *Boost of influence* (b) *Running time*

Fig. 8: Effects of the boosting parameter ($k = 1000$).

B. *Budget allocation between seeding and boosting*

In this part, we explore the budget allocation problem where a company can decide both the number of seeders and the number of users to boost. We assume that we can target 100 users as seeds with all the budget, and targeting a seed user costs 100 to 800 times as much as boosting a user. For example, suppose targeting a seeder costs 100 times as much as boosting a user, we can boost 100 more users if we target one less seed user. We explore the expected influence spreads with different budget allocations. Given the budget allocationwe first identify influential seeds using the `IMM` method, then we use `PRR-Boost` to select boosted users.



(a) *Flixster* (b) *Flickr*

Fig. 9: Budget allocation between seeding and boosting.

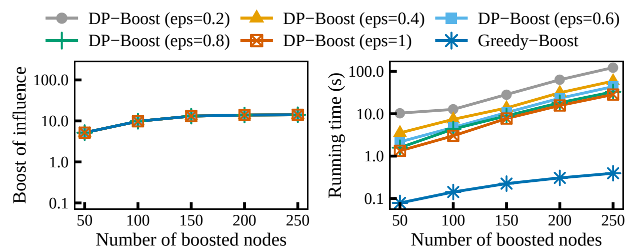
Figure 9 shows the results for *Flixster* and *Flickr*. Spending a mixed budget among initial adopters and boosting users achieves higher final influence spread than spending all budget on initial adopters. For example, for cost ratio of 800 between seeding and boosting, if we choose 80% budget for seeding and 20% for boosting, we would achieve around 20% and 92% higher influence spread than pure seeding, for *Flixster* and *Flickr* respectively. Moreover, the best budget mix is different for different networks and different cost ratio, suggesting the need for specific tuning and analysis for each case.

VIII. Experiments on Bidirected Trees

In this section, we show experimental results of `DP-Boost` and `Greedy-Boost` for bidirected trees. We show that `DP-Boost` efficiently approximates the k-boosting problem for bidirected trees with thousands of nodes. And, `Greedy-Boost` returns near-optimal solutions.

For a given number of nodes n, we construct a complete (undirected) binary tree with n nodes, then we replace each undirected edge by two directed edges, one in each direction. We assign influence probabilities on edges according to the *Trivalency* model. For each edge e_{uv}, p_{uv} is randomly chosen from $\{0.001, 0.01, 0.1\}$. Moreover, for every edge e_{uv}, let $p'_{uv} = 1-(1-p_{uv})^2$. For every tree, we select 50 seeds using the `IMM` method. We compare `Greedy-Boost` and `DP-Boost`. The boost of influence of the returned sets are computed exactly. We run each experiment five times with randomly assigned influence probabilities and report the average results.

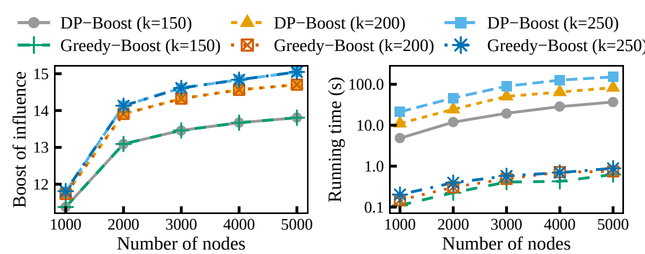
`Greedy-Boost` v.s. `DP-Boost` with varying ϵ. For `DP-Boost`, the value of ϵ controls the tradeoff between the accuracy and computational costs. Figure 10 shows that, for `DP-Boost`, the running time decreases dramatically when ϵ increases, but the boost is almost unaffected. Because `DP-Boost` returns $(1-\epsilon)$-approximate solutions, it provides a benchmark for the greedy algorithm. Figure 10a shows that the greedy algorithm `Greedy-Boost` returns near-optimal solutions in practice. Moreover, Figure 10b shows `Greedy-Boost` is orders of magnitude faster than `DP-Boost` with $\epsilon = 1$ where the theoretical guarantee is in fact lost.



(a) Boost of influence (b) Running time

Fig. 10: The greedy algorithm versus the rounded dynamic programming on random bidirected trees with 2000 nodes.

`Greedy-Boost` versus `DP-Boost` with varying tree sizes. Figure 11 compares `Greedy-Boost` and `DP-Boost` ($\epsilon = 0.5$) for trees with varying sizes. Figure 11a suggests that `Greedy-Boost` always return near-optimal solutions on trees with varying sizes. Figure 11b demonstrates the efficiency of `Greedy-Boost`. Results for smaller values of k are similar.



(a) Boost of influence (b) Running time

Fig. 11: The greedy algorithm versus the rounded dynamic programming on random bidirected trees with varies sizes.

IX. Conclusion

In this work, we address a novel k-boosting problem that asks how to *boost* the influence spread by offering k users incentives so that they are more likely to be influenced by friends. For the k-boosting problem on general graphs, we develop efficient approximation algorithms, `PRR-Boost` and `PRR-Boost-LB`, that have *data-dependent* approximation factors. Both `PRR-Boost` and `PRR-Boost-LB` are delicate integration of *Potentially Reverse Reachable Graphs* and the state-of-the-art techniques for influence maximization problems. For the k-boosting problem on bidirected trees, we present an efficient greedy algorithm `Greedy-Boost` based on a linear-time exact computation of the *boost of influence spread*, and we also repsent a fully polynomial-time approximation scheme `DP-Boost`. We conduct extensive experiments on real datasets to evaluatie `PRR-Boost` and `PRR-Boost-LB`. Results demonstrate the superiority of our proposed algorithms over intuitive baselines. Compared with `PRR-Boost`, experimental results show that `PRR-Boost-LB` returns solution with comparable quality but has significantly lower computational costs. On real social networks, we also explore the scenario where we are allowed to determine how to spend the limited budget on both targeting initial adopters and boosting users. Experimental results demonstrate the importance of studying the problem of targeting initial adopters and boosting users with a mixed strategy. We also conduct experiments on synthetic bidirected to show the efficiency and effectiveness of our proposed algorithms `Greedy-Boost` and `DP-Boost` for trees. In particular, we show via experiments that `Greedy-Boost` is extremely efficient and returns near-optimal solutions in practice.

References

[1] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *SIGKDD*, 2003, pp. 137–146.

[2] T. Carnes, C. Nagarajan, S. M. Wild, and A. van Zuylen, "Maximizing influence in a competitive social network: A follower's perspective," in *EC*, 2007, pp. 351–360.

[3] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *SIGKDD*, 2009, pp. 199–208.

[4] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *SIGKDD*, 2010, pp. 1029–1038.

[5] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *ICDM*, 2010, pp. 88–97.

[6] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *SODA*, 2014, pp. 946–957.

[7] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *SIGMOD*, 2014, pp. 75–86.

[8] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *SIGMOD*, 2015, pp. 1539–1554.

[9] "Global trust in advertising," http://www.nielsen.com/us/en/insights/reports/2015/global-trust-in-advertising-2015.html, accessed: 2016-09-18.

[10] Y. Lin, W. Chen, and J. C. S. Lui, "Boosting information spread: An algorithmic approach," *arXiv:1602.03111 [cs.SI]*, 2016.

[11] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *SIGKDD*, 2007, pp. 420–429.

[12] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: Optimizing the greedy algorithm for influence maximization in social networks," in *WWW*, 2011, pp. 47–48.

[13] W. Chen, L. V. S. Lakshmanan, and C. Castillo, "Information and influence propagation in social networks," *Synthesis Lectures on Data Management*, 2013.

[14] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *ICDM*, 2016, pp. 695–710.

[15] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *ICDM*, 2012, pp. 918–923.

[16] S. Bharathi, D. Kempe, and M. Salek, "Competitive influence maximization in social networks," in *International Workshop on Web and Internet Economics*, 2007, pp. 306–311.

[17] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt, "Recommendations to boost content spread in social networks," in *WWW*, 2012, pp. 529–538.

[18] D.-N. Yang, H.-J. Hung, W.-C. Lee, and W. Chen, "Maximizing acceptance probability for active friending in online social networks," in *SIGKDD*, 2013, pp. 713–721.

[19] S. Antaris, D. Rafailidis, and A. Nanopoulos, "Link injection for boosting information spread in social networks," *Social Network Analysis and Mining*, vol. 4, no. 1, pp. 1–16, 2014.

[20] D. Rafailidis, A. Nanopoulos, and E. Constantinou, "with a little help from new friends: Boosting information cascades in social networks based on link injection," *Journal of Systems and Software*, vol. 98, pp. 1–8, 2014.

[21] D. Rafailidis and A. Nanopoulos, "Crossing the boundaries of communities via limited link injection for information diffusion in social networks," in *WWW*, 2015, pp. 97–98.

[22] Y. Yang, X. Mao, J. Pei, and X. He, "Continuous influence maximization: What discounts should we offer to social network users?" in *ICDM*, 2016, pp. 727–741.

[23] W. Lu, W. Chen, and L. V. Lakshmanan, "From competition to complementarity: comparative influence diffusion and maximization," *VLDB*, vol. 9, no. 2, pp. 60–71, 2015.

[24] W. Chen, F. Li, T. Lin, and A. Rubinstein, "Combining traditional marketing and viral marketing with amphibious influence maximization," in *EC*, 2015, pp. 779–796.

[25] Y. Lin, W. Chen, and J. C. S. Lui, "Boosting information spread: An algorithmic approach," in *ICDE*, 2017, pp. 883–894.

[26] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[27] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[28] Y. Lin and J. C. Lui, "Analyzing competitive influence maximization problems with partial information: An approximation algorithmic framework," *Performance Evaluation*, vol. 91, pp. 187–204, 2015.

[29] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *RecSys*, 2010, pp. 135–142.

[30] T. Hogg and K. Lerman, "Social dynamics of digg," *EPJ Data Science*, vol. 1, no. 1, p. 5, 2012.

[31] N. O. Hodas and K. Lerman, "The simple rules of social contagion," *Scientific reports*, vol. 4, 2014.

[32] M. Cha, A. Mislove, and K. P. Gummadi, "A measurement-driven analysis of information propagation in the flickr social network," in *WWW*, 2009, pp. 721–730.

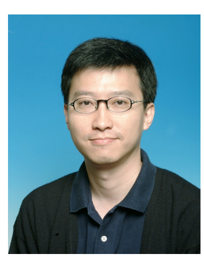
[33] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.

Yishi Lin received her B.E. degree from the School of Computer Science and Technology at the University of Science and Technology of China in 2013. She is currently a Ph.D. candidate in the Department of Computer Science and Engineering at the Chinese University of Hong Kong, under the supervision of Prof. John C.S. Lui.

Her main interests include social networks (e.g., information diffusion models and applications) and graph algorithms.

Wei Chen received his Ph.D. degree from the Department of Computer Science, Cornell University, Ithaca, New York.

He is a Senior Researcher in Microsoft Research Asia. He is also an Adjunct Professor of Tsinghua University, and an Adjunct Researcher of the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include social and information networks, online learning, algorithmic game theory, Internet economics, distributed computing, and fault tolerance. He is a member of the IEEE.



John C.S. Lui received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA.

He is currently a Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong. His current research interests include communication networks, network/system security, network economics, network sciences, cloud computing, large scale distributed systems, and performance evaluation theory.

Prof. Lui serves in the editorial board of the IEEE/ACM Transactions on Networking, the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the *Journal of Performance Evaluation*, and the *International Journal of Network Security*. He is an Elected Member of the IFIP WG 7.3, a fellow of ACM, a fellow of IEEE, and Croucher Senior Research Fellow.