On Modeling Influence Maximization in Social Activity Networks under General Settings

RUI WANG, University of Science and Technology of China
YONGKUN LI, University of Science and Technology of China and AnHui Province Key Laboratory of High Performance Computing
SHUAI LIN, University of Science and Technology of China
HONG XIE, Chongqing University
YINLONG XU, University of Science and Technology of China
JOHN C. S. LUI, The Chinese University of Hong Kong

Finding the set of most influential users in online social networks (OSNs) to trigger the largest influence cascade is meaningful, e.g., companies may leverage the "word-of-mouth" effect to trigger a large cascade of purchases by offering free samples/discounts to those most influential users. This task is usually modeled as an influence maximization problem, and it has been widely studied in the past decade. However, considering that users in OSNs may participate in various online activities, e.g., joining discussion groups and commenting on same pages or products, influence diffusion through online activities becomes even more significant. In this article, we study the impact of online activities by formulating social-activity networks which contain both users and online activities, and thus induce two types of weighted edges, i.e., edges between users and edges between users and activities. To address the computation challenge, we define an influence centrality via random walks, and use the Monte Carlo framework to efficiently estimate the centrality. Furthermore, we develop a greedy-based algorithm with novel optimizations to find the most influential users for node recommendation. Experiments on real-world datasets show that our approach is very computationally efficient under different influence models, and also achieves larger influence spread by considering online activities.

CCS Concepts: • **Information systems → Top-k retrieval in databases**; • **Social and professional topics** → *Consumer products policy*;

Additional Key Words and Phrases: OSN, user activities, influence maximization, random walk

ACM Reference format:
Rui Wang, Yongkun Li, Shuai Lin, Hong Xie, Yinlong Xu, and John C. S. Lui. 2021. On Modeling Influence Maximization in Social Activity Networks under General Settings. *ACM Trans. Knowl. Discov. Data* 15, 6, Article 108 (May 2021), 28 pages.
https://doi.org/10.1145/3451218

The work by Yongkun Li was supported in part by NSFC 61772484 and Youth Innovation Promotion Association CAS. The work by John C. S. Lui was supported in part by the GRF 14200420.

Authors' addresses: R. Wang, S. Lin, and Y. Xu, University of Science and Technology of China, 96 Jinzhai Rd, Baohe District, Hefei; emails: {rwang067, shuailin}@mail.ustc.edu.cn, ylxu@ustc.edu.cn; Y. Li (corresponding author), University of Science and Technology of China and AnHui Province Key Laboratory of High Performance Computing, 96 Jinzhai Rd, Baohe District, Hefei; email: ykli@ustc.edu.cn; H. Xie, Chongqing University, No.174 Shazhengjie, Shapingba, Chongqing; email: xiehong2018@cqu.edu.cn; J. C. S. Lui, The Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR; email: cslui@cse.cuhk.edu.hk.

1556-4681/2021/05-ART108 $15.00
https://doi.org/10.1145/3451218

1 INTRODUCTION

Due to the popularity of **online social networks (OSNs)**, and their excellent capability to spread information, ideas, and innovations, viral marketing which exploits the "word-of-mouth" effect is commonly used by companies to promote product sales [32]. For example, consider the scenario that a company seeks to boost the adoption of a new product. One approach is to choose a set of influential users (i.e., seed set) in an OSN and give them a discount or free samples. These initial influential users may recommend the product to their friends, and their friends may further attract friends of their friends to adopt the product. Finally, a large amount of people may buy the product due to the influence spread through out the whole network. One key algorithmic issue is how to find the seed set so as to trigger the largest influence spread or product adoption. This viral marking problem can be modeled as **influence maximization problem (IMP)**, which was first formulated by Kempe et al. [29] as follows: given an OSN and an information diffusion model, how to select a set of k users, which is called the seed set, so as to trigger the largest influence spread over the OSN under the difusion model. Besides viral marketing, IMP also has been applied in many other applications such as rumor blocking, which selects k seed users to trigger the spread of a positive cascade to maximize the amount users not being influenced by a rumor [8, 22, 42]. More applications of IMP include network monitoring [3], social recommendation [48], and the like [19–21]. Due to its wide range of applications and its NP-hardness [9, 11], IMP has been studied extensively in the past decade [4, 6, 10, 16–18, 33, 40, 41].

We notice that current works mainly focus on using only the "friendship" relationships to spread influence between users, while ignoring the fact that users in today's OSNs may participate in various kinds of online activities, e.g., joining a discussion group and clicking `like` or comment on Facebook. Hence, users not only can create friendship relationships, which we call *user–user* links, but can also form relationships by participating in online activities, which we call *user–activity–user links.* For example, two users in Facebook form a user-activity-user link no matter they are friends or not, if there is a discussion group that both of them join or a public page that both of them write comments on. They may influence each other during the discussion. They may also read each other's comments or visit each other's homepage and get influenced. To distinguish with traditional OSNs, we call this kind of networks which contain both user–user relationships and user–activity–user relationships as ***social-activity networks*** **(SANs)**.

With the consideration of online activities in SANs, influence may also spread through the user-activity-user links as well as the user-user links. In this article, we focus on the online activities which generate positive influence, e.g., clicking `like` on the same public page in Facebook, giving high rating to the same product in online rating systems, and joining in a community sharing the same interest in OSNs. Due to the large amount of online activities, many pair of users may participate in multiple common online activities. These pair of users may influence each other more even though they are not friends. This is because they may have a lot of common interests or common needs which drive them to participate these common activities. Thus, influence diffusion through the user–activity–user links becomes more significant. Therefore, one need to considering social activities in selecting the seed set. However, social activities are ignored by previous works on influence maximization. *This motivates us to formulate the influence maximization problem for SANs, and to determine the most influential nodes by taking online activities into consideration.*

However, modeling influence maximization with online activities is challenging. First, influence maximization in OSNs without online activities was already proved to be NP-hard, and considering online activities makes this problem even more complicated. Second, the amount of online activities in a SAN is very large even for small OSNs, this is because online activities happen more frequently than friendship formation in OSNs. As a result, the underlying graph which

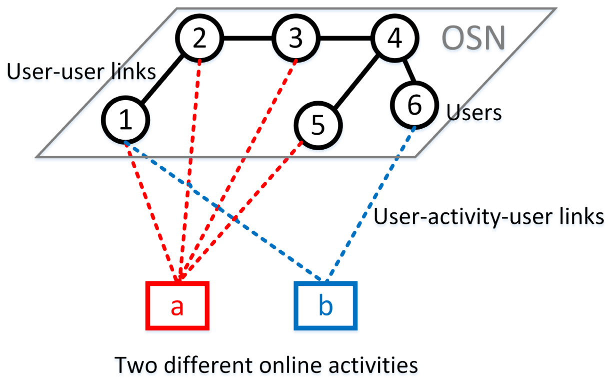


Fig. 1. An example of SAN.

characterizes users and their relationships may become extremely dense if we transform the user–activity–user links to user–user links, so it requires highly efficient algorithms for finding the most influential nodes. To address the above challenges, in this article, we make the following contributions.

— We generalize the influence diffusion models for SANs by modeling SANs as hypergraphs. We approximate the influence of nodes in SANs by defining an influence centrality based on random walk. Our influence centrality applies to both unweighted and weighted graphs.
— We employ the Monte Carlo framework to estimate the influence centrality in SANs, and also develop a greedy-based algorithm with two novel optimization techniques to solve the influence maximization problem under both **independent cascade (IC)** model and **linear threshold (LT)** model for SANs.
— We conduct experiments with six real-world datasets, and results show that our approach is more efficient while keep almost the same accuracy compared to the the most popular influence maximization algorithm IMM [40].

This article is organized as follows. In Section 2, we formulate the influence maximization problem for SANs. In Section 3, we present our random walk-based methodology. In Section 4, we present the Monte Carlo method to estimate the influence centrality in SANs. In Section 5, we present our greedy-based algorithm and optimization techniques to solve the influence maximization problem. In Section 6, we present the experimental results. Related work is given in Section 7 and Section 8 concludes.

2 PROBLEM FORMULATION

In this section, we first model the SAN with a hypergraph, and then formulate the influence maximization problem for SANs.

2.1 Model for SANs

We use a hypergraph $G(V, E, \mathcal{E}_1, \dots, \mathcal{E}_l)$ to characterize a SAN, where V denotes the set of users, E denotes the *user-user* links, and $\mathcal{E}_i$ ($i = 1, 2, \dots, l$) denotes the set of type i hyperedges in which each hyperedge is a set of users who participated in the same online activity, and represented as a tuple. Considering Figure 1, only activity a is of the first type, so $\mathcal{E}_1 = \{(1, 2, 3, 5)\}$. For ease of presentation, we denote $N(j)$ as the set of neighbors of user j, i.e., $N(j) = \{i|(i,j) \in E\}$, $M_e(j)$ as the set of users except for user j who connected to the hyperedge e, i.e., $M_e(j) = \{i|i \in e, \&i \neq j\}$, and denote $\mathcal{E}_t(j)$ as the set of type t hyperedges that are connected to user j, i.e., $\mathcal{E}_t(j) = \{e|e \in \mathcal{E}_t \& j \in e\}$. Considering Figure 1, $N(1) = \{2\}$, $M_e(1) = \{2,3,5\}$ when $e = (1,2,3,5)$ and $\mathcal{E}_1(1) = \{(1,2,3,5)\}$.

2.2 Influence Maximization in SANs

To solve the influence maximization problem, we first introduce the influence diffusion process. Before describing the influence diffusion process over SANs, we first introduce the two typical influence diffusion models over OSNs, i.e., the IC model and the LT model [29]. Each user has two states, i.e., either active or inactive. Initially, all users are in the state of inactive. We select a set of users change their state to active. Let $N(i)$ denote a set of all the neighbors user i. Let $d_i = |N(i)|$ denote the degree of user i. In the IC model, each active user i will activate each of her inactive neighbor $j \in N(i)$ to be active with probability $q_{ij} (0 \leq q_{ij} \leq 1)$. Typically, the activation probability is set to $q_{ij} = 1/d_j$ [9, 10, 29, 40, 41]. After a neighbor j being activated, she will further activate her inactive neighbors in the set $N(j)$, and this diffusion process continues until no user can change her state. In LT model, each user i is associated with a threshold $\theta_i (\theta_i \in [0,1])$. The state of an inactive user i will be changed into active if at least a fraction θ_i of her neighbors are active. The diffusion process continues until no inactive user can be further activated. We call the expected size of the final set of active users the *influence spread*, and denote it as $\sigma(S(k))$ if the set of k initial active users is $S(k)$.

Now we describe the influence diffusion process for SANs. The key issue is to define the influence between user i and user j (i.e., g_{ij}) after taking online activities into consideration. Our definition is based on three criteria:

— A user may make a purchase due to her own interest or being influenced by others through user–user or user–activity–user links, so we define the total influence probability by one-hop neighbors as c $(0 < c < 1)$, and call it the decay parameter. As we have l types of online activities, we define α_{jt} (where $0 \leq \alpha_{jt} \leq 1$ and $0 \leq \sum_{t=1}^{l} \alpha_{jt} \leq 1$) as the proportion of influence to user j through type t online activities, and call it *weight of activities*. Clearly, $1 - \sum_{t=1}^{l} \alpha_{jt}$ indicates the proportion of influence from direct neighbors.
— For the influence to user j from direct neighbors, we define the weight of each neighbor i $(i \in N(j))$ as u_{ij}, and assume that $0 \leq u_{ij} \leq 1$ and $\sum_{i \in N(j)} u_{ij} = 1$.
— For the influence to user j through the type t online activities, we define the weight of each online activity a as v_{aj}, where $0 \leq v_{aj} \leq 1$ and $\sum_{a \in N_t(j)} v_{aj} = 1$, where $N_t(j)$ means online activities of type t that user j participated, and we can also represent these activities as a set of hyperedges as $\mathcal{E}_t(j)$. Besides, considering that maybe multiple users participated in the same online activity a, we define the weight of each user i who participated in a as u_{ij}^a $(i \in N(a)\backslash\{j\})$, and assume that $0 \leq u_{ij}^a \leq 1$ and $\sum_{i \in N(a)\backslash\{j\}} u_{ij}^a = 1$, where $N(a)$ means all users participated in activity a. If we represent the activity a as a hyperedge e, then $N(a)\backslash\{j\}$ can also be represented as $M_e(j)$.

For simplicity, we set $u_{ij} = 1/|N(j)|$ for unweighted graphs. Note that this uniform setting is exactly the same as the IC model and LT model in OSNs, which is a common setting for unweighted graphs and has been widely studied in [9, 10, 29, 40, 41]. Similarly, we also let $v_{aj} = 1/|\mathcal{E}_t(j)|$ and $u_{ij}^a = 1/|M_e(j)|$ by following the uniform setting. We would like to point out that our random walk approach also applies to general settings, e.g., we can simply reset u_{ij}, v_{aj} and u_{ij}^a according to the weights of weighted graphs. Now we can define the influence of user i to user j, which we denote

Table 1. List of Notations

Symbol	Meaning
V	Set of users
E	Set user–user links
l	Number of activity types
$\mathcal{E}_i$ $(i = 1, 2, \ldots, l)$	Set of type i hyperedges
$G(V, E, \mathcal{E}_1, \ldots, \mathcal{E}_l)$	Hypergraph to characterize a SAN
$N(j)$	Set of neighbors of user j
$M_e(j)$	Set of users connected to user j by hyperedge e
$\mathcal{E}_t(j)$	set of type t hyperedges connected to user j
d_i	Degree of user i
q_{ij}	Influence probability from user i to user j in IC model
θ_i	Threshold of user i can be activated
$S(k)$	Set of k initial active users
$\sigma(S(k))$	Expected size of the final set of active users
c $(0 < c < 1)$	Decay parameter
α_{jt}	Proportion of influence to user j through online activity of type t
$1 - \sum_{t=1}^{l} \alpha_{jt}$	Proportion of influence from direct neighbors
u_{ij}	Weight of each neighbor i $(i \in N(j))$
v_{aj}	Weight of each online activity a
u_{ij}^{a}	weight of each user i participated in a, $(i \in N(a)\backslash\{j\})$
g_{ij}	Influence of user i to user j

as g_{ij}:

$$g_{ij} = c \times \Bigg(\frac{1 - \sum_{t=1}^{l} \alpha_{jt}}{|N(j)|} \times \mathbf{1}_{\{i \in N(j)\}} + \sum_{t \in [1,l]} \sum_{e \in \mathcal{E}_t(j)} \frac{\alpha_{jt}}{|\mathcal{E}_t(j)|} \times \frac{1}{|M_e(j)|} \times \mathbf{1}_{\{i \in M_e(j)\}} \Bigg). \tag{1}$$

The first term in the right hand side of Equation (1) denotes the influence diffused through user-user links. The proportion of influence to user j through user-user links equals $1 - \sum_{t=1}^{l} \alpha_{jt}$. The influence of user i to user j is set as $1/|N(j)|$, if $i \in N(j)$. The second term of Equation (1) represents the influence diffused through user–activity–user links. There are l types of activities in total. The proportion of influence to user j through user–activity–user links associated with type t activity is set as α_{jt}. The influence of user i to user j equals $\frac{1}{|\mathcal{E}_t(j)|} \times \frac{1}{|M_e(j)|}$, if $i \in M_e(j)$. This captures that in the influence diffusion process, an activity e is first selected from type t edges uniformly at random, and then a user is selected from users connected to user j by hyper-edge e uniformly at random. The parameter c captures the influence decay rate during influence diffusion.

Now we formulate the influence maximization problem for SANs, which we denote as **IMP(SAN)**, as follows.

Definition 1. **IMP(SAN):** Given a SAN $G(V, E, \mathcal{E}_1, \ldots, \mathcal{E}_l)$, an influence diffusion model with parameters α_{jt}, find a set of k nodes $S(k)$, where k is an integer, so as to make the influence spread $\sigma(S(k))$ maximized.

For ease of read, we list the mainly used notations in Table 1.

3 METHODOLOGY

In this section, we present our methodology to address the (**IMP(SAN)**) problem. One key issue is to measure the influence spread $\sigma(S(k))$ of a node set $S(k)$. To achieve it, one needs to compute the probability of each user being influenced by $S(k)$, which involves iteratively computing the influence probability by influence g_{ij} for each user pair i and j. This computation is computationally expensive, therefore, the accurate computation for $\sigma(S(k))$ is unrealistic. To reduce the large computation cost, we first develop a random walk framework on hypergraphs to estimate the influence diffusion process. Then, we define a centrality measure based on random walk to approximate the influence of a node set. With this centrality measure, we can approximate the influence maximization problem by solving a centrality maximization problem.

3.1 Random Walk on Hypergraph

Here, we present our random walk-based framework, which is extended from the classical random walk on a simple graph $G(V, E)$, which can be stated as follows. For a random walk at vertex i $(i \in V)$, it randomly selects a neighbor j $(j \in N(i))$, and then moves to j in the next step. Mathematically, let $Y(t)$ denote the position of the walker at step t, then $\{Y(t)\}$ constitutes a Markov chain with the one-step transition probability p_{ij}, where $p_{ij} = 1/|N(i)|$ for unweighted graphs and $p_{ij} = C_{ij}/\sum_{k\in i} C_{ik}$ for weighted graphs. Here, C_{ij} represents the weight of edge e_{ij}, if $(i, j) \in E$ and $C_{ij} = 0$ otherwise.

We now define the one-step transition probability p_{ij} when performing a random walk on the hypergraph $G(V, E, \mathcal{E}_1, \ldots, \mathcal{E}_l)$. Note that each hyperedge may contain more than two vertices, so we take the one-step random walk from user i to user j as a two-step process.

—**Step one:** Choose a hyperedge associated to user i. Precisely, according to the influence diffusion models in Section 2.2, we set the probability of selecting type t hyperedges as α_{it}, and choose hyperedges of the same type at random. Mathematically, if the walker is currently at user i, then it chooses a hyperedge e of type t with probability $\frac{\alpha_{it}}{|\mathcal{E}_t(i)|}$.

Note that, α_{it} quantifies the strength that type t activities influence user i. It can be learned from historical data or solicited from user input. We consider the case that α_t is given and we focus on selecting a subset of nodes to boost influence spreading. Note that this setting is similar with most previous works on influence maximization, which assume that the influence model is given. We vary α_t to study the impact of online activities on seed set selection in Section 6.2. Experiment results show that the selected seed set can bring a larger influence spreading compared with the seed set selected by previous works ignoring activities in seed selection. Besides, our method outperforms previous works more under a larger influence strength from activities, i.e., larger α.

—**Step two:** Choose a user associated to the hyperedge e selected in step one as the next stop of the random walk. We consider random walks without backtrace. In particular, if a walker is currently at node i, then we select the next stop randomly from the vertices that are connected to the same hyperedge with user i. We define the probability of choosing user j as $1/|M_e(i)|$.

By combing the two steps defined above, we can derive the transition probability from user i to j as follows.

$$\begin{aligned} p_{ij} = &\frac{1 - \sum_{t=1}^{l} \alpha_{it}}{|N(i)|} \times \mathbf{1}_{\{j \in N(i)\}} \\ &+ \sum_{t\in[1,l]} \sum_{e\in\mathcal{E}_t(i)} \frac{\alpha_{it}}{|\mathcal{E}_t(i)|} \times \frac{1}{|M_e(i)|} \times \mathbf{1}_{\{j\in M_e(i)\}}. \end{aligned} \tag{2}$$

The influence of user j over user i is $g_{ji} = c \times p_{ij}$.

3.2 Influence Centrality Measure

Since the accurate computation of $\sigma(S(k))$ is computationally expensive as we mentioned before, to measure the influence spread of a node set, we define a centrality measure based on random walks on hypergraphs to approximate the influence of a node set $S(k)$. We call it *influence centrality*, and denote it as $I(S)$, which is defined as follows.

$$I(S) = \sum\nolimits_{j \in V} h(j, S), \tag{3}$$

where $h(j, S)$ aims to approximate the influence of S to j, which is called decayed hitting probability. It is defined as

$$h(j, S) = \begin{cases} \sum_{i \in V} c p_{ji} h(i, S), j \notin S, \\ 1, j \in S, \end{cases} \tag{4}$$

where c is the decay parameter defined in Section 2.2, and p_{ji} is the one-step transition probability defined in Equation (2). Equation (4) aims to approximate the influence of S to user j of any-step connected to S by the recursive computation. We approximate the influence of S to j of any-step connected to S, i.e., $h(j, S)$, iteratively. More specifically, we start the first iteration via an initial guess on $h(j, S)$ and then use the right hand side of Equation (4) to update the estimation of $h(j, S)$. The decaying parameter $c \in (0, 1)$ governs the convergence of this iteration.

To solve the influence maximization problem of **IMP(SAN)**, we use the influence centrality measure $I(S)$ to approximate the influence of the node set S, and our goal is to find a set S of k users so that $I(S)$ is maximized. In other words, we approximate the influence maximization problem **IMP(SAN)** by solving the centrality maximization problem **CMP** defined as follows.

Definition 2. **CMP:** Given a hyperghraph $G(V, E, \mathcal{E}_1, \ldots, \mathcal{E}_l)$ and the corresponding parameters α_{jt}, find a set S of k nodes, where k is an integer, so as to make the influence centrality of the set S of k nodes $I(S)$ maximized.

Note that, the key difference between **IMP(SAN)** and **CMP** is the definition of influence spread of a node set. In **IMP(SAN)**, the influence spread of a node set is defined as an accurate value $\sigma(S(k))$, i.e., the sum of probabilities of each user being activated by the initial node set $S(k)$, which is computationally expensive. While in **CMP**, the influence spread of a node set is defined as an estimated value based on random walks on hypergraphs $I(S)$, i.e., the sum of decayed hitting probabilities $h(j, S)$ from S to j for each user j. In summary, we use the process of random walks on hypergraphs to simulate the process of influence diffusion among SANs. We did not provide the analysis of theoretical bounds here, but we conduct the experiments on real world datasets and compared our methods with the most popular influence maximization algorithm with theoretical bounds, i.e., IMM [40], in Section 6.3. Experiment results show that our algorithm achieves almost the same accuracy in seed selection as IMM, while it only requires much less running time.

4 CENTRALITY COMPUTATION

We note that the key challenge of solving the centrality maximization problem **CMP** is how to efficiently estimate the influence centrality of a node set $I(S)$, or the decayed hitting probability $h(j, S)$. We give an efficient framework to estimate $h(j, S)$ as follows. We first rewrite $h(j, S)$ in a linear expression which is an infinite converging series, and then truncate the converging series to save computation time (see Section 4.1). To further estimate the truncated series, we first explain the expression with a random walk approach, and then use a Monte Carlo framework via random walks to estimate it efficiently (see Section 4.2).

4.1 Linear Expression

We first transform $h(j,S)$ defined in Equation (4) to a linear expression.

Theorem 1. *The decayed hitting probability $h(j,S)$ can be rewritten as*

$$h(j,S) = c e_j^T Q' e + c^2 e_j^T Q Q' e + c^3 e_j^T Q^2 Q' e + \cdots . \tag{5}$$

where Q is a $(|V|-|S|)\times(|V|-|S|)$ dimensional matrix which describes the transition probabilities between two nodes in the set $V-S$, Q' is a $(|V|-|S|)\times|S|$ dimensional matrix which describes the transition probabilities from a node in $V-S$ to a node in S, I is an identity matrix, e is a column vector with all elements being 1, and finally e_j is a column vector with only the element corresponding to node j being 1 and 0 for all other elements.

Proof. Based on the definition of $h(j,S)$ in Equation (4), we can get

$$h(j,S) = \sum_{h=1}^{\infty} c^h P(j,S,h), \text{ for } j \notin S,$$

where $P(j,S,h)$ denotes the probability that a random walk starting from j hits a node in S at the h-th step. Now if we denote p_{iS} as the probability that a random walk starting from i hits a node in S in one step, we have

$$\begin{aligned} h(j,S) &= \sum_{h=1}^{\infty} c^h P(j,S,h) = \sum_{h=1}^{\infty} c^h \sum_{i\notin S} (Q^{h-1})_{ji} p_{iS}, \\ &= \sum_{i\notin S}\sum_{h=1}^{\infty} c^h (Q^{h-1})_{ij} p_{iS} = \sum_{i\notin S} c(I-c\,Q)^{-1}_{ji} p_{iS}, \\ &= c e_j^T (I - cQ)^{-1} Q' e. \end{aligned}$$

Note that the largest eigenvalue of $c\,Q$ is less than one, so by further expanding the expression above with an infinite series, we can rewrite $h(j,S)$ as follows.

$$h(j,S) = c e_j^T Q' e + c^2 e_j^T Q Q' e + c^3 e_j^T Q^2 Q' e + \cdots . \qquad \square$$

We only keep the L leading terms of the infinite series, and denote the truncated result as $h^L(j,S)$, so we have

$$h^L(j,S) = c e_j^T Q' e + c^2 e_j^T Q Q' e + \cdots + c^L e_j^T Q^{L-1} Q' e. \tag{6}$$

Since c is defined as $0 < c < 1$, the series truncation error is bounded as follows.

$$0 \le h(j,S) - h^L(j,S) \le c^{L+1}/(1-c). \tag{7}$$

Based on the above error bound, we can see that $h^L(j,S)$ converges to $h(j,S)$ with rate c^{L+1}. This implies that if we want to compute $h(j,S)$ with a maximum error ϵ $(0 \le \epsilon \le 1)$, we only need to compute $h^L(j,S)$ by taking a sufficiently large enough L, or $L \ge \lceil \frac{\log(\epsilon - \epsilon c)}{\log c} \rceil - 1$.

4.2 Monte Carlo Algorithm

In this subsection, we present a Monte Carlo algorithm to efficiently approximate $h^L(j,S)$. Our algorithm is inspired from the random walk interpretation of Equation (6), and it can achieve a high accuracy with a small number of walks.

Consider the random walk interpretation of a particular term $e_j^T Q^{t-1} Q' e$ $(t = 1,\ldots,L)$ in Equation (6). Let us consider a L-step random walk starting from $j \notin S$ on the hypergraph. At each step, if the walker is currently at node k $(k \notin S)$, then it selects a node i and transits to i with probability p_{ki}, which is defined in Equation (2). As long as the walker hits a node in S, then it stops. Let $j^{(t)}$ be the t-th step position, and define an indicator $X(j,t)$ as

$$X(j,t) = \begin{cases} 1, & j^{(t)} \in S, \\ 0, & j^{(t)} \notin S. \end{cases}$$

We can see that $e_j^T Q^{t-1} Q' e$ is the probability that a random walk starting from j hits a node in S at the t-th step. We have

$$e_j^T Q^{t-1} Q' e = E[X(j,t)]. \tag{8}$$

By substituting $e_j^T Q^{t-1} Q' e$ with Equation (8), we can rewrite $h^L(j,S)$ as

$$h^L(j,S) = cE[X(j,1)] + c^2 E[X(j,2)] + \cdots + c^L E[X(j,L)]. \tag{9}$$

Now we estimate $h^L(j,S)$ by using a Monte Carlo method with random walks on the hypergraph based on Equation (9). Specifically, for each node j where $j \notin S$, we set R independent L-step random walks starting from j. We denote the t-th step position of the R random walks as $j_1^{(t)}, j_2^{(t)}, \ldots, j_R^{(t)}$, respectively, and use $X_r(j,t)$ to indicate whether $j_r^{(t)}$ belongs to set S or not. Precisely, we set $X_r(j,t) = 1$ if $j_r^{(t)} \in S$, and 0 otherwise, so $c^t E[X(j,t)]$ can be estimated as

$$c^t E[X(j,t)] \approx \frac{c^t}{R} \sum_{r=1}^{R} X_r(j,t).$$

By substituting $c^t E[X(j,t)]$ in Equation (9), we can approximate $h^L(j,S)$, which we denote as $\hat{h}^L(j,S)$, as follows.

$$\hat{h}^L(j,S) = \frac{c}{R} \sum_{r=1}^{R} X_r(j,1) + \cdots + \frac{c^L}{R} \sum_{r=1}^{R} X_r(j,L). \tag{10}$$

Algorithm 1 presents the process of the Monte Carlo method described above. We can see that its time complexity is $O(RL)$ as the number of types of online activities l is usually a small number. In other words, we can estimate $h^L(j,S)$ in $O(RL)$ time and compute $I(S)$ in $O(nRL)$ time as we need to estimate $h^L(j,S)$ for all nodes. The main benefit of this Monte Carlo algorithm is that its running time is independent of the graph size, so it scales well to large graphs.

Note that $\hat{h}^L(j,S)$ computed with Algorithm 1 is an approximation of $h^L(j,S)$, and the approximation error depends on the sample size R. To estimate the number of samples required to compute $h^L(j,S)$ accurately, we derive the error bound by applying Hoeffding inequality [23], and the results are as follows.

THEOREM 2. *Let the output of Algorithm 1 be $\hat{h}^L(j,S)$, then we have*

$$P\{|\hat{h}^L(j,S) - h^L(j,S)| > \epsilon\} \leq 2L \exp(-2(1-c)^2 \epsilon^2 R). \tag{11}$$

PROOF. Let $X_1, \ldots, X_R$ be R independent random variables with $X_r \in [0,1]$ for all $r = 1, \ldots, R$, and set $T = (X_1 + \cdots + X_R)/R$. According to Hoeffding's inequality, we have

ALGORITHM 1: Monte Carlo Estimation for $h^L(j,S)$

```
1: function h^L(j, S)
2:     $\sigma \leftarrow 0$;
3:     for $r = 1$ to $R$ do
4:         $i \leftarrow j$;
5:         for $t = 1$ to $L$ do
6:             Generate a random number $x \in [0,1]$;
7:             for $T = 0$ to $l$ do
8:                 if $x \le \alpha_{iT}$ then                    ▷ $\alpha_{0T} = 1 - \sum_{T=1}^{l} \alpha_{iT}$;
9:                     $E \leftarrow \mathcal{E}_T(i)$;
10:                    break;
11:                $x \leftarrow x - \alpha_{iT}$;
12:            Select a hyperedge $e$ from $E$ randomly;
13:            $i \leftarrow$ select a user from $\{k | k \in e, k \neq i\}$ randomly;
14:            if $i \in S$ then
15:                $\sigma \leftarrow \sigma + c^t/R$;
16:                break;
17:    return $\sigma$;
18: end function
```

$P\{|T - E(T)| > \epsilon\} \le 2\exp(-2\epsilon^2 R)$. By applying this inequality, we have

$$\begin{aligned}
&P\{|\hat{h}^L(j,S) - h^L(j,S)| > \epsilon\} \\
&= P\Big\{\Big|\sum_{t=1}^{L} \frac{c^t}{R} \sum_{r=1}^{R} X_r(j,t) - \sum_{t=1}^{L} c^t E[X(j,t)]\Big| > \epsilon\Big\}, \\
&\le P\Big\{\sum_{t=1}^{L} \Big|\frac{c^t}{R} \sum_{r=1}^{R} X_r(j,t) - c^t E[X(j,t)]\Big| > \epsilon\Big\}, \\
&\le \sum_{t=1}^{L} P\Big\{\Big|\frac{c^t}{R} \sum_{r=1}^{R} X_r(j,t) - c^t E[X(j,t)]\Big| > (1-c)c^t\epsilon\Big\}, \\
&\le 2L\exp(-2(1-c)^2\epsilon^2 R).
\end{aligned}$$ □

Based on Theorem 2, we see that Algorithm 1 can estimate $h^L(j,S)$ with a maximum error ϵ with least probability $1-\delta$ ($0 < \delta, \epsilon < 1$) by setting $R \ge \log(2L/\delta)/(2(1-c)^2\epsilon^2)$.

5 CENTRALITY MAXIMIZATION

In this section, we develop efficient algorithms to address the centrality maximization problem **CMP** defined in Section 3.2. Noted that even though we can efficiently estimate the decayed hitting probability $h(j,S)$ by using random walks (see Section 4), finding a set S of k nodes in a SAN to maximize its influence centrality $I(S)$ is still computationally difficult as it requires to estimate the influence centrality of all combinations of k nodes. In particular, **CMP** is NP-hard.

Theorem 3. *The centrality maximization problem* **CMP** *is NP-hard.*

Proof. We first briefly introduce the vertex cover problem. An instance of vertex cover problem is specified by a graph $G(V,E)$ and an integer k, and asks there exists a vertex set $S \subseteq V$ such that $|S| \le k$ and for every $(i,j) \in E$, $\{i,j\} \cap S \neq \emptyset$.

We now map our centrality maximization problem into an instance of the vertex cover problem by taking the same graph $G(V, E)$ and asking whether there exists a vertex set S such that $|S| \le k$ and $I(S) \ge (n-k) \times c + k$. We aim to show that S is a vertex cover if and only if $I(S) \ge (n-k) \times c + k$. Assuming S is a vertex cover. Note that $h(i, S) = 1$ if $i \in S$ and $h(i, S) = c$ otherwise. Observe that for every $(i, j) \in E$, $\{i, j\} \cap S \neq \emptyset$. This implies that $I(S) = (n-k) \times c + k$. Suppose S is not a vertex cover, then there should be an edge (u, v) which satisfies that $\{u, v\} \cap S = \emptyset$. A random walk from u passing through v and at last arriving at S will have length at least 2. So $h(u, S) = \sum_{j \in N(u)/v} cp_{uj} h(j, S) + cp_{uv} h(v, S)$. Due to $v \notin S$, $h(v, S) < 1$. Thus $h(u, S) < c$, which contradicts. Therefore S is a vertex cover. □

To solve the centrality maximization problem **CMP**, we develop greedy-based approximation algorithms by exploiting the submodularity property of $I(S)$. Specifically, we first show the submodularity property and present a baseline greedy algorithm to maximize $I(S)$, and then develop two novel optimization techniques to accelerate the greedy algorithm.

5.1 Baseline Greedy Algorithm

Before presenting the greedy-based approximation algorithm for maximizing $I(S)$, we first show that $I(S)$ is a non-decreasing submodular function, and the result is stated in the following theorem.

Theorem 4. *$I(S)$ is a non-decreasing submodular function.*

Proof. We first show the non-decreasing property. Note that since $h(j, S) = 1$ if $j \in S$, so we rewrite $I(S)$ as follows.

$$I(S) = |S| + \sum\nolimits_{j \in (V-S)} h(j, S).$$

Now suppose that a user $u \notin S$ is added into the set S, then the marginal increment of the influence centrality $\Delta(u) = I(S \cup \{u\}) - I(S)$ can be derived as

$$\begin{aligned}\Delta(u) &= \sum_{j \in V} h(j, S \cup \{u\}) - \sum_{j \in V} h(j, S), \\ &= 1 + \sum_{j \in (V - S \cup \{u\})} h(j, S \cup \{u\}) - \sum_{j \in (V-S)} h(j, S), \\ &= 1 - h(u, S) + \sum_{j \in (V - S \cup \{u\})} \Big[h(j, S \cup \{u\}) - h(j, S) \Big].\end{aligned}$$

According to the definition of $h(j, S)$ in Equation (5) and the random walk interpretation, we rewrite $h(j, S)$ as

$$h(j, S) = \sum_{h=1}^{\infty} c^h P(j, S, h), \text{ for } j \notin S,$$

ALGORITHM 2: Baseline Greedy Alg. for Maximizing $I(S)$

```
Inputs: A hypergraph, and a parameter $k$;
Output: A set $S$ of $k$ nodes for maximizing $I(S)$;
1: $S \leftarrow \emptyset, I(S) \leftarrow 0$;
2: for $s = 1$ to $k$ do
3:     for $u \in (V - S)$ do
4:         $I(S \cup \{u\}) \leftarrow 0$;
5:         for $j \in (V - S \cup \{u\})$ do
6:             $I(S \cup \{u\}) \leftarrow I(S \cup \{u\}) + h(j, S \cup \{u\})$;
7:     $v \leftarrow \arg\max_{u \in (V-S)} I(S \cup \{u\}) - I(S)$;
8:     $S \leftarrow S \cup \{v\}$;
```

where $P(j, S, h)$ denotes the probability that a random walk starting from j hits a node in S at the h-th step for the first time. Now we can rewrite $h(j, S \cup \{u\}) - h(j, S)$ as

$$\begin{aligned} & h(j, S \cup \{u\}) - h(j, S) \\ = & \sum_{h=1}^{\infty} c^h P(j, S \cup \{u\}, h) - \sum_{h=1}^{\infty} c^h P(j, S, h) \\ = & \sum_{h=1}^{\infty} c^h \left[P^{\{u\}}(j, S, h) + P^S(j, \{u\}, h)\right] - \sum_{h=1}^{\infty} c^h \left[P^{\{u\}}(j, S, h) + P^S(j, \{u\}, h) P(u, S, h)\right] \\ = & \sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h) \left[1 - \sum_{h=1}^{\infty} c^h P(u, S, h)\right] \\ = & \sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h) \left[1 - p(u, S, h)\right], \end{aligned}$$

where $P^T(j, S, h)$ represents the probability that a random walk starting from j hits a node in S at the h-th step for the first time without passing any node in T. Therefore, $\Delta(u)$ can be derived as follows.

$$\begin{aligned} \Delta(u) &= I(S \cup \{u\}) - I(S) \\ &= (1 - h(u, S)) \left[1 + \sum_{j \in V - S \cup \{u\}} \sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h)\right]. \end{aligned} \tag{12}$$

Note that $0 < c < 1$ and $\sum_{h=1}^{\infty} P(u, S, h) \leq 1$, so we have $h(u, S) \leq 1$ and $1 - h(u, S) \geq 0$. That is, $\Delta(u) \geq 0$, and $I(S)$ is a non-decreasing function. We now show that $I(S)$ is a submodular function. Mathematically, we only need to prove that the inequality $I(S \cup \{u\}) - I(S) \geq I(T \cup \{u\}) - I(T)$, for $S \subseteq T$, holds. Note that $P^S(j, \{u\}, h) \geq P^T(j, \{u\}, h)$ if $S \subseteq T$. Besides, according to the non-decreasing feature of $I(S)$, we have $h(u, S) \leq h(u, T)$. Based on these inequalities and Equation (12), we can obtain $I(S \cup \{u\}) - I(S) \geq I(T \cup \{u\}) - I(T)$ if $S \subseteq T$. Therefore, $I(S)$ is a submodular function. □

Based on the submodularity property, we develop a greedy algorithm for approximation when maximizing $I(S)$, and we call it *the baseline greedy algorithm*. Algorithm 2 describes this procedure. To find a set of k nodes to maximize $I(S)$, the algorithm works for k iterations. In each iteration, it selects the node which maximizes the increment of $I(S)$.

Recall that the time complexity for estimating the influence of a set S to a particular node $j \notin S$, i.e., $h(j, S)$, is $O(RL)$ (see Section 4.2). Thus, the total time complexity for the baseline greedy algorithm is $O(kn^2RL)$ where n denotes the total number of users in the SAN, because estimating the influence of a set $S \cup \{u\}$ requires us to sum up its influence to all nodes, and we need to check every node u so as to select the one which maximizes the increment of $I(S)$. Although the baseline greedy algorithm gives a polynomial time complexity, it is inefficient when the number of users becomes large. To further speed up the computation, we present two novel optimization techniques in the next subsection.

5.2 Optimizations

—**Parallel Computation:** The key component in the greedy algorithm is to measure the marginal increment of the influence after adding node u, i.e., $\Delta(u) = I(S \cup \{u\}) - I(S)$, which can be derived as follows.

$$\Delta(u) = \left[1 - \sum_{h=1}^{\infty} c^h P(u, S, h)\right] \times \left[1 + \sum_{j \in (V - S \cup \{u\})} \sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h)\right].$$

In the baseline greedy algorithm, $\Delta(u)$'s are computed sequentially, which as a result incurs a large time overhead. Our main idea to speed up the computation is to estimate the marginal increment of all nodes, i.e., $\Delta(u)$ for every u, *in parallel*. Specifically, when performing R random walks from a particular node j, we measure the contribution of j to the marginal increment of every node. In other words, we obtain $P^S(j, u, h)$ for every u by using only the R random walks starting from j. As a result, we need only $O(nR)$ random walks to derive the marginal increment of all nodes, i.e., $\Delta(u)$ for every u, instead of $O(n^2R)$ random walks as in the baseline greedy algorithm.

—**Walk Reuse:** The core idea is that in each iteration of choosing one node to maximize the marginal increment, we record the total $O(nR)$ random walks in memory, and apply the updates accordingly after one node is added into the result set. By doing this, we can reuse the $O(nR)$ random walks to derive the marginal increment in the next iteration instead of starting new random walks from each node again.

By incorporating the above optimization techniques, we can reduce the time complexity to $O(nRL)$, where L denotes the maximum walk length. In other words, we can use the L leading terms to estimate $\sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h)$ and $\sum_{h=1}^{\infty} c^h P(u, S, h)$ as described in Section 4. Thus, we let each walk runs for L steps at most. Algorithm 3 states the procedure. We use $score[u]$ and $P[u]$ to record $\sum_{j \in V - S \cup \{u\}} \sum_{h=1}^{\infty} c^h P^S(j, \{u\}, h)$ and $\sum_{h=1}^{\infty} c^h P(u, S, h)$ for computing $\Delta(u)$, respectively. Algorithm 3 runs in two phases. The first phase (lines 1–13) is to select the first seed node by running random walks and also record all the walking information for reuse. The second phase (lines 14–18) is to select the remaining $k - 1$ nodes based on the stored information which requires to be updated after selecting each node. We give the update function in Algorithm 4.

The update function is to update the walk information stored in *score* and P. Every time after we selecting a node v, the random walk in the following iterations should stop when it encounters v, and the values stored in *score* and P should change accordingly. To achieve this, for each random walk that hits v (line 2), we first check if it has visited any node in S (lines 4–7). If not, we increase $P[w.j]$ after adding v in S (lines 8 and 9). Since the following walks should stop when hitting v, we update $score[u]$ if node u is visited after v (lines 10–12).

ALGORITHM 3: Optimized Greedy Algorithm

```
Inputs: A hypergraph and a parameter $k$;
Output: A set $S$ of $k$ nodes for maximizing $I(S)$;
1: $S \leftarrow \emptyset$, $score[1...n] \leftarrow 0$, $P[1...n] \leftarrow 0$;
2: for $j \in V$ do
3:     for $r = 1$ to $R$ do
4:         $i \leftarrow j$, $visited \leftarrow \emptyset$;
5:         for $t = 1$ to $L$ do
6:             $visited \leftarrow visited \cup \{i\}$;
7:             $i \leftarrow$ Select a user according to the transition prob.;
8:             $RW[j][r][t] \leftarrow i$;
9:             if $i \notin visited$ then
10:                $index[i].add(item(j, r, t))$;
11:                $score[i] \leftarrow score[i] + \frac{c^t}{R}$;
12: $v \leftarrow \arg\max_{u \in V} score[u]$;
13: for $s = 2$ to $k$ do
14:     Update $(RW, index, P, score, S, v, L)$, $S \leftarrow S \cup \{v\}$;
15:     $v \leftarrow \arg\max_{u \in (V-S)} (1 - P[u])(1 + score[u])$;
16: $S \leftarrow S \cup \{v\}$;
```

ALGORITHM 4: Update Function

```
1: function UPDATE $(RW, index, P, score, S, v, L)$
2:     for $w \in index[v]$ do
3:         $k \leftarrow L$;
4:         for $t = 1$ to L do
5:             if $RW[w.j][w.r][t] \in S$ then
6:                 $k \leftarrow t$;
7:                 break;
8:         if $k == L$ then
9:             $P[w.j] \leftarrow P[w.j] + c^t/R$
10:        for $i = w.t + 1$ to $k$ do
11:            $u \leftarrow RW[w.j][w.r][i]$, $score[u] \leftarrow score[u] - c^t/R$;
12: end function
```

5.3 Optimizations On Weighted Graphs

Considering that weighted graph also have a wide use in OSNs, for example, we often have preference on some special neighbors in real-world OSNs, we also optimize our solution to support addressing the (**IMP(SAN)**) problem in weighted networks.

We use the random walk-based approach to measure and maximize the influence diffusion over a weighted network. Recall that in unweighted graphs, the walker moves to each neighbor of the current node with equal probability. While in weighted graphs, walker moves to each neighbor of the current node with a probability proportional to the weights of the edge. Formally, suppose the current node is i, the walker moves to neighbor j with probability $p_{ij} = \frac{C_{ij}}{\sum_{k \in N(i)} C_{ik}}$, where C_{ij} denotes the weight of edge e_{ij}. To compute the transition probability p_{ij}, one needs to search all neighbors of the current node to get their weights.

We use a typical approach [40] to simulate the random walk. We summarize its key idea as follows:

- **Step one:** Generate p uniformly at random from $[0, 1]$.
- **Step two:** Select j uniformly at random from the neighbors of the current node i denoted by $N(i)$. The walker moves to node j if $p_{ij} >= p$, otherwise, delete node j from $N(i)$, and set $p = p - p_{ij}$.

The above two steps are repeated until a node is selected, to which the walker moves.

Recall that, the optimized greedy algorithm (i.e., Algorithm 3) has a time complexity of $O(nRL)$ in addressing the **(IMP(SAN))** problem in unweighted network. Applying Algorithm 3 to the weighted graph, the time complexity is $O(ndRL)$, where d denote the average degree of the graph. Generally $d \approx O(lg(n))$ in a common social networks. To further reduce the time complexity, we optimize simulation of the random walk over weighted graphs using binary search. Specifically, we first construct an incremental array A_i for each node i, representing the cumulative transition probabilities of its neighbors. For example, if node i has three neighbors u, v, w, with transition probabilities p_{iu}, p_{iv}, p_{iw}, then $A_i = \{0, p_{iu}, (p_{iu} + p_{iv}), (p_{iu} + p_{iv} + p_{iw})\}$. Then we optimize the simulation of the random walk as follows:

- **Step one:** Generate p uniformly at random form $[0, 1]$.
- **Step two:** Using binary search to find the location of p in A_i.

The total time complexity of the above two steps is $O(lg(d))$. Thus, we reduce the total time complexity of our optimized greedy algorithm in weighted graphs to $O(nlg(d)RL)$.

6 EXPERIMENTS

To show the efficiency and effectiveness of our approach, we conduct experiments on real-world datasets. In particular, we first show that incorporating online activities in seed selection can lead to a significant improvement on the influence spread, i.e., influence more users with the same seed size. Then we show that our IM-RW algorithm takes much less running time than the most popular influence maximization algorithm, while achieves almost the same influence spread. Lastly, we show the generality of our IM-RW algorithm, i.e., it applies to bot LT and IC influence diffusion model, as well as multiple types of users and multiple types of online activities. All experiments are conducted in both unweighted graphs and weighted graphs.

6.1 Datasets

We consider six datasets from social rating systems: Ciao [39], Epinions [1], Slashdot [2], Yelp [27] , Youtube [2], and Flixster [26]. Such social rating networks are composed of a social network, where the links can be interpreted as either friendships (undirected link) or a following relationship (directed link), and a rating network, where a link represents that a user assigns a rating (or writes a review) to a product. Assigning a rating corresponds to an online activity, and multiple users assigning ratings to the same product means that they participate in the same online activity. In the rating network, we remove rating edges if the associated rating is less than 3 so as to filter out the users who dislike a product. Through this we guarantee that all the remaining users who give ratings to the same product have similar interests, e.g., they all like the product. Since the original Flixster dataset is too large to run the most popular influence maximization algorithms, we extract only a subset of the Flixster dataset for comparison studies. In particular, since the OSN of Flixster is almost a connected component, we randomly select a user, and run the breadth-first search algorithm until we get 300,000 users. In the Youtube dataset, a very small fraction (i.e., 2%) of users participates in activities. This small fraction makes it inadequate to validate the impact

Table 2. Datasets Statistics

Dataset	Users	Links in OSN	Products	Ratings	OSN Type
Ciao	2,342	57,544	15,783	32,783	directed
Epinions	18,089	355,813	241,576	662,870	directed
Slashdot	82,168	870,161	35,065	3,357,879	directed
Yelp	174,100	2,576,179	56,951	958,415	directed
Flixtser	300,000	6,394,798	28,262	2,195,134	undirected
Youtube	199,957	1,214,943	29,677	292,497	undirected

of user activities. Thus, we randomly delete some users who do not participate in any activities to increase the fraction of users participating in activities to around 20%. We state the statistics of the six datasets in Table 2.

All the above datasets are unweighted graphs. To enable experiments on weighted graphs, we transform them to weighted ones. Two users have a closer friendship if there are more common neighbors between two them. We therefore set the weight of the edge between user i and j, say C_{ij}, to be proportional to the number of common neighbors of user i and j. Let S_{ij} denote the number of common neighbors of user i and j. Then, we set $C_{ij} = S_{ij} + k$. As S_{ij} may equals 0, we set $k > 0$ to guarantee C_{ij} being positive for all $e_{ij} \in E$. For simplicity, we let $k = 1$, so we have $C_{ij} = S_{ij} + 1$. All algorithms are run on a server with two Intel Xeon E5-2650 2.60 GHz CPU and 64 GB memory.

6.2 The Benefit of Incorporating Activities

We first show that incorporating online activities in seed selection can lead to a significant improvement on the influence spread. We fix the seed size k as 50. To show the impact of activities, we use the most popular influence maximization algorithm IMM [40] to select the seed set on OSNs and use our IM-RW algorithm to select the seed set on SANs which take online activities into account. Then we use simulations to estimate the expected influence spread of the selected k users on SANs and denote the results as $\sigma(\text{OSN})$ and $\sigma(\text{SAN})$, respectively. Finally, we define the improvement ratio on the expected influence spread as $[\sigma(\text{SAN}) - \sigma(\text{OSN})]/\sigma(\text{OSN})$.

To present the key insights, we consider the simple case in which there is only one type of users and online activities. Namely, all users have a same value of α which indicates the weight of activities. We emphasize that our model also works in the general case of multiple types of users and online activities, which are shown in Section 6.4. By default, we consider the IC model of influence diffusion. We also consider the LT model of influence diffusion in a group of experiments in Section 6.4.

We show the improvement of incorporating online activities by varying the weight of activities α from 0 to 1. Figures 2 and 3 respectively show the results under unweighted and weighted graphs. The horizontal axis shows the value of α, and the vertical axis presents the corresponding improvement ratio. From Figures 2 and 3, one can observe that the improvement ratio is 0 when $\alpha = 0$. This is because users are not affected by other users through online activities when $\alpha = 0$. As α increases, the improvement ratio also increases. This shows that as users are more prone to be affected by other users through online activities, incorporating online activities bring larger benefit. When $\alpha = 0.5$, the improvement ratio is around 25% for unweighted Ciao graph. That is, we can influence 25% more users when incorporating online activities in the seed selection. Similar conclusions can also be observed for the other graphs, in both unweighted and weighted settings. It is interesting to observe that as α approaches to one, the improvement ratio reaches up to 16 for unweighted Flixster graph, and even 36 for weighted Flixster graph. which implies a more than an

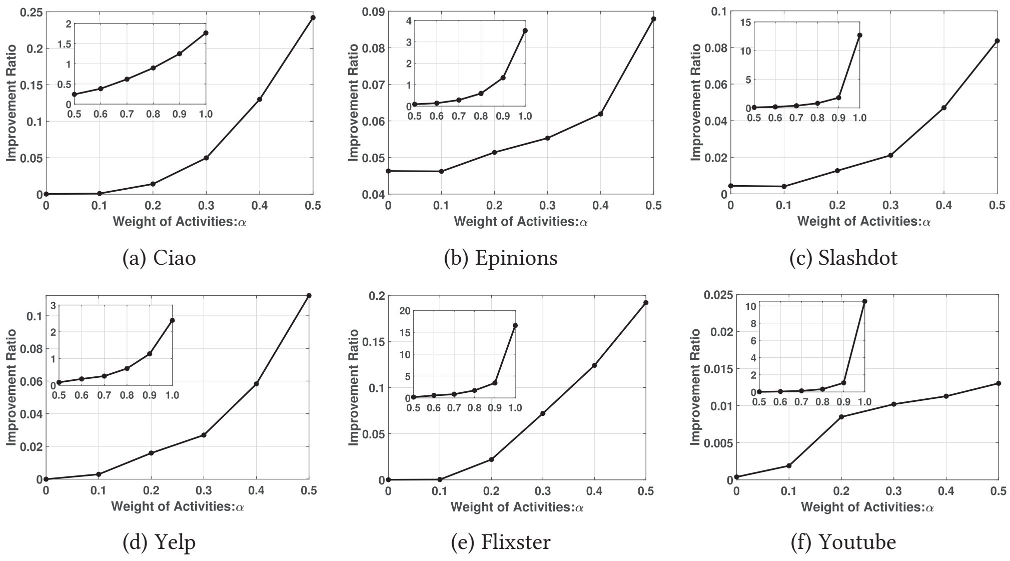


Fig. 2. Impact of online activities on influence spread (unweighted graph).

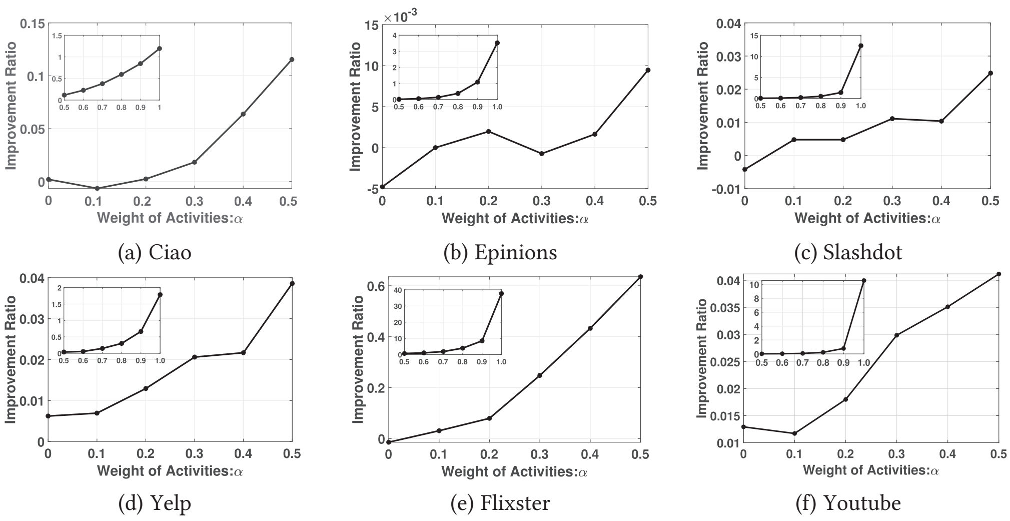


Fig. 3. Impact of online activities on influence spread (weighted graph).

order of magnitude improvement. In summary, incorporating online activities in the seed selection by using IM-RW significantly improves the selection accuracy.

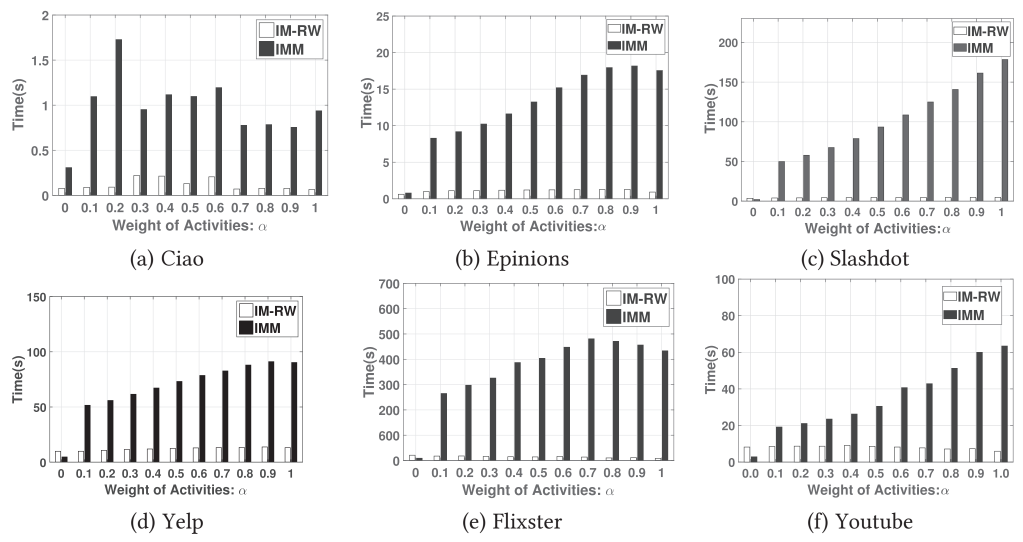
6.3 Performance Evaluation of IM-RW

In this subsection, we validate the efficiency and effectiveness of IM-RW by comparing it with IMM, which is the most used algorithm for solving influence maximization problem in OSNs, from two aspects, the running time and the influence spread.

Note that IMM was originally developed for OSNs, which did not take into account user activities. For fair comparison, we enable IMM to process SANs by transferring the SAN to a

Table 3. Preprocess Time

Pre-time dataset / α	Ciao	epinions	Youtube
0.0	0.613	15.5	97.6
0.1	0.988	75.5	679.9
0.2	0.587	70.6	648.0
0.3	1.081	76.2	675.6
0.4	0.964	78.1	706.6
0.5	0.583	71.3	655.1
0.6	0.586	74.9	719.0
0.7	0.583	70.6	689.6
0.8	0.584	68.6	689.4
0.9	0.584	70.0	710.2
1.0	0.644	68.1	627.2



(a) Ciao (b) Epinions (c) Slashdot

(d) Yelp (e) Flixster (f) Youtube

Fig. 4. Running time of IM-RW and IMM with different activity weights (unweighted graph).

weighted OSN, where we convert the online activities into edge weights. We call this converting process as *preprocess*. Table 3 shows the time cost of converting activities into edge weights for Ciao, Epinions, and Youtube datasets under different settings of α. We find that it needs more than 10 minutes to convert the Youtube dataset, which is around 10× of the time cost of IMM in finding the seed set. For fair comparison, we ignore the cost of this converting process for IMM in the following results.

We first compare the running time of IM-RW and IMM by varying the weight of activities α and the seed size k under unweighted graph, and the results are presented in Figures 4 and 5. Specifically, Figure 4 shows that IMM takes much longer time than IM-RW, especially when the network is large and online activities become more important (i.e., with larger α). This is because as α increases, the time cost of IMM depends more on user–activity–user links than user–user links. Thus, as the amount of user–activity–user links is much more than that of user–user links

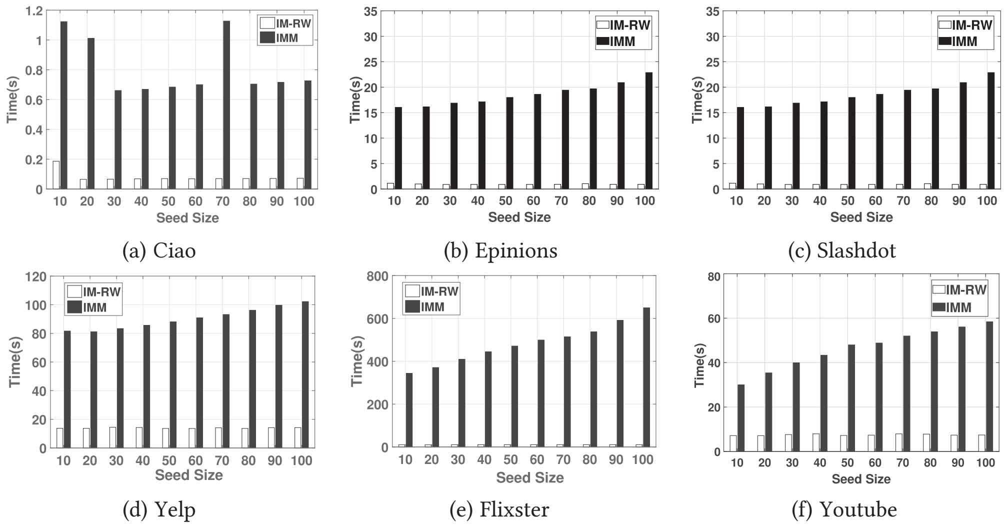


Fig. 5. Running time of IM-RW and IMM with different seed sizes (unweighted graph).

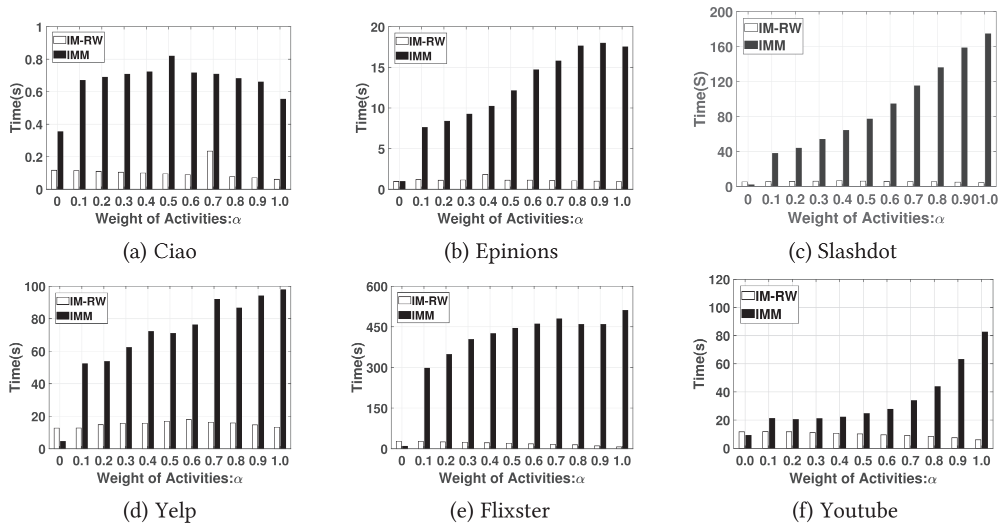
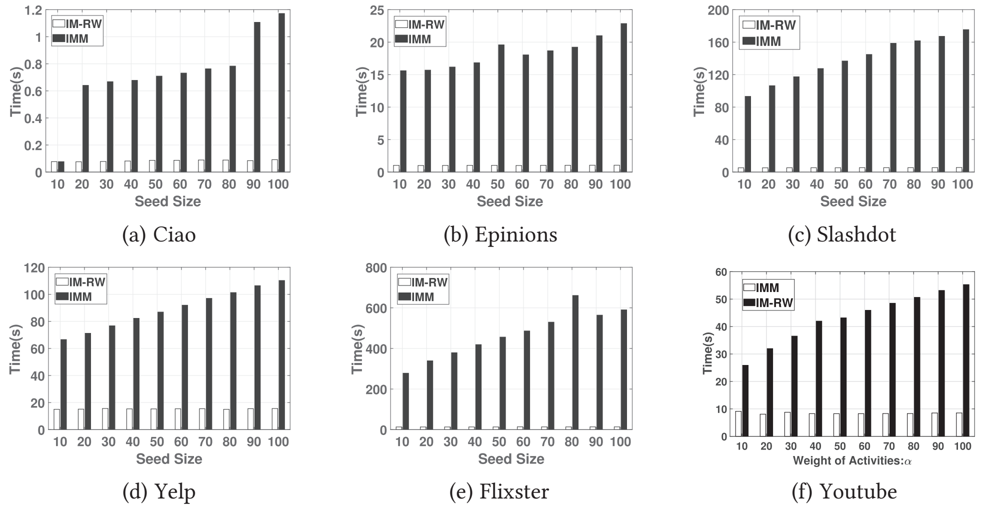


Fig. 6. Running time of IM-RW and IMM with different activity weights (weighted graph).

in SANs, the time cost of IMM will increase. On the other hand, when we fix α as 0.8 and vary the seed size k, Figure 5 also shows that IMM takes much longer time than IM-RW under all settings in unweighted graphs. Figures 6 and 7 show the results in weighted graphs with the same settings as above. Similar to unweighted graphs, IM-RW always takes much less time than IMM when incorporating online activities and growing the seed size. Therefore, we can conclude that our IM-RW algorithm really improves the efficiency of solving the influence maximization problem in SANs with online activities being considered.

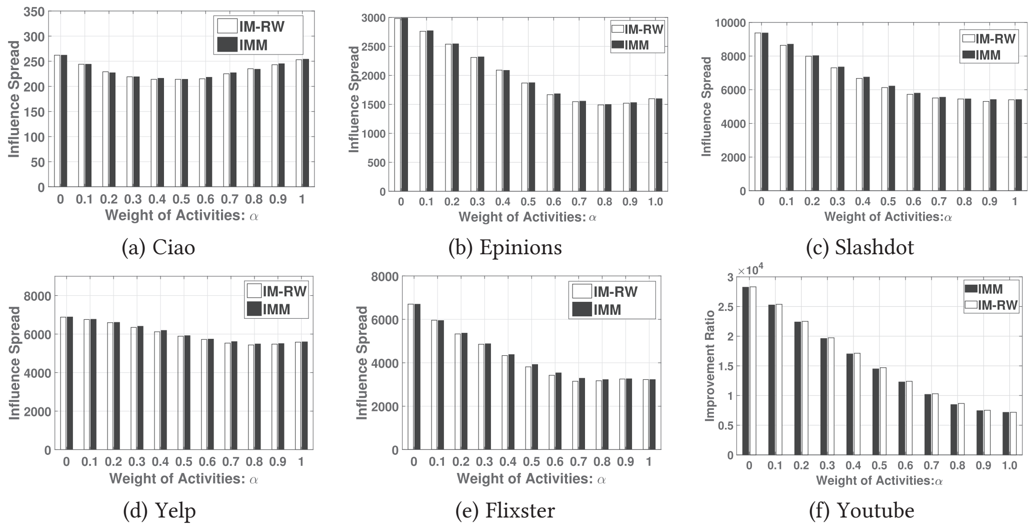
We further show the influence spread of the most influential users selected by the above two algorithms in Figure 8 for unweighted graphs and Figure 9 for weighted graphs. The x-axis shows the values of α, and the y-axis represents the corresponding influence spread. We see that by



(a) Ciao (b) Epinions (c) Slashdot

(d) Yelp (e) Flixster (f) Youtube

Fig. 7. Running time of IM-RW and IMM with different seed sizes (weighted graph).



(a) Ciao (b) Epinions (c) Slashdot

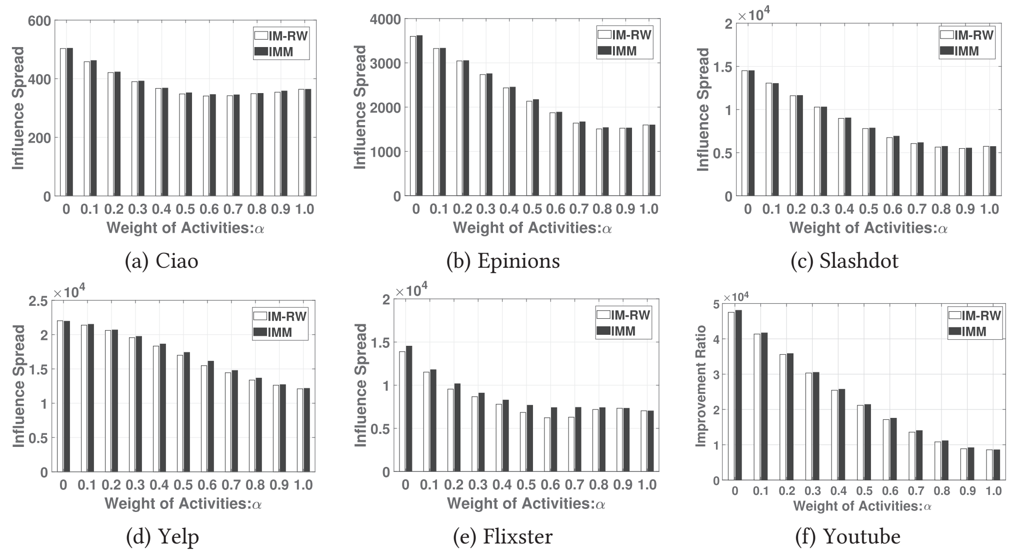
(d) Yelp (e) Flixster (f) Youtube

Fig. 8. Influence spread of IM-RW and IMM (unweighted graph).

taking online activities into consideration, both IMM and IM-RW can achieve almost the same performance. Because IMM is an influence maximization algorithm with theoretical performance guarantees, we can conclude that our IM-RW approach also has a good performance to maximize the influence spread. By combining with the running time experiments above, we like to point out that IM-RW can realize a similar influence spread, while it requires much less running time compared with IMM.

6.4 Model Generality

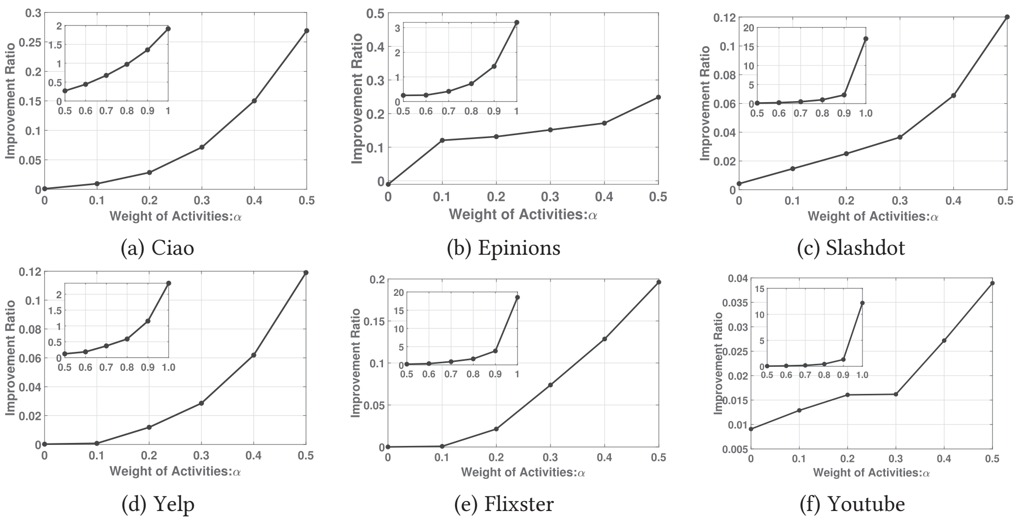
We show the generality of our algorithm (IM-RW) in two aspects: (1) it is applicable to different influence diffusion models, e.g., the LT model; and (2) it is applicable to heterogeneous system settings, e.g., multiple types of users and online activities.



(a) Ciao (b) Epinions (c) Slashdot

(d) Yelp (e) Flixster (f) Youtube

Fig. 9. Influence spread of IM-RW and IMM (weighted graph).

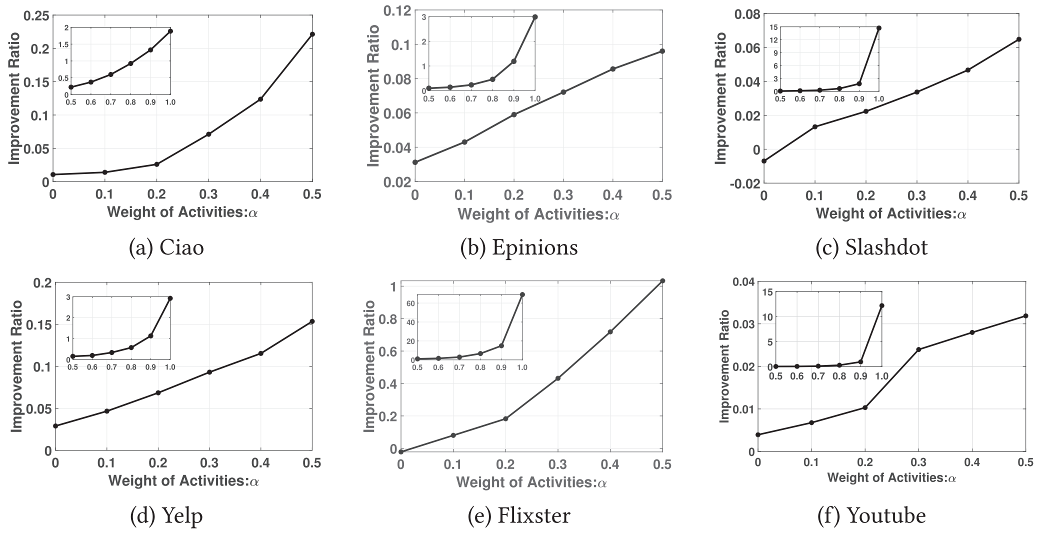


(a) Ciao (b) Epinions (c) Slashdot

(d) Yelp (e) Flixster (f) Youtube

Fig. 10. Improvement ratio under the LT model (unweighted graph).

6.4.1 Results under LT Model. We now show that in addition to the IC model, our IM-RW algorithm also applies to the LT influence diffusion model. Similar to Section 6.2, we still consider one type of users and one type of online activities under both unweighed and weighted graphs by default. We show the improvement ratio (in terms of the expected influence spread) of the IM-RW algorithm over the IMM algorithm in Figures 10 and 11. We find the similar results as that under IC model. Under LT influence diffusion model, we also have high improvement when incorporate the users' online activities in both unweighed and weighted graphs. This implies that our IM-RW algorithm maintains the benefits of incorporating online activities under different influence diffusion settings.



(a) Ciao (b) Epinions (c) Slashdot
(d) Yelp (e) Flixster (f) Youtube

Fig. 11. Improvement ratio under the LT model (weighted graph).

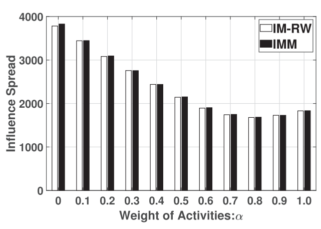
Table 4. Running Time of IM-RW Under the LT Model (Unweighted Graph)

Dataset	Running time (s)										
	$\alpha = 0$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ciao	0.105	0.114	0.100	0.103	0.115	0.097	0.102	0.118	0.117	0.100	0.104
Epinions	0.618	0.710	1.085	1.049	1.174	1.190	0.881	0.902	0.932	0.925	1.256
slashdot	3.587	3.975	4.188	4.253	4.370	4.619	4.662	4.765	4.770	4.734	4.387
Yelp	8.396	9.838	10.655	11.267	11.940	12.321	12.828	13.362	13.382	13.639	14.147
Flixster	18.169	18.233	17.448	21.350	15.352	14.618	12.965	11.833	10.717	8.575	6.121
Youtube	8.19605	8.60072	8.65797	8.33353	8.494	8.55115	8.124	7.73262	7.30208	6.88423	5.97629

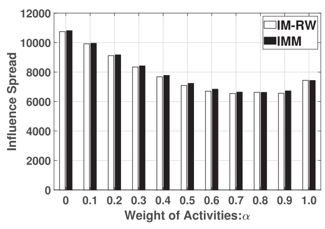
Table 5. Running Time of IM-RW Under the LT Model (Weighted Graph)

Dataset	Running time (s)										
	$\alpha = 0$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ciao	0.118	0.115	0.109	0.105	0.100	0.095	0.089	0.083	0.077	0.070	0.061
Epinions	0.981	1.056	1.076	1.127	1.256	1.157	1.074	1.106	1.034	1.085	0.960
Slashdot	5.324	5.598	5.674	5.721	5.728	5.701	5.579	5.514	5.282	5.060	4.471
Yelp	12.750	14.051	14.837	15.214	15.695	15.643	15.884	15.502	15.403	14.952	13.571
Flixster	28.552	26.474	25.022	26.848	21.304	19.698	17.211	15.226	12.785	9.995	6.114
Youtube	11.6643	11.7657	11.4863	11.059	10.6567	10.0913	9.60594	9.08515	8.3635	7.47677	5.94343

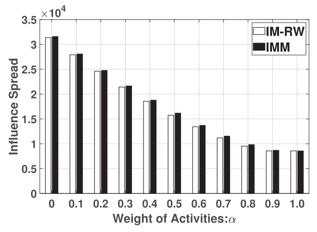
Tables 4 and 5 show the time cost of IM-RW under LT model in both unweighted graphs and weighed graphs, respectively. We can observe that IM-RW cost at most 0.12 second for the Ciao dataset, and at most 1.3 seconds, 5.8 seconds, 15.9 seconds, 28.6 seconds, and 8.7 seconds respectively for the Epinions, Slashdot, Yelp, Flixster, and Youtube dataset. This implies that IM-RW is still highly efficient under the LT model.



(a) Epinions

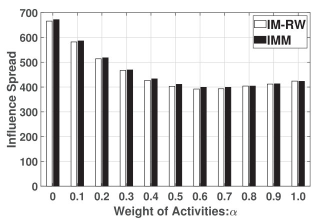


(b) Slashdot

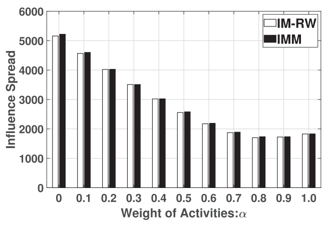


(c) Youtube

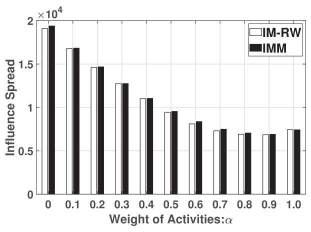
Fig. 12. Influence spread of IM-RW and IMM under LT model (unweighted graph).



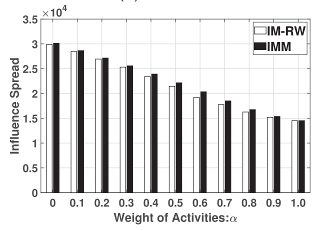
(a) Ciao



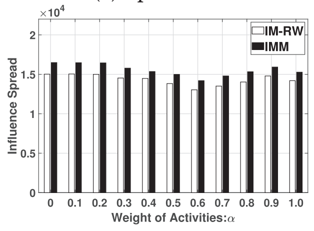
(b) Epinions



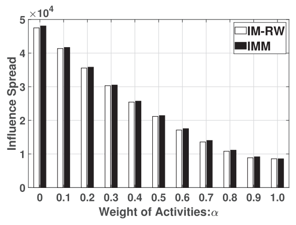
(c) Slashdot



(d) Yelp



(e) Flixster



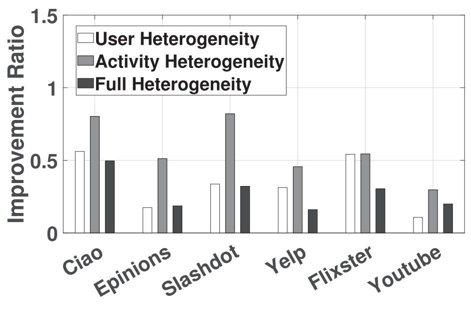
(f) Youtube

Fig. 13. Influence spread of IM-RW and IMM under LT model (weighted graph).

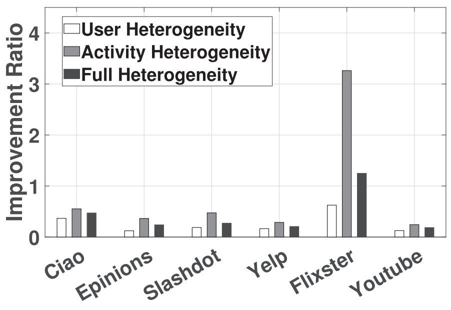
We also show influence spread under the LT model by using IM-RW algorithm and IMM algorithm for both unweighted graphs and weighed graphs. The results are shown in Figures 12 and 13, respectively. We can see that IM-RW can influence almost the same number of nodes comparing with IMM. Here we like to emphasize that both IMM and IM-RW take online activities into consideration in this experiment, and IMM is an influence maximization algorithm with theoretical performance guarantee, so these results indicate that our IM-RW approach also has a good performance to maximize the influence spread under the LT model. However, even though IMM and IM-RW have similar performance in terms of influence spread, IM-RW incurs much less time cost compared with IMM as studied above.

6.4.2 Multiple User Types and Activity Types. To study the generality of our IM-RW algorithm, we consider three heterogeneous settings: (1) user heterogeneity: two types of users and one type of online activities; (2) activity heterogeneity: one type of users and two types of online activities; and (3) full heterogeneity: two types of users and online activities.

Note that in the datasets there is no information on user types and activity types, so we synthesize user and activity types by following the 80–20 rule [28], which is widely adopted in the fields of economics and computer science. In particular, in the user heterogeneity case, we randomly divide users into two types where the first type accounts for 80% of population. Considering that users are usually easy to be influenced by online activities in real life, we set the



(a) Unweighted graph



(b) Weighted graph

Fig. 14. Improvement ratio under heterogeneous settings.

Table 6. Running Time of IM-RW Under Heterogeneous Settings (Unweighted Graph)

Dataset	Running time (s)		
	User heterogeneity	Activity heterogeneity	Full heterogeneity
Ciao	0.078	0.059	0.060
Epinions	0.900	0.938	0.972
Slashdot	4.625	4.646	4.654
Yelp	13.254	8.615	9.018
Flixster	11.870	8.993	10.180
Youtube	9.342	7.7652	9.173

Table 7. Running Time of IM-RW Under Heterogeneous Settings (Weighted Graph)

Dataset	Running time (s)		
	User heterogeneity	Activity heterogeneity	Full heterogeneity
Ciao	0.082	0.218	0.141
Epinions	1.088	1.087	1.157
Slashdot	5.506	5.344	5.663
Yelp	15.064	13.996	14.741
Flixster	17.265	12.523	14.938
Youtube	12.5123	9.534	10.92348

weight of activities α for the first type as 0.8, and set it as 0.2 for the second type of users. In the activity heterogeneity case, we also divide online activities into two types by following the 80–20 rule and fix the total weight of activities as 0.8. Precisely, the weight of the first type of online activities is set as $0.8 \times 0.2 = 0.16$ and that of the second type is set as $0.8 \times 0.8 = 0.64$. Similarly, in the full heterogeneity case, we divide both users and online activities into two types by using the parameters in the first two cases.

Figure 14(a) and (b) shows the improvement ratio of IM-RW over IMM in terms of the expected influence spread. We observe that the improvement ratio is at least 10% in all cases, as high as 300% for the Flixster dataset. This shows that our IM-RW algorithm significantly outperforms the IMM algorithm by incorporating online activities even under the heterogeneous setting with both weighted and unweighted graphs. Tables 6 and 7 show the running time. We can observe that

IM-RW only takes not more than 20 seconds. This implies that IM-RW is also efficient even under heterogeneous users and activities.

Summary: Our IM-RW algorithm achieves a good performance in both the running time and the influence spread by taking online activities into account in SANs. In particular, comparing to a variant of the most popular algorithm IMM (adapted to social activities by converting activities into weights of edges), our IM-RW algorithm achieves almost the same performance in seed selection, while it only requires much less running time and greatly improves the computation efficiency.

On the other hand, our IM-RW algorithm is applicable for general situations, i.e., it is flexible to different influence diffusion models like IC model and LT model. It is also scalable to heterogeneous system setting of multiple types of users or online activities.

7 RELATED WORK

Influence maximization problem in OSNs was first formulated by Kempe et al. [29], and in this seminal work, the authors proposed the IC model and the LT model. Since then, this problem receives a lot of interests in academia in the past decade [9–11]. Because of the NP-hardness under both the IC model [9] and the LT model [11], many of the previous studies focus on how to reduce the time complexity. Recently, Borgs et al. [7] developed an algorithm which maintains the performance guarantee while reduces the time complexity significantly, and Tang et al. [40, 41] further improved the method and proposed the IMM algorithm. Then , two typical variants of the IMM algorithms were proposed, i.e., Nguyen et al. [35] developed a **stop-and-stare strategy (SSA)** and Wang et al. [46] developed a novel **bottom-k sketch-based RIS framework (BKRIS)** to further speed up the IMM algorithm. However, SSA and BKRIS still considered only the OSNs. Namely, they still need a great cost in *preprocess* to transfer SANs to weighted OSNs. Besides reducing the computation overhead, a number of works proposed several new influence models, e.g., topic-aware influence model [5], competitive influence model [34], and opinion-based influence model [15]. Also, variants of the conventional influence maximization problems are also studied in recent years, and the detailed survey of influence maximization can be found in [33]. In particular, Sun et al. [38] proposed a multi-round influence maximization problem, which allows influence to propagate in multiple rounds independently from different seed sets so as to select seeds in each round to maximize the expected number of nodes that are activated at least in one round. Zhu et al. [52] studied the social influence maximization problem in hypergraph by modeling crowd influence as a hyperedge.

A number of works studied different centrality measures, which can also be used for top-k node recommendation. For example, degree centrality [12, 14, 50] indicates that the importance of a node is proportional to its degree, so it is reasonable to recommend the top-k nodes with the highest degree. Closeness centrality [10, 12] based approach chooses to recommend the top-k nodes with the smallest average distance to other nodes. Commonly used centrality measures also include betweenness centrality [12, 50], which counts the number of times that a node was visited in the shortest path between any two nodes, and PageRank centrality [36], which allows the importance of nodes to spread through edges.

We would like to emphasize that our work differs from existing studies which address the traditional influence maximization problem or use classic centrality measures. We defined a novel influence centrality to measure the influence of each node, and also use a random walk-based Monte Carlo framework to estimate the influence centrality. More importantly, we take online activities into consideration. When we consider these online activities, only considering user-user links alone may not trigger the largest influence spread. Although we can also transform the user-activity-user links to user-user links, the underlying graph may become extremely dense so that traditional methods may not be efficient.

Random walk is a powerful algorithmic tool to analyze large-scale graphs, such as random walk sampling [44, 45, 51], Personalized PageRank [13, 31, 36], and SimRank [30]. A number of variants of random walk are also developed, such as random walk with restart [43], common neighbor aware random walk [49], FolkRank [24], and TrustWalker [25]. There are also some works focusing on accelerating the computation of random walks from system design perspective, such as DrunkardMob [31], KnightKing [47], and GraphWalker [37]. Our work is orthogonal to these studies, and it targets for node recommendation by addressing the influence maximization problem via random walks, while accelerating the random walk process with the above optimizations can also help improve the efficiency of our random walk-based approach.

8 CONCLUSIONS

In this article, we take online activities into consideration to formulate the influence maximization problem for SANs, and address the IMP(SAN) with a random walk approach. Specifically, we propose a general framework to measure the influence of nodes in SANs via random walks on hypergraphs, and develop a greedy-based algorithm with two novel optimization techniques to find the top k most influential nodes in SANs by using random walks. Experiments with real-world datasets show that our approach greatly improves the computation efficiency, while keeps almost the same performance in seed selection accuracy compared with the most popular influence maximization algorithm.

REFERENCES

[1] [n.d.]. http://www.public.asu.edu/ jtang20/datasetcode/truststudy.htm.
[2] [n.d.]. SNAP. Retrieved from https://snap.standford.edu/data/index.html.
[3] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 420–429.
[4] Suman Banerjee, Mamata Jenamani, and Dilip Kumar Pratihar. 2020. A survey on influence maximization in a social network. *Knowledge and Information Systems* 62, 9 (2020), 3417–3455.
[5] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. 2012. Topic-aware social influence propagation models. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining.*
[6] Song Bian, Qintian Guo, Sibo Wang, and Jeffrey Xu Yu. 2020. Efficient algorithms for budgeted influence maximization on massive social networks. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1498–1510.
[7] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms.*
[8] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. 2011. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th International Conference on World Wide Web*. 665–674.
[9] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*
[10] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *Proceedings of the 2009 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*
[11] Wei Chen, Yifei Yuan, and Li Zhang. 2010. Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of the 2010 IEEE International Conference on Data Mining.*
[12] Martin G. Everett and Stephen P. Borgatti. 1999. The centrality of groups and classes. *The Journal of Mathematical Sociology* 23, 3 (1999), 181–201.
[13] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.
[14] Linton C. Freeman. 1979. Centrality in social networks conceptual clarification. *Social Networks* 1, 3 (1979), 215–239.
[15] Sainyam Galhotra, Akhil Arora, and Shourya Roy. 2016. Holistic influence maximization: Combining scalability and efficiency with opinion-aware models. In *Proceedings of the 2016 International Conference on Management of Data.*
[16] Jianxiong Guo and Weili Wu. 2019. A novel scene of viral marketing for complementary products. *IEEE Transactions on Computational Social Systems* 6, 4 (2019), 797–808.

[17] Jianxiong Guo and Weili Wu. 2020. Influence maximization: Seeding based on community structure. *ACM Transactions on Knowledge Discovery from Data* 14, 6 (2020), 1–22.

[18] Kai Han, Keke Huang, Xiaokui Xiao, Jing Tang, Aixin Sun, and Xueyan Tang. 2018. Efficient algorithms for adaptive influence maximization. *Proceedings of the VLDB Endowment* 11, 9 (2018), 1029–1040.

[19] F. Hao and D.-S. Park. 2018. cSketch: A novel framework for capturing cliques from big graph. *The Journal of Supercomputing* 74, 3 (2018), 1202–1214.

[20] F. Hao, D.-S. Park, and Z. Pei. 2017. Exploiting the formation of maximal cliques in social networks. *Symmetry* 9, 7 (2017), 100.

[21] F. Hao, D.-S. Park, Z. Pei, H. Lee, and Y.-S. Jeong. 2016. Identifying the social-balanced densest subgraph from signed social networks. *The Journal of Supercomputing* 72, 7 (2016), 2782–2795.

[22] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. 2012. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 463–474.

[23] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of The American Statistical Association* 58, 301 (1963), 13–30.

[24] Andreas Hotho, Robert Jäschke, Christoph Schmitz, Gerd Stumme, and Klaus-Dieter Althoff. 2006. Folkrank: A ranking algorithm for folksonomies. In *Proceedings of the 2006 LWA*.

[25] Mohsen Jamali and Martin Ester. 2009. Trustwalker: A random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[26] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the 4h ACM Conference on Recommender Systems*.

[27] Stoppelman Jeremy and Simmons Russel. 2004. Yelp Dataset. Retrieved from https://www.yelp.com/dataset_challenge/dataset.

[28] Joseph M. Juran and James F. Riley. 1999. *The Quality Improvement Process*. McGraw Hill New York, NY.

[29] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 2003 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[30] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*.

[31] Aapo Kyrola. 2013. Drunkardmob: Billions of random walks on just a PC. In *Proceedings of the 7th ACM Conference on Recommender Systems*.

[32] Lakhotia, Kartik, and Kempe David. 2019. Approximation algorithms for coordinating ad campaigns on social networks.*In Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 339–48.

[33] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey.*IEEE Transactions on Knowledge and Data Engineering* 30, 10 (2018), 1852–1872.

[34] Yishi Lin and John C. S. Lui. 2015. Analyzing competitive influence maximization problems with partial information: An approximation algorithmic framework. *Performance Evaluation* 91 (2015), 187–204.

[35] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. 2016. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 695–710.

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report.

[37] Wang Rui, Yongkun Li, Hong Xie, Yinlong Xu, and John C. S. Lui. 2020. GraphWalker: An I/O-efficient and resource-friendly graph analytic system for fast and scalable random walks. In *Proceedings of the Annual Technical Conference*. USENIX.

[38] Lichao Sun, Weiran Huang, Philip S. Yu, and Wei Chen. 2018. Multi-round influence maximization.*In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2249–2258.

[39] Jiliang Tang, Huiji Gao, and Huan Liu. 2012. mTrust: Discerning multi-faceted trust in a connected world. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*.

[40] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*.

[41] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*.

[42] Guangmo Tong, Wu Weili, Guo Ling, Li Deying, Liu Cong, Liu Bin, and DingZhu Du. 2020. An efficient randomized algorithm for rumor blocking in online social networks.*IEEE Transactions on Network Science and Engineering* 7, 2 (2020), 845–54.

[43] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *Proceedings of the 2006 IEEE International Conference on Data Mining*. IEEE.

[44] Pinghui Wang, Junzhou Zhao, John C.S. Lui, Don Towsley, and Xiaohong Guan. 2013. Sampling node pairs over large graphs. In *Proceedings of the 2013 IEEE 29th International Conference on Data Engineering*. IEEE.

[45] Rui Wang, Min Lv, Zhiyong Wu, Yongkun Li, and Yinlong Xu. 2019. Fast graph centrality computation via sampling: A case study of influence maximisation over OSNs. *International Journal of High Performance Computing and Networking* 14, 1 (2019), 92–101.

[46] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Chen Chen. 2016. Bring order into the samples: A novel scalable method for influence maximization. *IEEE Transactions on Knowledge and Data Engineering* 29, 2 (2016), 243–256.

[47] Ke Yang, MingXing Zhang, Kang Chen, Xiaosong Ma, Yang Bai, and Yong Jiang. 2019. KnightKing: A fast distributed graph random walk engine. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM.

[48] Mao Ye, Xingjie Liu, and Wang-Chien Lee. 2012. Exploring social influence for recommendation: A generative model approach. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 671–680.

[49] Li Yongkun, Zhiyong Wu, Shuai Lin, Hong Xie, Min Lv, Yinlong Xu, and John C. S. Lui. 2019. Walking with perception: Efficient random walk sampling via common neighbor awareness. In *Proceedings of the IEEE 35th International Conference on Data Engineering*.

[50] Junzhou Zhao, John Lui, Don Towsley, and Xiaohong Guan. 2014. Measuring and maximizing group closeness centrality over disk-resident graphs. In *Proceedings of the 23rd International Conference on World Wide Web*.

[51] Junzhou Zhao, John Lui, Don Towsley, Pinghui Wang, and Xiaohong Guan. 2015. A tale of three graphs: Sampling design on hybrid social-affiliation networks. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering*.

[52] Jianming Zhu, Junlei Zhu, Smita Ghosh, Weili Wu, and Jing Yuan. 2018. Social influence maximization in hypergraph in social networks.*IEEE Transactions on Network Science and Engineering* 6, 4 (2018), 801–11.

Received September 2020; revised January 2021; accepted February 2021