Random-walk Domination in Large Graphs

Rong-Hua Li [#], Jeffrey Xu Yu [*†], Xin Huang [*], and Hong Cheng [*]

[#] *Guangdong Province Key Laboratory of Popular High Performance Computers, Shenzhen University, China*

[*] *The Chinese University of Hong Kong*

[†] *Key Laboratory of High confidence Software Technologies Ministry of Education (CUHK Sub-Lab)*

{rhli, yu, xhuang, hcheng}@se.cuhk.edu.hk

***Abstract*—We introduce and formulate two types of random-walk domination problems in graphs motivated by a number of applications in practice (e.g., item-placement problem in online social networks, Ads-placement problem in advertisement networks, and resource-placement problem in P2P networks). Specifically, given a graph G, the goal of the first type of random-walk domination problem is to target k nodes such that the total hitting time of an L-length random walk starting from the remaining nodes to the targeted nodes is minimized. The second type of random-walk domination problem is to find k nodes to maximize the expected number of nodes that hit any one targeted node through an L-length random walk. We prove that these problems are two special instances of the submodular set function maximization with cardinality constraint problem. To solve them effectively, we propose a dynamic-programming (DP) based greedy algorithm which is with near-optimal performance guarantee. The DP-based greedy algorithm, however, is not very efficient due to the expensive marginal gain evaluation. To further speed up the algorithm, we propose an approximate greedy algorithm with linear time complexity w.r.t. the graph size and also with near-optimal performance guarantee. The approximate greedy algorithm is based on carefully designed random walk sampling and sample-materialization techniques. Extensive experiments demonstrate the effectiveness, efficiency and scalability of the proposed algorithms.**

I. Introduction

Given a graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, how can we target k nodes such that the targeted nodes can be easily reached by the remaining nodes through an L-length random walk where the random walk moves at most L hops? And how can we find k nodes so as to maximize the expected number of nodes that hit any one targeted node by the L-length random walk? We refer to these two problems as two types of random-walk domination problems, because a node hitting any one targeted node can be regarded as that the targeted nodes dominate such a node by an L-length random walk. Intuitively, the random-walk domination problems are very difficult because there are C_n^k possible solutions and for each solution one should perform $n-k$ calculations to check (or record the hitting time) whether or not a node reaches any one targeted node by the L-length random walk. These problems are encountered in many data mining and social network applications. Some of them are discussed as follows.

A. Motivation

Item-placement problem in online social networks: Recently, social networking service has become an important medium for users to search for information online [1], [2], [3], [4]. In many online social networks, users primarily rely on a social process called social browsing [1], [2] to find information. Specifically, social browsing depicts a process that the users in a social network find information along their social ties [1], [2]. For example, in an online photo-sharing website Flickr (http://www.flickr.com), a user can view his friends' photos via their home-pages. Once the user arrives at one of his friends' home-page, then he is also able to apply the same way to browse the photos created by his friend's friends. Clearly, the next home-page that a user will visit depends on the current home-page where he is browsing. Therefore, a user's social browsing process can be regarded as a random-walk process on the social network. Furthermore, users typically have an *implicit* time-limit to browse the others' home-pages because users cannot browse an infinite number of home-pages. As a result, we can model the social browsing process as an L-length random walk by assuming that each user visits at most L home-pages in a social browsing process.

Based on the social browsing process, two interesting questions are: (1) how to place items (e.g., news, photos, videos, and applications) on a small fraction of users in a social network so that the other users can easily discover such items via social browsing, and (2) how to place items on a small fraction of users so that as many users as possible can search for such items by social browsing. Let us consider a more concrete application in Facebook social network. Assume that an application developer wants to popularize his Facebook application. Then, he may select a small fraction of users, say k users, to install his application for free. Note that in Facebook, if a user has installed an application, then his friends can view it by browsing his home-page (social browsing). Now the question is how to select k users so that the other users can easily find such an application (or as many users as possible can find such an application). Since we model the social browsing process as an L-length random walk, these questions are actually two instances of the random-walk domination problems.

Optimizing Ads-placement in advertisement networks: A similar example is also encountered in online advertisement networks, where an advertisement developer would like to place an advertisement (Ad) on a small fraction of users (he may pay for these users) so that it can be easily found by other users via social browsing (or as many users as possible can find such an Ad by social browsing). Likewise, we can model the user-information-finding process in the advertisement networks

978-1-4799-2555-1/14/$31.00 © 2014 IEEE

as an L-length random walk. As a consequence, these problems become two instances of the random-walk domination problems.

Accelerating resource search in P2P networks: The study of the random-walk domination problems could also be beneficial to accelerate resource search in P2P networks. Specifically, in a P2P network, how to place resources on a small number of peers such that other peers can easily search for such resources via some pre-specified search strategies. In P2P networks, a commonly-used search strategy is based on random walk [5]. Moreover, a resource-search process in P2P networks typically has a lifespan. That is to say, the resource-search process generally has a time-limit or hop-limit. Therefore, we can also model the resource-search process in P2P networks as an L-length random walk, i.e., the resource-search process searches at most L peers in its lifespan. Clearly, based on the L-length random walk, the resource-placement problem in P2P network is an instance of the random-walk domination problem. Therefore, using the results of the random-walk domination problems can accelerate the resource search in P2P networks.

B. Our main contributions

This paper presents the first study on the random-walk domination problems. Our goal is to formulate the random-walk domination problems and devise efficient and effective algorithms for these problems which can be directly applied to all the above applications. In particular, we first formulate two types of random-walk domination problems described above as two discrete optimization problems respectively. Then, we prove that these two problems are the instances of submodular set function maximization with cardinality constraint problem [6]. In general, such problems are NP-hard [6]. Therefore, we resort to develop approximate algorithms to solve them efficiently. To this end, we devise a dynamic programming (DP) based greedy algorithm to solve these problems effectively. By a well-known result [6], the DP-based greedy algorithm achieves a $1 - 1/e$ (≈ 0.63) approximation factor. However, the time complexity of the DP-based greedy algorithm is over cubic w.r.t. the network size, thus it can only work well in the small graphs. To overcome this drawback, we develop an approximate greedy algorithm based on carefully designed random walk sampling and sample materialization techniques. The time and space complexity of the approximate greedy algorithm are linear w.r.t. the graph size, thus it can be scalable to handle large graphs. Moreover, we show that the approximate greedy algorithm is able to achieve a $1 - 1/e - \epsilon$ approximation factor, where ϵ is a very small constant. Finally, we conduct comprehensive experiments to evaluate the proposed algorithms. The results indicate that the performance of the approximate greedy algorithm is very similar to that of the DP-based greedy algorithm, and it is substantially better than the baselines. In addition, the results confirm that the approximate greedy algorithm scales linearly w.r.t. the graph size.

The rest of this paper is organized as follows. Below, we will briefly review the existing studies that are related to ours. After that, we formulate the random-walk domination problems in Section II. We propose the DP-based greedy algorithm and the approximate greedy algorithm for solving the random-walk domination problems in Section III. Extensive experiments are reported in Section IV. We conclude this work and discuss some future directions in Section V.

C. Related work

Our problems are closely related to the dominating set problem in graphs. The dominating set problem is a classic NP-hard problem which has been well-studied in the literature [7], [8]. There is an $O(\log n)$ approximate algorithm for solving this problem efficiently [8]. Moreover, it has turned out that such an approximation factor is optimal unless P=NP [8], [9]. The dominating set has been widely used in the networking community due to a large number of applications in wireless sensor networks [10], [11] and other Ad Hoc networks [12], [13]. Recently, many different extensions of the dominating set problem have also been investigated. Notable examples include the distributed dominating set problem [14], the connected dominating set problem [15], [16], [11], [12], the Steiner connected dominating set problem [16], and the k-dominating set problem [8], [10]. All of these extensions are based on the traditional definition of domination [7] where the nodes deterministically dominate their immediate (or L-hop) neighbors. In our work, the problems are based on a newly defined concept called random-walk domination in which the targeted nodes dominate an L-hop neighbor if and only if such a neighbor-node hits at least one targeted node through an L-length random walk.

Our work is also related to the submodular set function maximization problem [6]. Generally, the problem of submodular function maximization subject to cardinality constraint is NP-hard. Nemhauser et al. [6] propose a greedy algorithm with $1 - 1/e$ approximation factor to settle this issue. Recently, many applications have been formulated as the submodular set function maximization subject to cardinality constraint problem. Some notable examples include the classic maximal k coverage problem [9], the influence maximization problem in social networks [17], the outbreak detection problem in networks [18], the observation selection and sensor placement problem [19], the document summarization problem [20], the privacy preserving data publishing problem [21], and the diversified ranking problem [22], [23]. All of these problems can be approximately solved by the greedy algorithm given in [6]. Here we study two random-walk domination problems in graphs, and we show that both of them can also be formulated as the submodular set function maximization with cardinality constraint problem. Also, we present a near-optimal approximate greedy algorithm to solve them efficiently.

II. Problem Statement

Consider an undirected and un-weighted graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Although we only

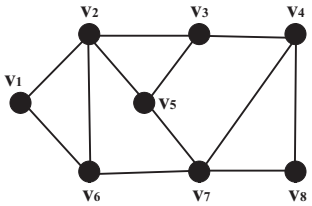

Fig. 1. Running example.

focus on undirected and un-weighted graphs in this paper, the proposed techniques can also be easily extended to directed and weighted graphs. Below, we first introduce some useful concepts of random walk on graphs, and then we formulate two different types of random-walk domination problems.

A random walk on an undirected and un-weighted graph denotes the following process. Given an undirected and un-weighted graph G and a starting node u, the random walk picks a neighbor-node of u uniformly at random and moves to this neighbor-node, and then follows this way recursively [24]. In this work, we concentrate on a general random walk model called L-length random walk where the path length of the random walk is bounded by a nonnegative integer L [25]. We emphasize that the traditional random walk is a special case of the L-length random walk by setting the parameter L to infinity. Moreover, as discussed in Section I, many practical applications should be modeled by the L-length random walk.

Next, we define an important concept called hitting time of the L-length random walk. In particular, the hitting time between the source and targeted nodes measures the expected number of hops taken by an L-length random walk starting from the source node and ending at the targeted node for the first time. Formally, denote by Z_u^t the position of an L-length random walk starting from node u at discrete time t. Let T_{uv}^L be a random variable defined as

$$T_{uv}^L \triangleq \min\{\min\{t : Z_u^t = v, t \geq 0\}, L\}. \tag{1}$$

Then, the hitting time between node u and v denoted by h_{uv}^L is defined as the expectation of T_{uv}^L, i.e., $h_{uv}^L \triangleq \mathbb{E}[T_{uv}^L]$. By this definition, the following lemma immediately holds.

Lemma 2.1: For any nodes u and v, the hitting time h_{uv}^L is bounded by L, i.e., $h_{uv}^L = \mathbb{E}[T_{uv}^L] \leq L$.

The following theorem shows that the exact hitting time between two nodes can be computed recursively.

Theorem 2.2: Let d_u be the degree of node u and $N(u)$ be the set of neighbor nodes of u. Further, let $p_{uw} = 1/d_u$ be the transition probability for $w \in N(u)$, and $p_{uw} = 0$ for $w \notin N(u)$. Then, for any nodes u and v, h_{uv}^L can be recursively computed by

$$h_{uv}^L = \begin{cases} 0, u = v \\ 1 + \sum_{w \in V} p_{uw} h_{wv}^{L-1}, u \neq v, \end{cases} \tag{2}$$

where h_{wv}^{L-1} denotes the hitting time between w and v based on an $(L-1)$-length random walk.

Proof: See Appendix. ∎

Remark: Sarkar and Moore in [25] define the hitting time of the L-length random walk in a recursive manner which is given in Eq. (2). Note that our definition is more intuitive than their definition because our definition is based on Eq. (1) in which the hitting time is "explicitly" bounded by L. In the above theorem, we show that our definition of hitting time can be computed by the same recursive equation (Eq. (2)) as defined in [25]. Furthermore, by our definition, the hitting time is an expectation of the random variable T_{uv}^L, thus it is very easy to design a sampling-based algorithm to estimate the hitting time.

A. The random-walk domination problems

Based on the L-length random walk model, we introduce two types of random-walk domination problems in graphs. We describe the first type of random-walk domination problem as follows. Denote by $S \subseteq V$ a subset of nodes. Assume that there is an L-length random walk starting from a node $u \in V$. If such a random walk reaches any one node in S at any discrete time in $[0, L]$, we call that u hits S or S dominates u by an L-length random walk. For example, consider the graph shown in Fig. 1. Suppose that $S = \{v_5, v_6\}$ and $L = 4$. There is an L-length random walk $(v_1, v_2, v_3, v_2, v_6)$ starting from v_1. Since this random walk reaches node v_6 and $v_6 \in S$, we call that v_1 hits S or S dominates v_1. Clearly, if $u \in S$, then u hits S. Below, we define another important concept called *generalized hitting time* which measures the hitting time from a single source node to a set of targeted nodes S. Specifically, let T_{uS}^L be a random variable defined as

$$T_{uS}^L \triangleq \min\{\min\{t : Z_u^t \in S, t \geq 0\}, L\}. \tag{3}$$

By this definition, T_{uS}^L denotes the number of hops of that the L-length random walk starting at u hits any one node in S for the first time. Note that if $S = \emptyset$, we have $T_{uS}^L = L$. This is because if S is an empty set, u cannot hit S, and thereby $\min\{t : Z_u^t \in S, t \geq 0\}$ is infinity. In addition, if $L = 0$, $T_{uS}^L = 0$ because $\min\{t : Z_u^t \in S, t \geq 0\} \geq 0$. Based on T_{uS}^L, the generalized hitting time from u to S denoted by h_{uS}^L is defined as the expectation of T_{uS}^L, i.e., $h_{uS}^L \triangleq \mathbb{E}[T_{uS}^L]$. By this definition, the smaller h_{uS}^L suggests that the node u is easier to reach a node in S through an L-length random walk. Similarly, the generalized hitting time can be computed recursively by the following theorem.

Theorem 2.3: For any node u and set S, h_{uS}^L can be computed by

$$h_{uS}^L = \begin{cases} 0, u \in S \\ 1 + \sum_{w \in V \setminus S} p_{uw} h_{wS}^{L-1}, u \notin S. \end{cases} \tag{4}$$

Proof: The proof is very similar to the proof of Theorem 2.2, thus we omit it for brevity. ∎

By definition, if $L = 0$, we have $h_{uS}^L = 0$ because $T_{uS}^0 = 0$. Based on the generalized hitting time, the first type of random-walk domination problem is to minimize the sum of the generalized hitting time from the nodes in $V \setminus S$ to the targeted set of nodes S subject to $|S| \leq k$. Formally, the problem is formulated as

$$\begin{aligned} &\min \sum_{u \in V \setminus S} h_{uS}^L \\ &s.t. \quad |S| \leq k. \end{aligned} \tag{5}$$

It is easy to verify that the above optimization problem is equivalent to the following one. For convenience, in the rest of this paper, we refer to the following problem as the first type of random-walk domination problem and it is denoted by Problem (1).

Problem (1):

$$\begin{aligned} &\max nL - \sum_{u \in V \setminus S} h_{uS}^L \\ &s.t. \quad |S| \leq k. \end{aligned} \tag{6}$$

Next, we formulate the second type of random-walk domination problem. Let X_{uS}^L be a random variable such that $X_{uS}^L = 1$ if u hits any one node in S by an L-length random walk, $X_{uS}^L = 0$ otherwise. Given a graph G and a constant k, the second type of random-walk domination problem is to maximize the expected number of nodes that can be dominated by S subject to a cardinality constraint, i.e., $|S| \leq k$. Formally, the problem is stated as

Problem (2):

$$\begin{aligned} &\max \mathbb{E}[\sum_{u \in V} X_{uS}^L] \\ &s.t. \quad |S| \leq k. \end{aligned} \tag{7}$$

Let p_{uS}^L be the probability of an event that an L-length random walk starting from node u successfully hits a node in S. Then, we have $\mathbb{E}[X_{uS}^L] = p_{uS}^L$. Moreover, by definition, the following theorem holds.

Theorem 2.4: For $L > 0$, we have

$$p_{uS}^L = \begin{cases} 1, u \in S \\ \sum_{w \in V} p_{uw} p_{wS}^{L-1}, u \notin S. \end{cases} \tag{8}$$

Proof: The proof can be easily obtained by definition, we therefore omit it for brevity. ■

For $L = 0$, we define $p_{uS}^0 = 1$ if $u \in S$, $p_{uS}^0 = 0$ otherwise. The rationale is that a 0-length random walk means that the walk does not move to any other nodes. Therefore, if $u \in S$, we have $p_{uS}^0 = 1$, $p_{uS}^0 = 0$ otherwise. Note that Problem (2) is different from Problem (1), because Problem (2) is to maximize the expected number of nodes that hit the targeted set by the L-length random walk, while Problem (1) is to minimize the total expected time (or the expected number of hops) of which every node hits the targeted set.

B. Discussion

Here we discuss the differences between Problem (2) and the influence maximization problem in social networks. Specifically, the influence maximization problem is to select k nodes to maximize the expected influence spread from those k nodes based on an influence spread model [17]. A commonly-used influence spread model is the independent cascade model [17]. Under the independent cascade model, the social network is modeled by a probabilistic graph, where each edge is associated with a probability and all of those probabilities are independent of one another. The influence maximization problem is to select k nodes to maximize the expected number of nodes that are reachable from the selected nodes. Recall that Problem (2) is to select k nodes to maximize the expected number of nodes that can reach a node in the targeted node set following an L-length random walk. Although these two problems are seemingly similar, Problem (2) is essentially different from the influence maximization problem. The reasons are described as follows. First, Problem (2) is based on the L-length random walk model which is a Markov-Chain model where the visiting probability of a node depends on the visiting probability of its immediate neighbors. The influence maximization problem, however, is based on the independent cascade model where the probabilities associated with the edges are independent of one another. Second, in the influence maximization problem, a targeted node could influence multiple immediate neighbors at a discrete time. However, in an L-length random walk model, each node only follows one immediate neighbor. Let us consider a concrete example to illustrate this point. For example, in Fig. 1, we assume that there is a 4-length random walk $(v_1, v_2, v_3, v_2, v_6)$ starting from v_1. Suppose that in the independent cascade model, the node v_1 has successfully influenced nodes v_2 and v_3. Clearly, in this case, v_1 has only one descendant node in the L-length random walk model, while in the independent cascade model v_1 has two. Finally, the influence maximization problem relies on the predefined influence probabilities which are the input parameters. In Problem (2), we do not require the knowledge of influence probabilities. The input parameters of our problems are the graph topology and the parameter k. Due to these differences, the techniques proposed to solve the influence maximization problem cannot be applied to our problems. In the next section, we shall develop several effective algorithms for the proposed problems.

III. The Proposed Algorithms

The goal of this section is to present algorithmic treatments for Problem (1) and Problem (2). Specifically, we first prove that both Problem (1) and Problem (2) are the instances of the submodular set function maximization with cardinality constraint problem [6]. In general, these problems are NP-hard [6]. Therefore, we strive to devise approximate algorithms for these problems. In the following, we will present several effective algorithms for Problem (1) and Problem (2) with near-optimal performance guarantee.

A. Submodularity and greedy algorithms

Before we proceed, let us give a definition of non-decreasing submodular set function [6].

Definition 3.1: Let V be a finite set, a real valued function $f(S)$ defined on the subset of V, i.e, $S \subseteq V$, is called a non-decreasing submodular set function, if the following two conditions hold. (1) For any subsets S and T of V such that $S \subseteq T \subseteq V$, we have $f(S) \leq f(T)$. (2) Let $\sigma_j(S) = f(S \cup \{j\}) - f(S)$ be the marginal gain. Then, for any subsets S and T of V such that $S \subseteq T \subseteq V$ and $j \in V \setminus T$, we have $\sigma_j(S) \geq \sigma_j(T)$.

By the above definition, we show that the objective functions of Problem (1) and Problem (2) are submodular. Specifically, let $F_1(S) = nL - \sum_{u \in V \setminus S} h_{uS}^L$, and $F_2(S) = \mathbb{E}[\sum_{u \in V} X_{uS}^L]$. Then, we have the following two theorems.

Theorem 3.2: $F_1(S)$ is non-decreasing submodular set functions with $F_1(\emptyset) = 0$.

Proof: First, it is easy to check that $F_1(\emptyset) = 0$. Second, we prove that $F_1(S)$ is a non-decreasing set function. Let $S \subseteq T \subseteq V$ be two subsets of V. Then, for any node $u \in V \backslash T$, we claim that

$$h^L_{uT} \leq h^L_{uS}. \tag{9}$$

We shall prove the above inequality by induction. By definition, we have $h^0_{uT} = h^0_{uS} = 0$ and $h^1_{uT} = h^1_{uS} = 1$. Therefore, the inequality defined in Eq. (9) holds if $L = 0$ and $L = 1$. Suppose that $h^L_{uT} \leq h^L_{uS}$ holds given $L = \alpha > 1$. Below, we show that the inequality still holds if $L = \alpha + 1$. By Eq. (4), we have

$$\begin{aligned} h^{\alpha+1}_{uS} &= 1 + \textstyle\sum_{w \notin S} p_{uw} h^{\alpha}_{wS} \\ &= 1 + \textstyle\sum_{w \notin T} p_{uw} h^{\alpha}_{wS} + \sum_{w \in T \backslash S} p_{uw} h^{\alpha}_{wS} \\ &\geq 1 + \textstyle\sum_{w \notin T} p_{uw} h^{\alpha}_{wS} \geq 1 + \sum_{w \notin T} p_{uw} h^{\alpha}_{wT} = h^{\alpha+1}_{uT}, \end{aligned}$$

where the last inequality holds due to the induction assumption. Based on Eq. (9), we have

$$\begin{aligned} F_1(S) - F_1(T) &= \textstyle\sum_{u \in V \backslash T} h^L_{uT} - \sum_{u \in V \backslash S} h^L_{uS} \\ &\leq \textstyle\sum_{u \in V \backslash T} (h^L_{uT} - h^L_{uS}) \leq 0. \end{aligned}$$

Thus, $F_1(S)$ is a non-decreasing set function as desired. Finally, we prove the submodularity property of $F_1(S)$. Let $T_u = T \cup \{u\}$ and $S_u = S \cup \{u\}$. Let $\sigma_u(S) = F_1(S_u) - F_1(S)$ be the marginal gain. Then, we have

$$\sigma_u(S) = \sum\nolimits_{w \in V \backslash S} h^L_{wS} - \sum\nolimits_{w \in V \backslash S_u} h^L_{wS_u}$$

and

$$\sigma_u(T) = \sum\nolimits_{w \in V \backslash T} h^L_{wT} - \sum\nolimits_{w \in V \backslash T_u} h^L_{wT_u}.$$

To prove the submodularity of $F_1(S)$, we show $\sigma_u(T) \leq \sigma_u(S)$ as follows:

$$\begin{aligned} &\sigma_u(S) - \sigma_u(T) \\ &= (\textstyle\sum_{w \in V \backslash S} h^L_{wS} - \sum_{w \in V \backslash T} h^L_{wT}) \\ &\quad - (\textstyle\sum_{w \in V \backslash S_u} h^L_{wS_u} - \sum_{w \in V \backslash T_u} h^L_{wT_u}) \\ &= \textstyle\sum_{w \in T \backslash S} (h^L_{wS} - h^L_{wT}) - \sum_{w \in T \backslash S} (h^L_{wS_u} - h^L_{wT_u}) \\ &= \textstyle\sum_{w \in T \backslash S} (h^L_{wS} - h^L_{wS_u}) \geq 0. \end{aligned} \tag{10}$$

Since $\sum_{w \in T \backslash S} h^L_{wT} = 0$ and $\sum_{w \in T \backslash S} h^L_{wT_u} = 0$ by Eq. (4), the third equality of the above equation holds. To prove the last inequality of Eq. (10), we can use a similar induction argument which is applied to prove Eq. (9). We omit the details for brevity. Put it all together, we conclude that $F_1(S)$ is a non-decreasing submodular set function with $F_1(\emptyset) = 0$. Therefore, the theorem is established. ■

Theorem 3.3: $F_2(S)$ is a non-decreasing submodular set function with $F_2(\emptyset) = 0$.

Proof: First, by definition, X^L_{uS} equals to zero if $S = \emptyset$, which results in $F_2(\emptyset) = 0$. Second, we show the non-decreasing property of $F_2(S)$. Let $S \subseteq T \subseteq V$ be two subsets of V. By the linearity of expectation, we have $F_2(S) = \sum_{w \in V} \mathbb{E}(X^L_{wS}) = \sum_{w \in V} p^L_{wS}$. Let p^L_{wv} be the probability that w hits v by an L-length random walk. Then, we have $p^L_{wS} = 1 - \prod_{v \in S} (1 - p^L_{wv})$. Further, we have

$$\begin{aligned} &F_2(S) - F_2(T) = \textstyle\sum_{w \in V} (p^L_{wS} - p^L_{wT}) \\ &= \textstyle\sum_{w \in V} ((1 - \prod_{v \in S} (1 - p^L_{wv})) - (1 - \prod_{v \in T} (1 - p^L_{wv}))) \\ &= \textstyle\sum_{w \in V} (\prod_{v \in T} (1 - p^L_{wv}) - \prod_{v \in S} (1 - p^L_{wv})) \leq 0. \end{aligned}$$

Therefore, $F_2(S)$ is a non-decreasing set function. Finally, we prove that $F_2(S)$ is a submodular set function. Let $u \in V \backslash T$, $S_u = S \cup \{u\}$, and $T_u = T \cup \{u\}$. Further, we let $\rho_u(S) = F_2(S \cup \{u\}) - F_2(S)$ be the marginal gain. Then, we have

$$\rho_u(S) = \sum\nolimits_{w \in V} (\prod\nolimits_{v \in S} (1 - p^L_{wv}) - \prod\nolimits_{v \in S_u} (1 - p^L_{wv}))$$

and

$$\rho_u(T) = \sum\nolimits_{w \in V} (\prod\nolimits_{v \in T} (1 - p^L_{wv}) - \prod\nolimits_{v \in T_u} (1 - p^L_{wv})).$$

In the following, we show that $\rho_u(S) \geq \rho_u(T)$. Specifically, we have

$$\begin{aligned} &\rho_u(S) - \rho_u(T) \\ &= \textstyle\sum_{w \in V} ((\prod_{v \in S} (1 - p^L_{wv}) - \prod_{v \in T} (1 - p^L_{wv})) \\ &\qquad - (\prod_{v \in S_u} (1 - p^L_{wv}) - \prod_{v \in T_u} (1 - p^L_{wv}))) \\ &= \textstyle\sum_{w \in V} ((1 - \prod_{v \in T \backslash S} (1 - p^L_{wv})) \prod_{v \in S} (1 - p^L_{wv}) \\ &\qquad - (1 - \prod_{v \in T \backslash S} (1 - p^L_{wv})) \prod_{v \in S_u} (1 - p^L_{wv}) \\ &= \textstyle\sum_{w \in V} ((1 - \prod_{v \in T \backslash S} (1 - p^L_{wv})) p^L_{wu} \prod_{v \in S} (1 - p^L_{wv}) \\ &\geq 0. \end{aligned}$$

This completes the proof. ■

Algorithm 1 The greedy algorithm

Input: A graph $G = (V, E)$, and a parameter k
Output: A set of nodes S

1: $S \leftarrow \emptyset$;
2: **for** $i = 1$ to k **do**
3: $\quad v \leftarrow \arg\max_{u \in V \backslash S} \{F(S \cup \{u\}) - F(S)\}$;
4: $\quad S \leftarrow S \cup \{v\}$;
5: **return** S;

Based on the submodularity of F_1 and F_2, we present a greedy algorithm for both Problem (1) and Problem (2) depicted in Algorithm 1. The greedy algorithm works in k rounds (line 2-4). In each round, the algorithm selects a node with the maximum marginal gain (line 3), and adds it into the answer set S (line 4) which is initialized to be an empty set (line 1). Note that to solve Problem (1) and Problem (2), we need to replace the function F in Algorithm 1 with F_1 and F_2 respectively. By a celebrated result in [6], Algorithm 1 achieves a $(1-1/e)$ approximation factor for Problem (1) and Problem (2), where $e \approx 2.718$ denotes the Euler's number.

Complexity analysis: The time complexity of Algorithm 1 is dominated by the time complexity for computing the marginal gain (line 3). Below, we focus on the analysis of the greedy algorithm for Problem (1), and similar analysis can be done for Problem (2). For F_1, let $\sigma_u(S) = F_1(S \cup \{u\}) - F_1(S)$ be the marginal gain. Then, $\sigma_u(S)$ can be calculated based on Eq. (4). Note that Eq. (4) immediately implies a dynamic programming (DP) algorithm for computing h^L_{uS}. Given a set S, the time

complexity for computing h^L_{uS} is $O(mL)$. Therefore, the time complexity for calculating $F_1(S) = \sum_{u \in V \setminus S}(L - h^L_{uS})$ is $O(nmL)$. Since the greedy algorithm needs to find the node with the maximum marginal gain, it has to evaluate $F_1(S \cup \{u\})$ for every node u in $V \setminus S$. As a result, the time complexity of the greedy algorithm is $O(kn^2mL)$. We can use the so-called lazy evaluation strategy [18] to speed up the greedy algorithm, which could result in several orders of magnitude speedup as observed in [18]. For the space complexity, the DP algorithm has to maintain an $n \times L$ array for a given S. To compute the marginal gain, the greedy algorithm needs to evaluate $F_1(S \cup \{u\})$ for every node u in $V \setminus S$, thus the space complexity of the greedy algorithm is $O(n^2L)$. Similarly, for Problem (2), the time and space complexity of the greedy algorithm are $O(kn^2mL)$ and $O(n^2L)$ respectively.

Approximate marginal gain computation: According to the above complexity analysis, the DP-based greedy algorithm is clearly impractical. The most time and space consuming step in the greedy algorithm is to compute the objective functions and the corresponding marginal gains. Here we present a sampling-based algorithm to estimate such quantities efficiently.

Given a set S, to estimate the objective function $F_1(S)$ ($F_2(S)$), the key step is to estimate h^L_{uS} ($\mathbb{E}[X^L_{uS}]$). Below, we present an unbiased estimator for estimating h^L_{uS}. In particular, to construct an unbiased estimator for h^L_{uS}, we independently run R L-length random walks starting from node u. Assume that there are r such random walks that have hit any one node in S for the first time at $t_{i_1}, \cdots, t_{i_r}$ hops respectively. Then, we give an estimator for h^L_{uS} by

$$\hat{h}^L_{uS} = \sum\nolimits_{k=1}^{r} t_{i_k}/R + (1 - r/R)L. \tag{11}$$

The following lemma shows that $\hat{h}^L_{uS}$ is unbiased.

Lemma 3.4: $\hat{h}^L_{uS}$ is an unbiased estimator of h^L_{uS}.

Proof: Recall that $h^L_{uS} = \mathbb{E}[T^L_{uS}]$. By Eq. (3), T^L_{uS} denotes the first time that an L-length random walk starting from u hits any arbitrary node in S. If such a random walk cannot hit the nodes in S, then $T^L_{uS} = L$. To estimate the expectation of T^L_{uS}, we independently run R L-length random walks starting from u, and take the average hitting time as the estimator. The proposed sampling process is equivalent to a simple random sampling with replacement, thus the estimator is unbiased. ■

Based on Lemma 3.4, $\hat{F}_1(S) = \sum_{u \in V \setminus S}(L - \hat{h}^L_{uS})$ is also an unbiased estimator of $F_1(S)$. Similarly, we can construct an estimator for $\mathbb{E}[X^L_{uS}]$ which is given by

$$\hat{\mathbb{E}}[X^L_{uS}] = r/R. \tag{12}$$

Also, we can show that the estimator $\hat{\mathbb{E}}[X^L_{uS}]$ is unbiased.

Lemma 3.5: $\hat{\mathbb{E}}[X^L_{uS}]$ is an unbiased estimator of $\mathbb{E}[X^L_{uS}]$.

Proof: The proof can be easily obtained by definition, thus we omit it for brevity. ■

Likewise, by Lemma 3.5, $\hat{F}_2(S) = \sum_{u \in V} \hat{\mathbb{E}}[X^L_{uS}]$ is an unbiased estimator of $F_2(S)$. We remark that in [26], Sarkar et al. presented a similar unbiased estimator for estimating the hitting time of the L-length random walk between two nodes. Our estimator ($\hat{h}^L_{uS}$) is to estimate the generalized hitting time of the L-length random walk between a source node and a targeted set. In this sense, the proposed estimator is more general than the estimator presented in [26]. Next, we apply Hoeffding's inequality [27] to bound the sample size R. Specifically, we have the following two lemmas.

Lemma 3.6: Given a set S, for two small constants ϵ and δ, if $R \geq \frac{1}{2\varepsilon^2} \log \frac{n-|S|}{\delta}$, then $\Pr[|\hat{F}_1(S) - F_1(S)| \geq \epsilon(n - |S|)L] \leq \delta$.

Proof: First, we have

$$\begin{aligned}\Pr[|\hat{F}_1(S) - F_1(S)| &\geq \varepsilon(n - |S|)L] \\ &\leq \Pr[\textstyle\sum_{u \in V/S} |\hat{h}_{uS} - h_{uS}| \geq \varepsilon(n - |S|)L],\end{aligned}$$

as $|\hat{F}_1(S) - F_1(S)| \geq \varepsilon(n - |S|)L$ implies $\sum_{u \in V/S} |\hat{h}_{uS} - h_{uS}| \geq \varepsilon(n - |S|)L$. Then, by the union bound, we have

$$\begin{aligned}\Pr[\sum_{u \in V/S} &|\hat{h}_{uS} - h_{uS}| \geq \varepsilon(n - |S|)L] \\ &\leq \textstyle\sum_{u \in V/S} \Pr[|\hat{h}_{uS} - h_{uS}| \geq \varepsilon L].\end{aligned}$$

Since $0 \leq \hat{h}_{uS} \leq L$ (Lemma 2.1), we can apply Hoeffding's inequality [27] to bound the sample size R. Specifically, we have

$$\Pr[|\hat{h}_{uS} - h_{uS}| \geq \varepsilon L] \leq \exp(-2\varepsilon^2 R).$$

Based on this, the following inequality immediately holds

$$\Pr[|\hat{F}_1(S) - F_1(S)| \geq \varepsilon(n - |S|)L] \leq (n - |S|)\exp(-2\varepsilon^2 R).$$

Let $(n - |S|)\exp(-2\varepsilon^2 R) \leq \delta$, then we can get $R \geq \frac{1}{2\varepsilon^2} \log \frac{n-|S|}{\delta}$, which completes the proof. ■

Lemma 3.7: Given a set S, for two small constants ϵ and δ, if $R \geq \frac{1}{2\varepsilon^2} \log \frac{n}{\delta}$, then $\Pr[|\hat{F}_2(S) - F_2(S)| \geq \epsilon n] \leq \delta$.

Proof: The proof is similar to the proof of Lemma 3.6, thus we omit for brevity. ■

Based on the above analysis, we present a sampling-based greedy algorithm where the marginal gain $\sigma_u(S) = F_1(S \cup \{u\}) - F_1(S)$ (or $\rho_u(S) = F_2(S \cup \{u\}) - F_2(S)$) can be estimated using the estimators $\hat{F}_1(S)$ and $\hat{F}_1(S \cup \{u\})$ ($\hat{F}_2(S)$ and $\hat{F}_2(S \cup \{u\})$). Note that each estimator ($\hat{F}_1(S)$ or $\hat{F}_2(S)$) has to run nR independent L-length random walks. Due to space limit, the detailed description of this algorithm is omitted, and it can be found in our technical report [28]. One can easily derive that the time complexity of the sampling-based greedy algorithm is $O(kn^2RL)$. The reason is described as follows. First, running an L-length random walk takes $O(L)$ time complexity. Therefore, each marginal gain evaluation requires $O(nRL)$ time complexity. Second, the algorithm has to select the node with maximal marginal gain, thus the time complexity taken in each round is $O(n^2RL)$. There are k rounds in total, thus the time complexity of this algorithm is $O(kn^2RL)$. The space complexity of this algorithm is $O(m + n)$, which is significantly better than the DP-based greedy algorithm.

By Lemma 3.6 and Lemma 3.7, the sampling algorithm can get a good approximation of the marginal gain. As a consequence, the approximation guarantee of the sampling-based greedy algorithm can be preserved. Indeed, by a similar

analysis presented in [17], the sampling-based greedy algorithm achieves a $1 - 1/e - \epsilon$ approximation factor through setting an appropriate parameter R.

B. Approximate greedy algorithm

Recall that the sampling-based greedy algorithm has to run $O(kn^2R)$ L-length random walks. Can we reduce the *sample complexity* of the sampling-based greedy algorithm? In this subsection, we give an algorithm that only runs $O(nR)$ L-length random walks in total, and it also preserves the $1 - 1/e - \epsilon$ approximation factor. For convenience, we call this algorithm the approximate greedy algorithm. Below, we mainly focus on describing the algorithm for Problem (1), and similar descriptions can be used for Problem (2) (we have added some remarks for Problem (2) in Algorithms 2, 3, 4, and 5).

The key idea is described as follows. First, for each node, the algorithm independently runs R L-length random walks. Then, the algorithm materializes such *samples* (An L-length random walk is a sample, there are nR samples in total), and applies them to estimate the marginal gain $\sigma_u(S)$ for any given node u and a given set S. Here the challenge is how to estimate $\sigma_u(S)$ efficiently using such samples, because S changes in each round of the greedy algorithm. To overcome this challenge, we present an inverted list structure to index the samples. Specifically, we build R inverted lists, and each inverted list includes n sublists. For each node u, a sublist indexes all the other nodes that hit u through an L-length random walk. Here the entry of the sublist is an object that includes two parts: a node ID (id) and a weight ($weight$), denoting a node id hits u at $weight$-th step of an L-length random walk. Algorithm 2 depicts the inverted index construction algorithm. In Algorithm 2, the R inverted lists, denoted by $I[1:R][1:n]$, are organized as a two-dimensional list array, in which $I[i][v]$ indexes all the nodes that hit v by the i-th L-length random walk. First, the algorithm initializes $I[1:R][1:n]$ by an empty array (line 1). Then, for each node w in V, the algorithm runs R L-length random walks (line 2-14). Let us consider the i-th L-length random walk starting at node w. If w hits a node v, the algorithm creates an object $< w, weight >$, where $weight$ denotes that w hits v at $weight$-step (line 11-12). Then, the algorithm adds it into $I[i][v]$ (line 13). Note that for the repeated nodes in an L-length random walk, we only need to index one node and record the $weight$ at the first visiting time according to the definition of hitting time. To remove such repeated nodes in an L-length random walk, the algorithm maintains a $visited[1:n]$ array (line 4, 6 and 9-10).

Given the inverted lists $I[1:R][1:n]$, how to estimate the marginal gain for any node u and a given set S? Here we tackle this issue by maintaining a two-dimensional array $D[1:R][1:n]$. Given a set S, let $D[i][u]$ be an estimator of the hitting time h^L_{uS} (or $\mathbb{E}[X^L_{uS}]$ for Problem (2)) based on the i-th L-length random walk. Let $S_u = S \cup \{u\}$, and $\sigma_u(S) = F_1(S_u) - F_1(S)$ be the marginal gain. Then, we can derive that $\sigma_u(S) = \sum_{w \in V \setminus S_u} (h^L_{wS} - h^L_{wS_u}) + h^L_{uS} - L$. Recall that in each round of the greedy algorithm, we need to find the node with the maximum marginal gain. Therefore, we can estimate the marginal gain σ_u by $\sum_{w \in V \setminus S_u} (h^L_{wS} - h^L_{wS_u}) + h^L_{uS}$ for each node u, because "$-L$" does not affect the results. Algorithm 3 describes an algorithm for estimating σ_u. Let us consider the i-th L-length random walk. First, σ_u is initialized by 0. Then, the algorithm adds $D[i][u]$, which is an estimator of h^L_{uS}, to σ_u (line 3). And then, the algorithm estimates $\sum_{w \in V \setminus S_u} (h^L_{wS} - h^L_{wS_u})$ and adds it to σ_u, which is implemented in line 4-7. By definition, if a node v in $V \setminus S_u$ does not hit u, then we have $h^L_{vS} = h^L_{vS_u}$. Thus, the algorithm only needs to consider the nodes that have hit u (line 4) which is indexed in $I[i][u]$. If $h^L_{vu} < h^L_{vS}$, then the algorithm adds $h^L_{vS} - h^L_{vS_u} = h^L_{vS} - h^L_{vu}$ to σ_u. Otherwise, we have $h^L_{vS} = h^L_{vS_u}$. Note that by definition, h^L_{vu} can be estimated by the $weight$ associated with v which is indexed in $I[i][u]$, and h^L_{vS} can be estimated by $D[i][v]$, and thus $h^L_{vS} - h^L_{vu}$ can be estimated by $D[i][v]$ minus the $weight$ associated with v (line 7). Therefore, line 3-7 of Algorithm 3 are used to estimate σ_u based on the i-th L-length random walk. Finally, Algorithm 3 takes an average over all the R estimations (line 10).

Algorithm 3 can be used to estimate the marginal gain for every node given a set S. In the greedy algorithm, after one round, the size of S increases by 1. Hence, we need to dynamically maintain the array $D[1:R][1:n]$ when S is changed. Algorithm 4 depicts an algorithm to update $D[1:R][1:n]$ given that S includes at least one node. As usual, let us consider the i-th L-length random walk. By definition, for a node v, if $h^L_{vu} < h^L_{vS}$, then we need to update $D[i][v]$. Otherwise, we have $h^L_{vS} = h^L_{vS_u}$, thus no need to update $D[i][v]$. In addition, for a node v that does not hit u, we do not need to update $D[i][v]$ as $h^L_{vS} = h^L_{vS_u}$ by definition. In Algorithm 4, the algorithm first sets $D[i][u]$ to 0 (line 2), because $h^L_{uS_u} = 0$ (u is in S_u). Then, the algorithm updates $D[i][v]$ for the node v that has hit u by the i-th L-length random walk (line 3-6).

Equipped with Algorithm 2, Algorithm 3, and Algorithm 4,

Algorithm 2 Invert_Index(G, L, R)

Input: A graph $G = (V, E)$, two parameters L and R
Output: An inverted index $I[1:R][1:n]$

```
1: Initialize an inverted list I[1:R][1:n] ← null;
2: for each node w ∈ V do
3:    for i = 1 : R do
4:       Initialize visited[1:n] ← 0;
5:       u ← w;
6:       visited[u] ← 1;
7:       for j = 1 : L do
8:          Randomly select a neighbor of u, denoted by v;
9:          if visited[v] = 0 then
10:            visited[v] ← 1;
11:            Object.id ← w;
12:            Object.weight ← j; /*w hits v at j-th step*/
               /*Object.weight ← 1; for Problem (2)*/;
13:            I[i][v].push_back(Object);
14:         u ← v;
15: return I[1:R][1:n];
```

Algorithm 3 Approx_Gain($I[1:R][1:n]$, $D[1:R][1:n]$, u, R)

Input: The inverted index $I[1:R][1:n]$, the array $D[1:R][1:n]$, a node u and parameter R
Output: Approximate marginal gain σ_u

1: Initialize $\sigma_u \leftarrow 0$;
2: **for** $i = 1 : R$ **do**
3: $\sigma_u \leftarrow \sigma_u + D[i][u]$;
/*$\sigma_u \leftarrow \sigma_u + 1 - D[i][u]$; for Problem (2)*/
4: **while** $Object \leftarrow I[i][u].pop()$ **do**
5: $v \leftarrow Object.id$;
6: **if** $Object.weight < D[i][v]$ **then**
7: $\sigma_u \leftarrow \sigma_u + D[i][v] - Object.weight$;
/*for Problem (2), use line 8-9 to replace line 6-7*/
8: **if** $Object.weight > D[i][v]$ **then**
9: $\sigma_u \leftarrow \sigma_u + Object.weight - D[i][v]$;
10: $\sigma_u \leftarrow \sigma_u / R$;
11: **return** σ_u;

Algorithm 4 Update($I[1:R][1:n]$, $D[1:R][1:n]$, u, R)

Input: The inverted index $I[1:R][1:n]$, the array $D[1:R][1:n]$, a node u and parameter R
Output: The updated array $D[1:R][1:n]$

1: **for** $i = 1 : R$ **do**
2: $D[i][u] \leftarrow 0$; /*$D[i][u] \leftarrow 1$; for Problem (2)*/
3: **while** $Object \leftarrow I[i][u].pop()$ **do**
4: $v \leftarrow Object.id$;
5: **if** $Object.weight < D[i][v]$ **then**
6: $D[i][v] \leftarrow Object.weight$;
/*for Problem (2), use line 7-8 to replace line 5-6)*/
7: **if** $Object.weight > D[i][v]$ **then**
8: $D[i][v] \leftarrow Object.weight$;

we present the approximate greedy algorithm in Algorithm 5. First, Algorithm 5 builds R inverted lists (line 1). Second, the algorithm initializes the answer set S to an empty set (line 2), and sets the value of each entry in $D[1:R][1:n]$ to L (line 3), because $h^L_{uS} = L$ given that $S = \emptyset$. Third, the algorithm works in k rounds (line 4-7). In each round, the algorithm invokes Algorithm 3 to estimate the marginal gain $\sigma_u(S)$, and selects the node v with the maximum $\sigma_u(S)$. Then, the algorithm adds v into the answer set S. After that, the algorithm invokes Algorithm 4 to update $D[1:R][1:n]$. The following example illustrates how the Algorithm 5 works.

Example 3.8: Let us re-consider the example graph shown in Fig. 1. For simplicity, we set $R = 1$, $L = 2$, and $k = 2$. Suppose that the 2-length random walks for each node generated by the algorithm are described as follows: (v_1, v_2, v_3), (v_2, v_3, v_5), (v_3, v_2, v_5), (v_4, v_7, v_5), (v_5, v_2, v_6), (v_6, v_7, v_5), (v_7, v_5, v_7), and (v_8, v_7, v_4). Then, the inverted index constructed by Algorithm 2 ($I[1][1:8]$) is illustrated Table I. Note that in (v_7, v_5, v_7), v_7 is a repeated node, thus the second v_7 will not be inserted into the inverted list by Algorithm 2. After building the inverted index, Algorithm 5 initializes S to an empty set, and set all the elements of $D[1][1:8]$ to 2. Then, in the first round, the algorithm invokes Algorithm 3 to estimate the marginal gain $\sigma_u(\emptyset)$ for

Algorithm 5 The approximate greedy algorithm

Input: A graph $G = (V, E)$, and parameters k, R
Output: A set of nodes S

1: $I[1:R][1:n] \leftarrow$Invert_Index(G, L, R);
2: $S \leftarrow \emptyset$;
3: Initialize $D[1:R][1:n] \leftarrow L$;
/*$D[1:R][1:n] \leftarrow 0$; for Problem (2)*/
4: **for** $i = 1$ to k **do**
5: $v \leftarrow \arg\max_{u \in V \setminus S}$ Approx_Gain($I[1:R][1:n], D[1:R][1:n], u, R$);
6: $S \leftarrow S \cup \{v\}$;
7: Update($I[1:R][1:n]$, $D[1:R][1:n]$, v, R);
8: **return** S;

TABLE I
INVERTED INDEX

v_1:	
v_2:	$< v_1, 1 >, < v_3, 1 >, < v_5, 1 >$
v_3:	$< v_1, 2 >, < v_2, 1 >$
v_4:	$< v_8, 2 >$
v_5:	$< v_2, 2 >, < v_3, 2 >, < v_4, 2 >, < v_6, 2 >, < v_7, 1 >$
v_6:	$< v_5, 2 >$
v_7:	$< v_4, 1 >, < v_6, 1 >, < v_8, 1 >$
v_8:	

each node. After this step, we can get that $\sigma_{v_1}(\emptyset) = 2$, $\sigma_{v_2}(\emptyset) = 5$, $\sigma_{v_3}(\emptyset) = 3$, $\sigma_{v_4}(\emptyset) = 2$, $\sigma_{v_5}(\emptyset) = 3$, $\sigma_{v_6}(\emptyset) = 2$, $\sigma_{v_7}(\emptyset) = 5$, and $\sigma_{v_8}(\emptyset) = 2$. For instance, for node v_2, there are three elements in the inverted list $I[1][2]$. Since the weights of v_1, v_3, and v_5 (all of them equal to 1) are smaller than $D[1][1]$, $D[1][3]$, and $D[1][5]$ (all of them equal to 2) respectively, thus $\sigma_{v_2}(\emptyset) = D[1][2] + 3 = 5$ as desired. Similar analysis can be used for other nodes. Clearly, v_2 and v_7 achieve the maximum marginal gain. The algorithm breaks ties randomly. Assume that in this round, the algorithm selects v_2 and adds it into S. Then, the algorithm invokes Algorithm 4 (Update($I[1][1:8], D[1][1:8], v_2, 1$)) to update $D[1][1:8]$. After this step, only $D[1][2]$, $D[1][1]$, $D[1][3]$, and $D[1][5]$ need to be updated, and they are re-set to 0, 1, 1, and 1 respectively. Similar arguments can be used for analyzing the second round. Here we only report the result, and omit the details for brevity. In the second round, the algorithm adds v_7 into the answer set. Therefore, the algorithm outputs $\{v_2, v_7\}$ as the targeted nodes.

Complexity analysis: We analyze the time and space complexity of Algorithm 5 as follows. First, to build the inverted index (line 1), Algorithm 2 takes $O(RLn)$ time complexity. Second, to estimate the marginal gain for every node, the algorithm needs to invoke Algorithm 3 $O(n)$ times. We can derive that the time complexity of this step (line 5) is $O(nRL)$, because the algorithm only needs to access the entire inverted index once and the size of the inverted index is bounded by $O(nRL)$. Third, to update $D[1:R][1:n]$, Algorithm 4 takes at most $O(Rn)$ time. Put it all together, the time complexity of Algorithm 5 is $O(kRLn)$, which is linear w.r.t. the graph size (R, k, and L are constants). For the space complexity, the algorithm needs to maintain two arrays: the inverted index $I[1:R][1:n]$ and the array $D[1:R][1:n]$. Clearly, $I[1:R][1:n]$ and $D[1:R][1:n]$ are bounded by $O(RLn)$

and $O(Rn)$ respectively. Therefore, the space complexity of Algorithm 5 is $O(nRL + m)$.

Note that in Algorithm 5, each marginal gain is estimated by the same R L-length random walks. Since the L-length random walks are independent of one another, the estimator is able to achieve high accuracy by setting an appropriate R. As a result, the approximation factor of Algorithm 5 is very close to $1 - 1/e - \epsilon$. Indeed, in the experiments, we find that the effectiveness of Algorithm 5 is comparable with the DP-based greedy algorithm even when R is a small value (e.g., $R = 100$).

IV. Experiments

In this section, we conduct extensive experiments over both synthetic and real-world graphs to evaluate the proposed algorithm. Below, we first describe the experimental setup and then report our results.

Different algorithms: To the best of our knowledge, we are the first group to study the random-walk domination problems. In the literature, no algorithm has been proposed to solve these problems. Therefore, to evaluate the proposed algorithms, we rely on comparing them with the following two baseline algorithms. The first baseline algorithm is the degree-based algorithm. Intuitively, the high-degree nodes are more easily reached by the other nodes. Hence, to maximize the expected number of reached nodes, a reasonable baseline algorithm is to select the top-k high-degree nodes as the targeted nodes. For convenience, we refer to this baseline algorithm as the *Degree* algorithm. The second baseline is the traditional dominating-set-based algorithm [7]. A dominating set is a subset of nodes $D \subset V$ such that every node in V is either in D or a neighbor of some nodes in D [7]. By this definition, every node can only dominate its neighbors. In our problems, since we have a cardinality constraint, i.e., $|S| \leq k$, we cannot select the entire dominating set. Instead, we turn to select k nodes such that they can dominate as many nodes as possible. Note that here the concept of domination is based on the definition of traditional dominating set. Specifically, let S be the set of targeted nodes. Initially, S is an empty set. The algorithm works in k rounds. In each round, the algorithm selects a node v such that $v = \arg\max_{u \in V/S} |N(\{u\}) - N(S)|$, where $N(S)$ denotes the set of immediate neighbors of nodes in S. Then, the algorithm adds v into the set S, and goes to the next round. We call this algorithm the *Dominate* algorithm.

The first proposed algorithm is the DP-based greedy algorithm, in which the marginal gain is calculated by the DP algorithm. The second proposed algorithm is the approximate greedy algorithm, i.e., Algorithm 5. Both of them are used to solve Problem (1) (Eq. (6)) and Problem (2) (Eq. (7)). Here we do not report the results of the sampling-based greedy algorithm because the approximate greedy algorithm is more efficient than such an algorithm. For convenience, we refer to the first algorithm for solving Problem (1) and Problem (2) as *DPF1* and *DPF2* respectively. Similarly, we call the second algorithm for solving Problem (1) and Problem (2) as *ApproxF1* and *ApproxF2* respectively.

TABLE II
Summary of the datasets

Name	# of nodes	# of edges
CAGrQc	5,242	28,968
CAHepPh	12,008	236,978
Brightkite	58,228	428,156
Epinions	75,872	396,026

Evaluation metrics: Two metrics are used to evaluate the effectiveness of different algorithms. The first metric is the average hitting time which is defined as $M_1(S) = \sum_{u \in V \setminus S} h^L_{uS}/|V \setminus S|$, where S denotes the set of selected nodes by an algorithm. This metric inversely measures the effectiveness of the algorithm. In other words, the smaller the $M_1(S)$ is, the more effective the algorithm is. The second metric is the expected number of nodes that hit a node in S via an L-length random walk. The formula of the second metric is given by $M_2(S) = \sum_{u \in V} \mathbb{E}[X^L_{uS}]$. The larger $M_2(S)$ is, the more effective the algorithm is. For convenience, we refer to the first metric and the second metric as *AHT* and *EHN* respectively. Note that to compute these metrics, we use the sampling algorithm described in Section III-A and set the sample size $R = 500$. To evaluate the efficiency of different algorithms, we record the running time, which is measured by the wall-clock time.

Datasets and experimental environment: We use four real-world datasets in the experiments: CAGrQc, CAHepPh, Brightkite, and Epinions. The CAGrQc and CAHepPh datasets are co-authorship networks, the Brightkite is a location-based social network dataset, and the Epinions is a trust social network dataset. We download these datasets from Stanford network data collections [29]. The detailed statistic information of the datasets is shown in Table II. We conduct all the experiments on a Windows XP PC with 2xQuad-Core Intel Xeon 2.66 GHz CPU, and 4GB memory. All the algorithms are implemented in C++.

A. Experimental Results

Performance of the approximate greedy algorithms: Here we compare the effectiveness of the approximate greedy algorithms (*ApproxF1* and *ApproxF2*) with those of the DP-based greedy algorithm (*DPF1* and *DPF2*). Due to the expensive time and space complexity of the *DPF1* and *DPF2* algorithms, these two algorithms can only work well on very small datasets. To this end, we generate a small synthetic graph with 1000 nodes and 9956 edges based on a commonly-used power-law random graph model [30]. We set the parameter k to 30 which denotes the number of selected nodes, and set the parameter L in the L-length random walk model to 5 and 10 respectively. Similar results can be observed for other values of k and L. The results are shown in Fig. 2 and Fig. 3. Specifically, Fig. 2 depicts the comparison of effectiveness of *DPF1* and *ApproxF1* algorithms. The black dash line in Fig. 2 describes the effectiveness of the *DPF1* algorithm, while the red solid curve depicts the effectiveness of the *ApproxF1* algorithm as a function of the parameter R. As can be seen

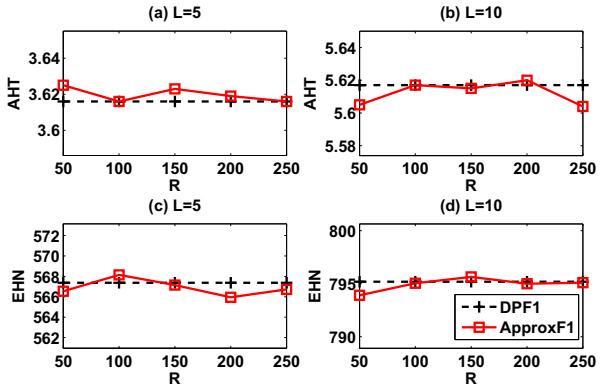

Fig. 2. Comparison of effectiveness of *DPF1* and *ApproxF1*

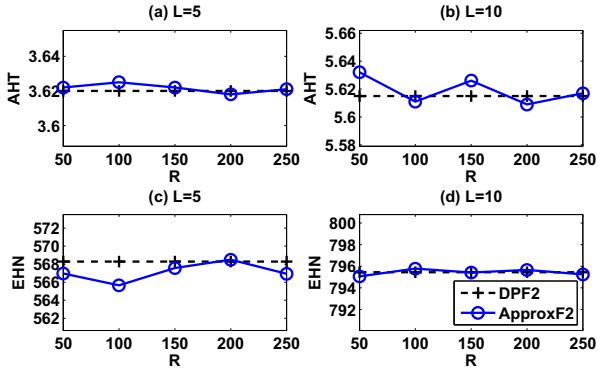

Fig. 3. Comparison of effectiveness of *DPF2* and *ApproxF2*

in Fig. 2, the *ApproxF1* algorithm is very accurate when the number of samples is greater than or equal to 50. For example, in Fig. 2(a), the greatest difference of *AHT* between *DPF1* and *ApproxF1* algorithms is around 0.01, which is achieved at $R = 50$. Moreover, when $R = 100$, the result of the *ApproxF1* algorithm matches the result of the *DPF1* algorithm. In Fig. 2(c), we can see that the expected number nodes that hit the selected nodes calculated by the *ApproxF1* algorithm is very close to the expected number of nodes computed by the *DPF1* algorithm. The maximal difference of *EHN* between *DPF1* and *ApproxF1* algorithms is around 1.5, which is achieved at $R = 200$.

Fig. 3 illustrates the comparison of effectiveness of *DPF2* and *ApproxF2* algorithms. Similarly, from Fig. 3, we can observe that the effectiveness of the *ApproxF2* algorithm is very close to that of the *DPF2* algorithm. In Fig. 3(a), for instance, the maximal difference of *AHT* between the *DPF2* and *ApproxF2* algorithms is smaller than 0.01 (obtained at $R = 100$). Hence, for both *AHT* and *EHN* metrics, the approximate greedy algorithms work very well with a small R value. These results are consistent with the analysis in Section III-B.

Effectiveness of different algorithms: As indicated in the previous experiment, under both *AHT* and *EHN* metrics, there is no significant difference between the *ApproxF1* (*ApproxF2*) algorithm and the *DPF1* (*DPF2*) algorithm. Furthermore, the former algorithms have much lower time and space complexity than the latter algorithms. Consequently, in the following experiments, we only report the results obtained by the *ApproxF1*

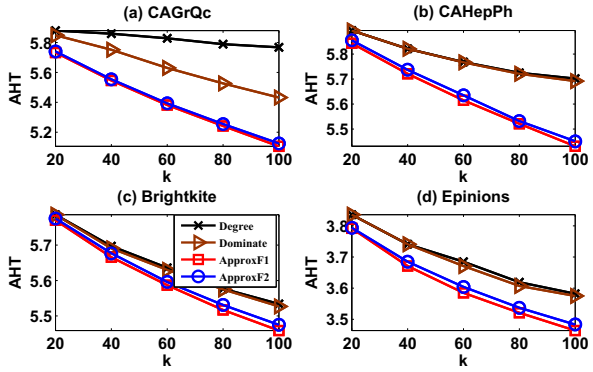

Fig. 4. Comparison of *AHT* of different algorithms

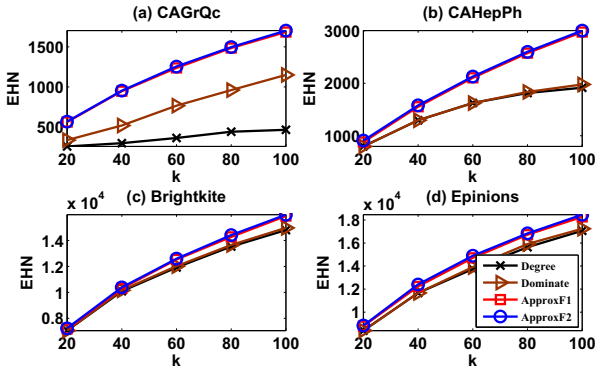

Fig. 5. Comparison of *EHN* of different algorithms

and *ApproxF2* algorithms. For these algorithms, we set the parameter R to 100 in all the following experiments without any specific statements, because $R = 100$ is sufficient to ensure good accuracy as indicated in the previous experiment. For all the algorithms, we set the parameter L to 6. Similar results can be observed for other L values. Fig. 4 and Fig. 5 describe the results of different algorithms over four real-world datasets under *AHT* and *EHN* metrics respectively. From Fig. 4, we can see that both the *ApproxF1* and *ApproxF2* algorithms are substantially better than the two baselines in all the datasets used. As desired, for all the algorithms, the *AHT* decreases as k increases. In addition, we can see that the *ApproxF1* algorithm slightly outperforms the *ApproxF2* algorithm, because the *ApproxF1* algorithm directly optimizes the *AHT* metric. Also, we can observe that the *Dominate* algorithm is slightly better than the *Degree* algorithm in CAHepPh, Brightkite, and Epinions datasets. In CAGrQc datasets, however, the *Degree* algorithm performs poorly, and the *Dominate* algorithm significantly outperforms the *Degree* algorithm. Similarly, as can be seen in Fig. 5, the *ApproxF1* and *ApproxF2* algorithms substantially outperform the baselines over all the datasets under the *EHN* metric. Moreover, we can see that the *ApproxF2* algorithm is slightly better than the *ApproxF1* algorithm, because the *ApproxF2* algorithm directly maximizes the *EHN* metric. Note that, under both *AHT* and *EHN* metrics, the gap between the curves of the approximate greedy algorithms and those of the two baselines increases with increasing k. The rationale is that the approximate greedy algorithms are near-optimal which achieve $1 - 1/e - \epsilon$ approximation factor, and such approximation factor is independent of the parameter k. The two

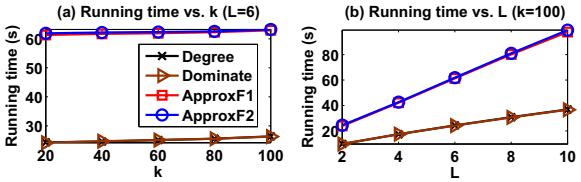

Fig. 6. Running time of different algorithms (Epinions)

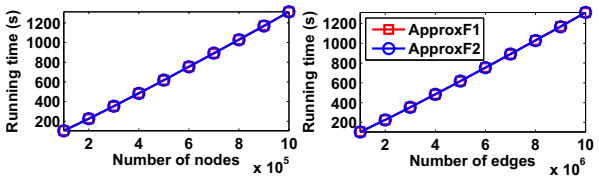

Fig. 7. Scalability testing

baselines, however, are without any performance guarantee, thus the effectiveness of these two algorithms would decrease as k increases. These results are consistent with our theoretical analysis in Section III.

Efficiency of different algorithms: In this experiment, we evaluate the efficiency of different algorithms. Fig. 6 shows the comparison of the running time of different algorithms over the Epinions dataset. Similar results can be obtained in other datasets. In particular, Fig. 6(a) depicts the running time of different algorithms as a function of the parameter k. Here the parameter L is set to 6. We are able to observe that the running time of the *ApproxF1* and *ApproxF2* algorithms are around 2.5 times longer than the running time of the *Degree* and *Dominate* algorithms. Fig. 6(b) illustrates the running time of different algorithms as a function of the parameter L, where we set the parameter k to 100. As can be observed in Fig. 6(b), the running time of the *ApproxF1* and *ApproxF2* algorithms are longer than that of the *Degree* and *Dominate* algorithms by 2.7 times at most. For example, when $L = 10$, the running time of the *ApproxF1* is 99 seconds, while the running time of the *Degree* algorithm is 37 seconds. These results indicate that the running time of the approximate greedy algorithms is only a small constant times longer than that of the *Degree* algorithm, which are consistent with the complexity analysis in Section III-B.

Scalability testing: Here we evaluate the scalability of the *ApproxF1* and *ApproxF2* algorithms. To this end, we generate ten large synthetic graphs according to a widely-used power-law random graph model [30]. More specifically, we generate ten graphs $G_1, \cdots, G_{10}$ such that G_i has $i \times 0.1$ million nodes and i million edges for $i = 1, \cdots, 10$. Fig. 7 shows the results of the *ApproxF1* and *ApproxF2* algorithms w.r.t. the number of nodes (left panel) and w.r.t. the number of edges (right panel). Here we set the parameter $L = 6$ and $k = 100$. Similar results can be observed for other values of L and k. From Fig. 7, we find that both the *ApproxF1* and *ApproxF2* algorithms scale linearly w.r.t. both the number of nodes and the number of edges, which is consistent with the linear time complexity (w.r.t. the graph size) of the algorithm.

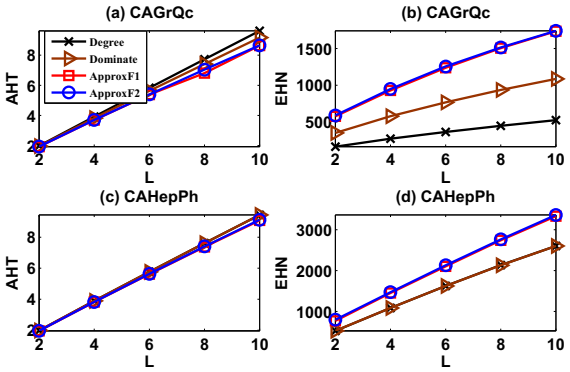

Fig. 8. Effect of parameter L

Effect of parameter L: Here we study the effect of parameter L. Fig. 8 reports the results in CAGrQc and CAHepPh datasets given $k = 60$. Similar results can be observed in other datasets and other values of k as well. From Fig. 8(a-d), we can see that both the *AHT* and *EHN* by different algorithms increase as L increases. Recall that the hitting time is bounded by L, and the hitting time of a node that cannot hit the targeted nodes is set to L. Therefore, the average hitting time increases if L increases. Clearly, with L increasing, the number of nodes that can hit the targeted nodes increases, thereby the *EHN* of different algorithms increase. In addition, we find that the gap between the curves of the *ApproxF1* and *ApproxF2* algorithms and the curves of the baselines increases as L increases, which suggests that the *ApproxF1* and *ApproxF2* algorithms perform very well for large L values.

V. Conclusions

In this paper, we introduce and formulate two types of random-walk domination problems in graphs motivated by a number of applications such as item-placement in social networks, resource-placement in P2P networks, and advertisement-placement in advertisement networks. We show that these problems are the instances of submodular set function maximization with cardinality constraint problem. Based on this, we propose a dynamic programming (DP) based greedy algorithm with $1 - 1/e$ approximation factor to solve them effectively. The DP-based greedy algorithm, however, is not very efficient because of the expensive marginal gain evaluation. To further accelerate the greedy algorithm, we present an approximate greedy algorithm with linear time complexity w.r.t. the graph size. We show that the approximate greedy algorithm is also with near-optimal performance guarantee. Extensive experiments are conducted to evaluate the proposed algorithms. The results demonstrate the effectiveness, efficiency, and scalability of the proposed algorithms.

There are several future directions which deserve further investigation. First, recall that both the objective functions of Problem (1) and Problem (2) are submodular. An interesting problem is to combine these two objective functions (e.g., by a positive weight, it is still submodular) and then optimize both the total hitting time and the expected number of nodes that hit the targeted set simultaneously. Second, Problem (2) is to maximize the expected number of nodes that are dominated by

the targeted set. It would be interesting to extend this problem to maximize the expected number of edges that are traversed by the L-length random walk starting from every node to the targeted set. Finally, a related problem of Problem (2) is that given a parameter $\alpha \in [0, 1]$, the goal is to find the minimum number of targeted nodes such that they can dominate at least αn number of nodes in expectation. It would also be interesting to devise efficient algorithms for this issue.

APPENDIX

Proof of Theorem 2.2: By definition, we have the following facts.

Fact 1: If $0 < i < L$, we have $\Pr[T_{uv}^L = i] = \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1]$, and if $i = L$, $\Pr[T_{uv}^L = i] = \sum_{w \in V} p_{uw} \Pr[T_{wv}^L \geq i-1]$ holds.

Fact 2: If $0 < i < L-1$, we have $\Pr[T_{uv}^{L-1} = i] = \Pr[T_{uv}^L = i]$, and if $i = L-1$, we have $\Pr[T_{uv}^{L-1} = i] = \Pr[T_{uv}^L = i] + \Pr[T_{uv}^L = L]$.

Equipped with the above two facts, we can prove the theorem as follows. Clearly, if $u = v$, then $T_{uv}^L = 0$, and thereby $h_{uv}^L = 0$. If $u \neq v$, by definition, we have

$$\begin{aligned} h_{uv}^L = \mathbb{E}[T_{uv}^L] &= \textstyle\sum_{i=1}^{L} i \Pr[T_{uv}^L = i] \\ &= \textstyle\sum_{i=1}^{L-1} i \Pr[T_{uv}^L = i] + L \Pr[T_{uv}^L = L] \\ &= \textstyle\sum_{i=1}^{L-1} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1] \\ &\quad + L \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L \geq L-1] \\ &= \textstyle\sum_{i=1}^{L} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1] \\ &\quad + L \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L = L], \end{aligned} \tag{13}$$

where the third equation holds due to Fact 1. Then, we can further decompose Eq. (13) as follows.

$$\begin{aligned} h_{uv}^L &= \textstyle\sum_{i=1}^{L} (i-1) \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1] \\ &\quad + \textstyle\sum_{i=1}^{L} \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1] \\ &\quad + \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L = L] \\ &\quad + (L-1) \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L = L] \\ &= \textstyle\sum_{i=1}^{L} (i-1) \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i-1] \\ &\quad + (L-1) \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L = L] + 1, \end{aligned} \tag{14}$$

where the second equality holds due to $\sum_{i=1}^{L} \Pr[T_{wv}^L = i] = 1$ and $\sum_{w \in V} p_{uw} = 1$. Based on Eq. (14) and Fact 2, we have

$$\begin{aligned} h_{uv}^L &= \textstyle\sum_{i=1}^{L-1} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i] \\ &\quad + (L-1) \textstyle\sum_{w \in V} p_{uw} \Pr[T_{wv}^L = L] + 1 \\ &= \textstyle\sum_{i=1}^{L-2} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^L = i] \\ &\quad + (L-1) \textstyle\sum_{w \in V} p_{uw} (\Pr[T_{wv}^L = L-1] + \Pr[T_{wv}^L = L]) + 1 \\ &= \textstyle\sum_{i=1}^{L-2} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^{L-1} = i] \\ &\quad + (L-1) \textstyle\sum_{w \in V} p_{uw} (\Pr[T_{wv}^{L-1} = L-1]) + 1 \ \{\text{By Fact 2}\} \\ &= \textstyle\sum_{i=1}^{L-1} i \sum_{w \in V} p_{uw} \Pr[T_{wv}^{L-1} = i] + 1 \\ &= 1 + \textstyle\sum_{w \in V} p_{uw} h_{wv}^{L-1}. \end{aligned} \tag{15}$$

This completes the proof.

ACKNOWLEDGEMENTS

The work was supported in part by (i) grants GRF 418512, 411211, and 411310 from HKRGC, (ii) NSFC project 61170076 from China, and (iii) China-863 project 2012AA01A309.

REFERENCES

[1] K. Lerman and L. Jones, "Social browsing on flickr," in *ICWSM*, 2007.
[2] K. Lerman, "Social browsing & information filtering in social media," *CoRR*, vol. abs/0710.5697, 2007.
[3] M. R. Morris, J. Teevan, and K. Panovich, "A comparison of information seeking using search engines and social networks," in *ICWSM*, 2010.
[4] X. Si, E. Y. Chang, Z. Gyöngyi, and M. Sun, "Confucius and its intelligent disciples: Integrating social with search," *PVLDB*, vol. 3, no. 2, 2010.
[5] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks: Algorithms and evaluation," *Perform. Eval.*, vol. 63, no. 3, pp. 241–263, 2006.
[6] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions-i," *Mathematical Programming*, vol. 14, pp. 265–294, 1978.
[7] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of domination in graphs*. MARCEL DEKKER, INC, 1998.
[8] ——, *Domination in graphs: advanced topics*. MARCEL DEKKER, INC, 1998.
[9] U. Feige, "A threshold of ln *n* for approximating set cover," *J. ACM*, vol. 45, no. 4, 1998.
[10] Y. Wu and Y. Li, "Construction algorithms for k-connected m-dominating sets in wireless sensor networks," in *MobiHoc*, 2008.
[11] I. Stojmenovic, M. Seddigh, and J. D. Zunic, "Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 1, pp. 14–25, 2002.
[12] J. Wu, M. Cardei, F. Dai, and S. Yang, "Extended dominating set and its applications in ad hoc networks using cooperative communication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 8, pp. 851–864, 2006.
[13] M. Couture, M. Barbeau, P. Bose, and E. Kranakis, "Incremental construction of k-dominating sets in wireless sensor networks," *Ad Hoc & Sensor Wireless Networks*, vol. 5, no. 1-2, pp. 47–68, 2008.
[14] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," in *PODC*, 2003.
[15] E. Sampathkumar and H. Walikar, "The connected domination number of a graph," *J. Math. Phys. Sci*, vol. 13, no. 6, pp. 607–613, 1979.
[16] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20, no. 4, pp. 374–387, 1998.
[17] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003.
[18] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *KDD*, 2007.
[19] A. Krause, A. P. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, pp. 235–284, 2008.
[20] H. Lin and J. Bilmes, "A class of submodular functions for document summarization," in *ACL*, 2011.
[21] A. Krause and E. Horvitz, "A utility-theoretic approach to privacy and personalization," in *AAAI*, 2008, pp. 1181–1188.
[22] R.-H. Li and J. X. Yu, "Scalable diversified ranking on large graphs," in *ICDM*, 2011.
[23] ——, "Scalable diversified ranking on large graphs," in *IEEE Transactions on Knowledge and Data Engineering*, 2012.
[24] L. Lovasz, "Random walk on graphs: A survey," *Combinatorics*, vol. 2, pp. 1–46, 1993.
[25] P. Sarkar and A. W. Moore, "A tractable approach to finding closest truncated-commute-time neighbors in large graphs," in *UAI*, 2007.
[26] P. Sarkar, A. W. Moore, and A. Prakash, "Fast incremental proximity search in large graphs," in *ICML*, 2008.
[27] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.
[28] R.-H. Li, J. X. Yu, X. Huang, and H. Cheng, "Random-walk domination in large graphs: problem definitions and fast solutions," *Technical report, http://arxiv.org/abs/1302.4546*, 2013.
[29] J. Leskovec, "Standford network analysis project," 2010. [Online]. Available: http://snap.standford.edu
[30] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *science*, 1999.