Influence Maximization in Independent Cascade Networks Based on Activation Probability Computation

Wenjing Yang, Leonardo Brenner, Alessandro Giua

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2018.DOI

Influence Maximization in Independent Cascade Networks Based on Activation Probability Computation

WENJING YANG[1], LEONARDO BRENNER[1], AND ALESSANDRO GIUA[2], (Fellow, IEEE)
[1]Information and Systems Laboratory, Aix-Marseille University, Marseille 13397, France
[2]Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari 09124, Italy

Corresponding author: Wenjing Yang (e-mail: wenjing.yang@lis-lab.fr).

This work was supported in part by the China Scholarship Council (CSC), the project PHC CAI YUANPEI 2017 from Campus France under Project No. 38908TK, and the National Natural Science Foundation of China under Grant No. 61703321 and 61672400.

ABSTRACT Based on the concepts of *"word-of-mouth"* effect and *viral marketing*, the diffusion of an innovation may be triggered starting from a set of initial users. Estimating the influence spread is a preliminary step to determine a suitable or even optimal set of initial users to reach a given goal. In this paper, we focus on a stochastic model called the Independent Cascade model, and compare a few approaches to compute activation probabilities of nodes in a social network, i.e., the probability that a user adopts the innovation. First we propose the *Path Method* which computes the exact value of the activation probabilities but has high complexity. Second an approximated method, called *SSS-Noself*, is obtained by modification of the existing *SteadyStateSpread* algorithm, based on fixed-point computation, to achieve a better accuracy. Finally an efficient approach, also based on fixed-point computation, is proposed to compute the probability that a node is activated though a path of minimal length from the seed set. This algorithm, called *SSS-Bounded-Path* algorithm, can provide a lower-bound for the computation of activation probabilities. Furthermore, these proposed approaches are applied to the influence maximization problem combined with *SelectTopK* algorithm, *RankedReplace* algorithm and greedy algorithm.

INDEX TERMS Independent Cascade model, influence maximization, social networks

I. INTRODUCTION

In recent years, a large number of social network sites have appeared to connect people and groups together. Networks have been proved to be a good tool to obtain information and communicate ideas. Besides, they are becoming an effective marketing platform, through which it is possible to spread information or products to a large scale with a high speed.

Consider the following marketing example: a company designs a new app for online users and aims to market it through the social network. It can only choose a small number of users to try the app initially (because usually a company has limited budget and manpower on a product). Then the company encourages these users to recommend the app to their friends. And their friends would use it and recommend to their friends and so on. Whether users adopt an innovation is strongly influenced by their acquaintances. That is called the "*word-of-mouth*" effect [9] and this type of marketing is called the *viral marketing* [9] since it is similar to the spread of an epidemic.

The studies on the diffusion of innovations in social networks began in the middle of the 20th century [10], [12]. Motivated by the application of *viral marketing*, Domingos and Richardson [9] proposed a general framework for the application of data mining and modeled the social network as a Markov random field. Kempe et al. [2] proposed two diffusion models, namely the *Independent Cascade model* and the *Linear Threshold model*, to model the propagation of innovations. As a type of epidemic models, the Independent Cascade model assumes that an individual adopts an innovation with a certain probability if at least one of its in-neighbors has adopted it. Differently, the Linear Threshold model assumes that an individual adopts an innovation if a certain ratio of its in-neighbors have already adopted it. Besides, Kempe et al. [5] also formulated the issue of choosing influential sets of individuals as a discrete optimization problem. It aims to identify a small subset of initial adopters in a social network to maximize the influence propagation under a given diffusion model. They

also proved this *influence maximization* problem [24], [25], [28], [29] is NP-hard and gave a greedy approximation algorithm which guarantees, under certain conditions, that the influence spread approximates the optimal one within a factor of $(1 - 1/e)$, where e is the base of the natural logarithm. However, this approach requires long time to run the simulation, thus later much effort was devoted to derive more efficient algorithms. Leskovec et al. [13] proposed a "Cost-Effective Lazy Forward" (CELF) scheme to reduce the number of evaluations on the influence spread, nevertheless it still satisfies the approximation guarantee. To further improve the CELF heuristic, Chen et al. [11] presented MixedGreedyIC algorithm for the Independent Cascade model and MixedGreedyWC algorithm for the Weighted Cascade model. Goyal et al. [19] explored the CELF++ approach. Recently, Zhou et al. [4] derived an upper bound for the influence spread and further proposed the Upper Bound based Lazy Forward algorithm (UBLF). Borgs et al. [20] developed an elegant framework, named Reverse Influence Sampling (RIS), focusing on the reduction of running time. Tang et al. [21], [22] proposed the hop-based algorithm for both the Independent Cascade model and Linear Threshold model.

A preliminary step to determine a suitable or even optimal set of initial users when the goal is that of maximizing the influence spread is to compute the set of final adopters for any given set of initial users. When the Independent Cascade model is considered, the measure of the influence spread is given by the *activation probability* of an individual, i.e., the probability that the individual adopts the innovation. It has been proved that computing the exact influence spread under the Independent Cascade model is #P-hard by Chen et al. [3]. *Monte Carlo simulation* applied in many studies [2], [5], [13], [19] is basic and simple but quite time-consuming. Aggarwal et al. [6] gave a more efficient approximate algorithm called *SteadyStateSpread*. However, the computed solution may be far from the exact one, depending on the network structure, and there are no guaranteed bounds. Besides, they did not discuss the convergency of their iterative equation and the uniqueness of the final solution.

In order to improve the accuracy of activation probability computation, in our previous paper [23], we proposed a new *SSS-Noself* algorithm for the influence spread evaluation under the Independent Cascade model. Experimental results show that our *SSS-Noself* algorithm performs well in terms of value approximation. In this paper, we considerably extend our previous paper [23] under several aspects: First, we enhance the theoretical analysis by proving the convergence of the iteration function and the uniqueness of the final solution for *SteadyStateSpread*, as well as proving a fixed relationship among the influence spread values given by different approaches. Second, we apply our proposed algorithms for activation probability computation to the influence maximization problem. These approaches are exploited in combination with selection algorithms such as *SelectTop*K, *RankedReplace* and the traditional greedy algorithm. Third, we present a set of new experiments to show that our proposed algorithms perform well for approximately computing the influence spread and maximizing the final influence.

Focusing on the Independent Cascade model, we analyse different approaches for computing the activation probabilities as well as for selecting the seed set for influence maximization. Our main contributions can be summarised as following:

1) To compute the exact solution to the influence spread in small networks, we propose a method that explores all possible evolutions of a model: we call this approach *Path Method*. We point out that, due to its complexity, this method is only viable for small networks but it is useful to test the correctness of different approaches.
2) We discuss the convergence problem [15] and the multiple solutions problem [16] of *SteadyStateSpread*, proving that it converges to a unique solution using fixed-point theory [14]. Moreover, we point out two factors leading to the gap between the result of *SteadyStateSpread* and the exact solution: the dependent relation of individuals and the existence of circuits in network structures. To partially overcome the error caused by circuits, we further propose a new *SSS-Noself* algorithm which updates the activation probability of one node assuming that it has not been activated at all before.
3) We propose an efficient way to compute the activation probabilities along paths of bounded length and provide a lower-bound for the activation probabilities by *SSS-Bounded-Path*.
4) We combine the algorithms for activation probability computation with *SelectTop*K, *RankedReplace* and greedy algorithm to select seed set for solving the influence maximization problem.

The rest of the paper is organized as follows. Section 2 reviews the Independent Cascade model and formally defines the problems of activation probability computation and influence maximization. Section 3 proposes different approaches for both exact computation and approximate computation of activation probabilities. Section 4 evaluates the algorithms proposed in Section 3 for solving the influence maximization problem. Section 5 presents a series of experimental results. We conclude the paper in Section 6.

II. BACKGROUND

In this paper, we use the Independent Cascade model to describe the propagation of innovations through social networks. For convenience, the variables used extensively throughout the paper are listed in Table 1.

A. NETWORK STRUCTURE

As in most literatures, a social network is represented by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is a set of nodes representing individuals in the network: we use the terms

TABLE 1. Notation.

Symbol	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	a network with node set $\mathcal{V}$ and edge set $\mathcal{E}$
$\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$	an Independent Cascade model
$\mathcal{G}_{IC}^{[q]} = (\mathcal{V}, \mathcal{E}^{[q]}, p^{[q]})$	an Independent Cascade model without node q's influence
N	number of nodes in $\mathcal{G}$
K	number of seed nodes to be selected
$\mathcal{N}_j^{in}$	set of nodes with direct influence on node j
$\mathcal{N}_j^{out}$	set of nodes on which node j has direct influence
ϕ_0	seed set
$\lvert\phi_0\rvert$	number of nodes in the seed set
π_j	activation probability of node j
π_j^p	activation probability of node j computed by *Path Method*
π_j^s	activation probability of node j computed by *SteadyStateSpread*
π_j^n	activation probability of node j computed by *SSS-Noself*
$\pi_j^{[q]}$	activation probability of node j without node q's influence
π_j^{bp}	activation probability of node j computed by *SSS-Bounded-Path*
$\mathbb{N}$	set of non-negative integers
$\mathbb{N}^+$	set of positive integers

individual or *node* interchangeably. An edge $(i, j) \in \mathcal{E}$ denotes that node i influences node j directly [2].

To describe all individuals with direct influence on node j, we denote the *in-neighbors* of node j as $\mathcal{N}_j^{in} = \{i \in \mathcal{V} | (i, j) \in \mathcal{E}\}$. The *out-neighbors* of node j denoted as $\mathcal{N}_j^{out} = \{i \in \mathcal{V} | (j, i) \in \mathcal{E}\}$ represent the individuals on which node j has direct influence.

B. INDEPENDENT CASCADE MODEL

The Independent Cascade model [17] is a type of epidemic models, which is based on the assumption that a node may adopt an innovation when one of its in-neighbors has adopted the innovation. In the Independent Cascade model, every edge $(i, j) \in \mathcal{E}$ is associated with a *propagation probability* $p : (\mathcal{V} \times \mathcal{V}) \to (0, 1]$, where $p_{i,j}$ represents the probability that node j is influenced by node i through the edge (i, j) at step t when node i is activated at step $t - 1$. Thus, we denote an Independent Cascade model by a triple $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$.

In the Independent Cascade model, each node can be either *active* or *inactive*. When it adopts the innovation (is activated), it becomes active, otherwise is said to be inactive. We also assume that nodes can switch from being inactive to being active, but can not switch in the other direction. It means that the adoption of an innovation is permanent and for this reason the model is called *progressive* [1].

Let us define ϕ_0 as the *seed set*, i.e., the set of nodes which have adopted the innovation at step $t = 0$. Then the innovation propagates from the seed set step by step. The activation progress can be described as follows. At each step $t = 1, 2, ...,$ an inactive node j is activated with a propagation probability $p_{i,j}$ by its in-neighbor i which is activated at step $t - 1$. Note that every active node has only one chance to influence each of its out-neighbors. If there is more than one in-neighbor of inactive node j activated at step $t - 1$, the order in which they attempt to activate node j at step t does not affect the final probability of node j being activated. This activation process ends when no more nodes adopt the innovation. The final probability that nodes are activated during the propagation process is denoted as the measure of influence spread in this paper.

C. PROBLEM FORMULATION

We give a mathematical description of the two problems studied in this paper: activation probability computation and influence maximization.

Activation Probability Computation. We focus on the evaluation of influence spread under the Independent Cascade model given a seed set. In this paper, we denote the influence spread as the sum of activation probabilities of all nodes in a network.

Definition 1 (Activation Probability)**.** *Given an Independent Cascade model $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$ and a seed set ϕ_0, the probability that a node $j \in \mathcal{V}$ is activated during the innovation propagation process is defined as the activation probability of node j, denoted as π_j.*

The final influence spread $\sigma(\phi_0)$ is given by

$$\sigma(\phi_0) = \sum_{j \in \mathcal{V}} \pi_j.$$

Influence Maximization. This problem aims to maximize the influence spread through a social network, by targeting a subset of individuals to adopt an innovation initially. We formalize it under the Independent Cascade model as follows.

Problem 1. *Given an Independent Cascade model $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$ and a constant integer K, find a seed set $\phi_0 \subseteq \mathcal{V}$ of cardinality $|\phi_0| = K$, such that the final influence spread $\sigma(\phi_0)$ is maximized, i.e.,*

$$\phi_0 = \underset{\phi_0 \subseteq \mathcal{V}}{argmax} \{\sigma(\phi_0) || \phi_0| = K\}.$$

III. ACTIVATION PROBABILITY COMPUTATION

The preliminary step to determine a seed set for achieving a large influence spread is to evaluate the final innovation adoption, i.e., the activation probability of all nodes in a network. The methodology for both exact and approximate activation probability computation is given in this section.

A. EXACT COMPUTATION OF ACTIVATION PROBABILITIES

In this part, we propose an algorithm to compute the exact solution to the influence propagation called *Path Method*. The value of activation probability computed by *Path Method* for node $j \in \mathcal{V}$ is defined as π_j^p. Since *Path Method* can give an exact value, we have $\pi_j^p = \pi_j$.

Path Method takes into account all evolutions of a model, so that it can offer a precise result of the final influence propagation. Firstly, it creates an evolution graph for a model, which is composed of cells shown in Fig. 1. Each cell C_k consists of three elements: *past active nodes* is a set A_k^p which contains all the nodes activated before the current step; *current active nodes* is a set A_k^c which contains the nodes activated in the current step; *cell probability* P_k is the

probability that the evolution described by A_k^p and A_k^c occurs. A cell whose current active nodes is null is called a *terminal cell*. Every non-terminal cell $C_i = (A_i^p, A_i^c, P_i)$ is connected with each of its successor cells $C_k = (A_k^p, A_k^c, P_k)$ by a directed arc with which is associated the *arc probability* $P_a^{(i,k)}$ that an evolution that reaches cell C_i proceed to reach C_k. Adding together all the cell probabilities of terminal cells whose past active nodes contains node j, the exact activation probability of node j can be computed.

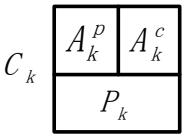


FIGURE 1. Cell of evolution graph.

TABLE 2. Symbols used in Algorithm 1.

Symbols	Descriptions
C_k	cell k
A_k^p	past active nodes of cell k
A_k^c	current active nodes of cell k
P_k	cell probability of cell k
$P_a^{(i,k)}$	arc probability of the arc $C_i \to C_k$
k	label of cell
i	label of no-terminal cell
n	number of cells in the evolution graph
new	set of cells to be explored
$\mathcal{S}$	inactive out-neighbors of the current active nodes of a cell
$\mathcal{S}'$	one subset of $\mathcal{S}$
$\mathcal{N}_{A_k^c}^{out}$	union of all out-neighbors of nodes in A_k^c

Algorithm 1 creates the evolution graph of an Independent Cascade model and computes the activation probabilities of nodes. The symbols used in Algorithm 1 are described in Table 2. The algorithm defines as initial cell $C_1 = (\emptyset, \phi_0, 1)$ because initially no node has been previously explored ($A_1^p = \emptyset$), the set of currently active nodes is the seed set ($A_1^c = \phi_0$) and the probability of reaching this condition during a run is 1 ($P_1 = 1$). C_1 is also added to the set new containing cells that need to be explored.

Then, while the set new is not empty a cell $i \in new$ is selected and its child cells are computed as follows. From a cell C_i the innovation can propagate to any subset of

$$\mathcal{S} = \mathcal{N}_{A_i^c}^{out} - (A_i^p \cup A_i^c),$$

which contains the out-neighbors of A_i^c that have not yet adopted the innovation.

Two cells are called *equivalent* if both their sets of past active nodes and sets of current active nodes are the same. For any subset $S_k \subseteq S$, a new cell $C_k = (A_k^p, A_k^c, P_k)$ is created when it is not equivalent with any other existing cell, with

Algorithm 1 *Path Method*

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; seed set $\phi_0 \subset \mathcal{V}$
Output: Activation probability π_j^p for all nodes $j \in \mathcal{V}$

```
/* construct the evolution graph */
Let A_1^p = ∅, A_1^c = φ_0, P_1 = 1
Add cell C_1 = (A_1^p, A_1^c, P_1) to the graph
new = {C_1}, k = 1
while new ≠ ∅ do
    Pick C_i = (A_i^p, A_i^c, P_i) ∈ new, let new = new \ {i}
    S = N^out_{A_i^c} − (A_i^p ∪ A_i^c)
    for all S' ⊆ S do
        k = k + 1
        S_k = S', A_k^c = S_k
        A_k^p = A_i^p ∪ A_i^c
        P_a^(i,k) = ∏_{q∈A_i^c, r∈S−A_k^c} (1 − p(q,r)) ·
                    (1 − ∏_{q'∈A_i^c∩N^in_r', r'∈A_k^c} (1 − p(q',r')))
        if ∃C_k', s.t. A_k'^c = A_k^c and A_k'^p = A_k^p then
            P_a^(i,k') = P_a^(i,k)
            Add an arc C_i → C_k' with probability P_a^(i,k')
            P_k' = P_k' + P_i · P_a^(i,k)
            k = k − 1
        else
            Add cell C_k = (A_k^p, A_k^c, P_k) to the graph
            Add an arc C_i → C_k with probability P_a^(i,k)
            if A_k^c ≠ ∅ then
                new = new ∪ {C_k}
            end if
        end if
    end for
end while
n = k
/* activation probability computation */
for j ∈ V do
    π_j^p = 0
    for k = 1 to n do
        if A_k^c = ∅ and j ∈ A_k^p then
            π_j^p = π_j^p + P_k
        end if
    end for
end for
return π_j^p for j ∈ V
```

$A_k^p = A_i^p \cup A_i^c$ and $A_k^c = S_k$. The probability of reaching cell C_k from C_i is

$$P_a^{(i,k)} = \prod_{\substack{q \in A_i^c \\ r \in \mathcal{S} - A_k^c}} (1 - p(q,r)) \cdot \left(1 - \prod_{\substack{q' \in A_i^c \cap \mathcal{N}_{r'}^{in} \\ r' \in A_k^c}} (1 - p(q',r'))\right),$$

Hence we have

$$P_k = P_i \cdot P_a^{(i,k)}.$$

If the new cell C_k has active nodes ($A_k^c \neq \emptyset$) then C_k is added to the set new, else it is a terminal cell and will not be

explored further. If the cell C_k is equivalent to another cell C'_k already in the graph, we just add an arc from C_i to $C_{k'}$ with $P_a^{(i,k')} = P_a^{(i,k)}$ and increase the value of $P_{k'}$ by this amount. Finally after cell C_i has been explored it is removed from set new.

After constructing the evolution graph, we can search for the terminal cells whose set of past active node contains node $j \in \mathcal{V}$. Adding all the cell probabilities of these terminal cells, then π_j^p can be computed.

For example, the evolution graph for the model in Fig. 2 with seed set $\phi_0 = \{5\}$ is shown in Fig. 3. We briefly explain how to construct the evolution graph of this model. In C_1, only seed node 5 is activated, thus $A_1^p = \emptyset$ and $A_1^c = \{5\}$ with $P_1 = 1$. Since the out-neighbor set of node 5 only contains node 3, only C_2 and C_3 can be obtained from C_1: C_2 corresponds to an evolution that does not activate node 3, while C_3 corresponds to an evolution that activates node 3. C_2 is a terminal cell since $A_2^c = \emptyset$. Moreover, $P_a^{(1,3)} = p_{5,3} = 0.4$ and $P_1 = 1$, thus $P_3 = P_1 * P_a^{(1,3)} = 0.4$. Four cells can be reached from C_3 according to which subset of out-neighbor of node 3 will be activated. Based on this procedure, the evolution graph corresponding to the network can be obtained. The value of each terminal cell C_k represents the probability of observing a run whose set of finally active nodes is A_k^p. For example, C_{13} corresponds to the evolution where nodes $\{1, 2, 3, 5\}$ are influenced by the order $5 \rightarrow 3 \rightarrow \{1, 2\}$ or $5 \rightarrow 3 \rightarrow 1 \rightarrow 2$, and no other node is activated. Thus the terminal cells which contain node j as a past active node describe all final evolutions in which node j can be activated. The sum of the cell probabilities of these terminal cells is the activation probability of node j. For the network in Fig. 2, the activation probabilities obtained from the evolution graph in Fig. 3 are shown in the second row of Table 3.

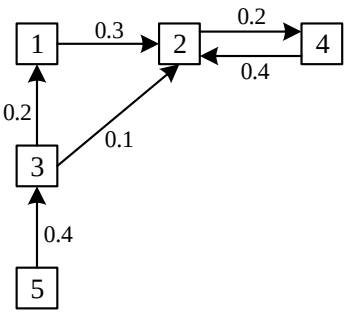


FIGURE 2. An Independent Cascade model with 5 nodes.

There are three possible states for each node $j \in \mathcal{V}$ in a cell of the evolution graph: belonging to *past active nodes*, belonging to *current active nodes* or in neither of these two sets. Thus the maximal number of cells in an evolution graph is 3^N. The number of subsets of $\mathcal{S}$ is $2^{|\mathcal{S}|}$. Since $\mathcal{S} \subset \mathcal{V}$ we have that number of subsets of $\mathcal{S}$ is bounded by 2^N. Thus the

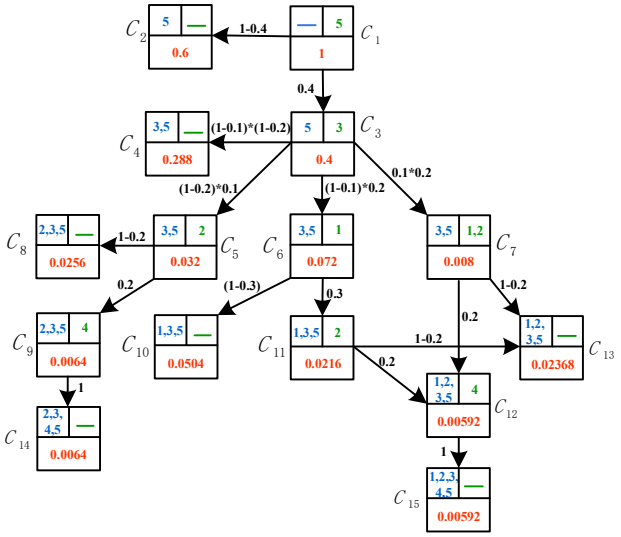


FIGURE 3. Evolution graph of the network in Fig. 2.

complexity of the part that constructs the evolution graph is $O(6^N)$. We need one pass of $j \in \mathcal{V}$ for each cell to obtain π_j^p. Thus the time complexity of the part that computes the activation probability is $O(N \cdot 3^N)$. The total complexity of Algorithm 1 is $O(6^N)$. Due to its exponential complexity, this method is only viable for small networks.

B. APPROXIMATE COMPUTATION OF ACTIVATION PROBABILITIES BY FIXED-POINT APPROACHES

Aggarwal et al. [6] proposed the *SteadyStateSpread* algorithm to evaluate the activation probabilities of nodes. This iterative method computes an approximated value of the node activation probability by solving a non-linear system of equations. Though Aggarwal et al. [6] also gave an iterative algorithm, as mentioned in a subsequent work [7], they did not prove that their iterative method can converge to only one final result. In this paper, we define the approximate value of activation probability computed by *SteadyStateSpread* algorithm for node $j \in \mathcal{V}$ as π_j^s.

Basic Fixed-Point Approach. In this part, we discuss the *SteadyStateSpread* algorithm based on fixed-point theory.

Definition 2 ([14]). *Given a real function of a real variable $f : \mathbb{R} \rightarrow \mathbb{R}$, a real number x is a fixed point of f if it satisfies*

$$x = f(x)$$

Given a point x_0 in the domain of f, the fixed-point iteration is

$$x(s+1) = f(x(s)),\ s = 0, 1, 2, \ldots$$

which generates the sequence $x(0)$, $x(1)$, $x(2)$, ... If the sequence converges to a point x and f is continuous, then one can prove that x is a fixed point of f.

To apply this theory for iterative activation probability computation, it is necessary to construct a function for

computing π_j^s. In the Independent Cascade model, node j can be activated by any of its in-neighbors. Equivalently, in order for node j to not be activated, it must not be activated by any of its in-neighbors. Assuming that the activation of the in-neighbors are independent events and that they do not depend on the activation of node j, the probability of that can be written as $\prod_{i\in\mathcal{N}_j^{in}}(1-\pi_i^s\cdot p_{i,j})$. Thus the computation function can be constructed as following:

$$\pi_j^s(s+1)=\begin{cases}1 & \text{if } j\in\phi_0\\ 1-\prod\limits_{i\in\mathcal{N}_j^{in}}(1-\pi_i^s(s)\cdot p_{i,j}) & \text{if } j\notin\phi_0\end{cases} \tag{1}$$

To prove that Equation 1 converges, we recall a classic monotone convergence theorem.

Theorem 1 ([8]). *If a sequence of real numbers is increasing and bounded above, then its supremum is the limit.*

From Theorem 1 next result follows.

Proposition 1. *The sequence $\{\pi_j^s(s)\}$ generated by Equation 1 converges to a unique fixed-point.*

Proof. Equation 1 generates a sequence for each node j, $\pi_j^s(0),\pi_j^s(1),\pi_j^s(2),...$ in which,

$$\pi_j^s(0)=\begin{cases}1 & \text{if } j\in\phi_0\\ 0 & \text{if } j\notin\phi_0\end{cases} \tag{2}$$

Such a sequence is non decreasing since there are more and more in-neighbors of node j that can propagate the innovation as the iteration proceeds. Also this sequence is upper bounded by 1. As the iteration unfolds, the sequence converges to the supremum.

Hence according to the monotone convergence theorem [8], the sequence $\{\pi_j^s(s)\}$ converges to the supremum, i.e., a fixed-point of Equation 1. □

In practice, the convergence to the fixed-point is asymptotic. However, we stop the iteration when the absolute difference between the computing results of adjacent iterations exceeds a given stopping criterion $\varepsilon^*>0$, i.e., $\sum_{j\notin\phi_0}|\pi_j^s(s+1)-\pi_j^s(s)|\geq\varepsilon^*$.

As an example, applying *SteadyStateSpread* to the network in Fig. 2, we can compute the activation probabilities shown in the third row of Table 3.

Inaccuracy of the Basic Fixed-Point Approach. *SteadyStateSpread* is generally not correct because Equation 1 holds only assuming that the activation events of node j's in-neighbors are independent events and that these events do not depend on the activation of node j itself. Thus it can not provide exact solution for every structure of networks, especially graphs containing bidirectional edges or dependent sub-structures. Yang et al. [7] discussed the scenario of *structural defect*, corresponding to this situation: nodes $i,j\notin\phi_0$ and every path from ϕ_0 to j has to pass i, nevertheless according to a certain computation algorithm, π_i depends on π_j.

Comparing the results for the network in Fig. 2 computed by *Path Method* and *SteadyStateSpread* shown in Table 3, we find $\pi_2^s>\pi_2^p$ and $\pi_4^s>\pi_4^p$ (shown in bold in Table 3). As mentioned by Yang et al. [7], one reason is: in Equation 1, π_2^s depends on π_4^s, i.e., node 4 increases π_2^s. However, node 4 can be activated only after node 2's activation. Another reason is that the dependent relation between node 2's in-neighbors (node 1 and node 3) increases the final result of node 2. Ignoring the influence of node 4, the equation to compute the activation probability of node 2 by *SteadyStateSpread* is

$$\begin{aligned}\pi_2^s &= 1-(1-\pi_3^s p_{3,2})\cdot(1-\pi_1^s p_{1,2})\\ &= \pi_3^s p_{3,2}+\pi_1^s p_{1,2}-\pi_1^s\pi_3^s p_{3,2}p_{1,2}\end{aligned} \tag{3}$$

Nevertheless, the exact equation to compute the activation probability of node 2 should be

$$\begin{aligned}\pi_2 &= \pi_3\cdot(p_{3,2}+(1-p_{3,2})\cdot p_{3,1}p_{1,2})\\ &= \pi_3 p_{3,2}+\pi_1 p_{1,2}-\pi_3 p_{3,2}p_{1,2}p_{3,1}\end{aligned} \tag{4}$$

Moreover, circuits among more than two nodes also contribute to the error in the solution computed by *SteadyStateSpread*.

Overall, circuits and dependent relations among nodes lead to computing activation probabilities by *SteadyStateSpread* that are equal to or greater than the exact solution.

Improving the Fixed-Point Approach. For partly solving the inaccuracy caused by circuits, we propose in the following an improved algorithm to assure that the iteration process to compute activation probability of node j is not influenced by itself. The new algorithm, that we call *SSS-Noself*, updates the activation probability of a node by Equation 1 without the influence of the node itself. We define the value of activation probability computed by *SSS-Noself* for node $j\in\mathcal{V}$ as π_j^n.

Given an Independent Cascade model $\mathcal{G}_{IC}=(\mathcal{V},\mathcal{E},p)$, a new network $\mathcal{G}^{[q]}=(\mathcal{V},\mathcal{E}^{[q]},p^{[q]})$ is obtained from $\mathcal{G}$ by removing the input and output arcs of node q. The total number of these new nets is N', where $N'=N-|\phi_0|$, $|\phi_0|$ is the number of nodes in a seed set. For node $j\in\mathcal{V}$ one can proceed to compute at each step s the activation probability of j assuming that q has not been activated $\pi_j^{[q]}(s)$. Finally, when updating the activation probability of node j by Equation 1, the used value of every j's in-neighbor i is $\pi_i^{[j]}$ obtained by previous iterations. Let us define the activation probability vector of network $\mathcal{G}^{[q]}$ $\boldsymbol{p}^{[q]}$ as:

$$p_{i,j}^{[q]}=\begin{cases}0 & \text{if } q=i \text{ or } q=j\\ p_{i,j} & otherwise\end{cases} \tag{5}$$

Then the computation function for $\pi_j^{[q]}(s+1)$ is constructed as:

$$\pi_j^{[q]}(s+1) = \begin{cases} 1 & \text{if } j \in \phi_0 \\ 1 - \prod_{i \in \mathcal{N}_j^{in}} (1 - \pi_i^{[q]}(s) \cdot p_{i,j}^{[q]}) & \text{if } j \notin \phi_0 \end{cases} \quad (6)$$

Algorithm 2 *SSS-Noself*

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; seed set $\phi_0 \subset \mathcal{V}$; stopping criterion $\varepsilon^* > 0$
Output: Activation probability π_j^n for all nodes $j \in \mathcal{V}$

1: $s = 0$
2: $\varepsilon = \varepsilon^* + 1$
3: **for** $q \in \mathcal{V} \setminus \phi_0$ **do**
4: $\quad \pi_j^{[q]}(0) = 1, \forall j \in \phi_0$
5: $\quad \pi_j^{[q]}(0) = 0, \forall j \in \mathcal{V} \setminus \phi_0$
6: **end for**
7: $\pi_j(0) = 1, \forall j \in \phi_0$
8: $\pi_j(0) = 0, \forall j \in \mathcal{V} \setminus \phi_0$
9: **while** $\varepsilon \geq \varepsilon^*$ **do**
10: $\quad$ **for** $q \in \mathcal{V} \setminus \phi_0$ **do**
11: $\quad\quad$ **for** $j \in \mathcal{V}$ **do**
12: $\quad\quad\quad$ **if** $j \in \phi_0$ **then**
13: $\quad\quad\quad\quad \pi_j^{[q]}(s+1) = 1$
14: $\quad\quad\quad\quad \pi_j(s+1) = 1$
15: $\quad\quad\quad$ **else**
16: $\quad\quad\quad\quad \pi_j^{[q]}(s+1) = 1 - \prod_{i \in \mathcal{N}_j^{in}} (1 - p_{i,j}^{[q]} \cdot \pi_i^{[q]}(s))$
17: $\quad\quad\quad\quad \pi_j(s+1) = 1 - \prod_{i \in \mathcal{N}_j^{in}} (1 - p_{i,j} \cdot \pi_i^{[j]}(s))$
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end for**
20: $\quad$ **end for**
21: $\quad \varepsilon_1 = \sum_{j \notin \phi_0} |\pi_j(s+1) - \pi_j(s)|$
22: $\quad \varepsilon_2 = \sum_{j \notin \phi_0} |\pi_j^{[q]}(s+1) - \pi_j^{[q]}(s)| \ \ (q \in \mathcal{V} \setminus \phi_0)$
23: $\quad \varepsilon = max(\varepsilon_1, \varepsilon_2)$
24: $\quad s = s + 1$
25: **end while**
26: **return** $\pi_j^n = \pi_j(s-1)$

Algorithm 2 is a modified version of *SteadyStateSpread* where the activation probability of node j is computed disregarding the influence of itself. The computation result for the network in Fig. 2 by Algorithm 2 is shown in the fourth row of Table 3. The results of node 2 and node 4 are in bold in Table 3 to highlight that they are different from their results by *Path Method* and *SteadyStateSpread*. It is obviously that the result of *SSS-Noself* is closer to the result of *Path Method* than *SteadyStateSpread*, i.e., *SSS-Noself* is more precise than *SteadyStateSpread*. In fact, *SSS-Noself* always gives a result between the result of *Path Method* and the result of *SteadyStateSpread*.

Different from *SteadyStateSpread*, for all nodes $j \in \mathcal{V}$, *SSS-Noself* not only computes the activation probability of node j at step s, but also the activation probability of node j at step s assuming that any node $q \in \mathcal{V} \setminus \phi_0$ remains inactive. Thus the time complexity of *SSS-Noself* is $O(N^2)$, while that of *SteadyStateSpread* is $O(N)$.

Fixed-Point Computation of Activation Probabilities Along Paths of Bounded Length. In this part, we propose an efficient algorithm, called *SSS-Bounded-Path*, to compute activation probabilities along paths of bounded length by applying Equation 1. It represents a family of efficient approaches for activation computation that is parameterized by the value of the bound b_0. The value of activation probability computed by *SSS-Bounded-Path* with bound $b_0 \in \mathbb{N}$ for node $j \in \mathcal{V}$ is denoted as $\pi_j^{bp(b_0)}$.

Let us firstly define the length of the shortest path from the seed set to a node $j \in \mathcal{V} \setminus \phi_0$ as sp_j and assume each node $j \in \mathcal{V} \setminus \phi_0$ is reachable from the seed set. Kimura et al. [18] proposed SPM and SP1M, where node j can be activated only at step $b = sp_j$ in SPM, or only at step $b = sp_j$ as well as step $b = sp_j + 1$ in SP1M. Nevertheless, they did not discuss how to compute these probabilities without previously determining the corresponding paths, a procedure that may be computationally expensive. Chen et al. [3] and Yang et al. [7] computed the maximum influence paths by the Dijkstra algorithm, which has high complexity.

We propose an approach based on fixed-point computation that does not require preliminarily computing the shortest path. In our procedure we compute sp_j as step s when π_j^{bp} firstly changes from zero to non-zero. Besides, we set the path bound to compute π_j^{bp} involving not only the shortest paths but also the paths whose length is no greater than $sp_j + b_0$, where b_0 is a constant integer called bound. Obviously, we have SPM when $b_0 = 0$, and SP1M when $b_0 = 1$. Besides, when b_0 is large enough, this algorithm is equivalent to *SteadyStateSpread*.

The procedure of *SSS-Bounded-Path* is shown in Algorithm 3. We denote the upper bound of iteration time for node j as b_j and it is initialized as infinity. Lines (11-12) find the step when π_j^{bp} firstly changes from zero to non-zero and record this step $s + 1$ as sp_j. Then b_j is set as $s + 1 + b_0$. The computation result for the network in Fig. 2 by *SSS-Bounded-Path* with $b_0 = \{0, 1, 2, 3, 4\}$ is shown in the fifth row of Table 3.

The *SSS-Bounded-Path* algorithm generalizes SPM [18] and SP1M [18], exploiting the efficient fixed-point computation of *SteadyStateSpread*. The result of *SSS-Bounded-Path* ($b_0 = 0$) can be regarded as a lower-bound for the exact activation probability. Same with *SteadyStateSpread*, the time complexity of *SSS-Bounded-Path* is $O(N)$. However, in most cases *SSS-Bounded-Path* ($b_0 = 0$) stops the iteration before it converges, thus *SSS-Bounded-Path* ($b_0 = 0$) is usually less time-consuming than *SteadyStateSpread*.

C. COMPARISON OF DIFFERENT FIXED-POINT APPROACHES

We show the activation probability computation results by different approaches for the network in Fig. 2 in Table 3. As

Algorithm 3 *SSS-Bounded-Path*

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; seed set $\phi_0 \subset \mathcal{V}$; path bound $b_0 \in \mathbb{N}$; stopping criterion $\varepsilon^* > 0$

Output: Activation probability π_j^{bp} for all nodes $j \in \mathcal{V}$

Initialize $\pi_j^{bp}(0) = 1, j \in \phi_0$; $\pi_j^{bp}(0) = 0, j \in \mathcal{V} \setminus \phi_0$; $s = 0$; $b_j = inf, j \in \mathcal{V} \setminus \phi_0$
$stop = 0$
$\varepsilon = \varepsilon^* + 1$
while $\varepsilon \geq \varepsilon^*$ **do**
  while $stop = 0$ **do**
    $stop = 1$
    for $j \in \mathcal{V}$ **do**
      $\pi_j^{bp}(s+1) = \pi_j^{bp}(s)$
      if $j \notin \phi_0$ and $s \leq b_j$ **then**
        $\pi_j^{bp}(s+1) = 1 - \prod_{i \in \mathcal{N}_j^{in}} (1 - p_{i,j} \cdot \pi_i^{bp}(s))$
        $stop = 0$
      end if
      if $\pi_j^{bp}(s+1) \neq 0$ and $\pi_j^{bp}(s) = 0$ **then**
        $b_j = s + 1 + b_0$
      end if
    end for
    $s = s + 1$
  end while
  $\varepsilon = \sum_{j \notin \phi_0} |\pi_j^{bp}(s+1) - \pi_j^{bp}(s)|$
end while
return $\pi_j^{bp} = \pi_j^{bp}(s-1)$

discussed above, *Path Method* provides the exact value, while *SteadyStateSpread* gives a larger value. *SSS-Noself* gives a more precise result than *SteadyStateSpread*, while *SSS-Bounded-Path* ($b_0 = 0$) provides a lower-bound. In fact, the activation probabilities value computed by these algorithms must satisfy the inequality, shown in Proposition 2.

TABLE 3. Comparison of activation probability values for the network in Fig. 2.

Node	1	2	3	4	5
Path Method	0.08	**0.0616**	0.4	**0.0123**	1
SteadyStateSpread	0.08	**0.0678**	0.4	**0.0132**	1
SSS-Noself	0.08	**0.0630**	0.4	**0.0126**	1
SSS-Bounded-Path ($b_0 = 0$)	0.08	**0.0400**	0.4	**0.0080**	1
SSS-Bounded-Path ($b_0 = 1$)	0.08	**0.0630**	0.4	**0.0126**	1
SSS-Bounded-Path ($b_0 = 2$)	0.08	**0.0660**	0.4	**0.0132**	1
SSS-Bounded-Path ($b_0 = 3$)	0.08	**0.0678**	0.4	**0.0136**	1
SSS-Bounded-Path ($b_0 = 4$)	0.08	**0.0680**	0.4	**0.0136**	1

Proposition 2. *Given an Independent Cascade model $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$ and a set of initial nodes $\phi_0 \subset \mathcal{V}$, the activation probabilities of node j computed by Path Method (π_j^p), SteadyStateSpread (π_j^s), SSS-Bounded-Path ($b_0 = 0$) ($\pi_j^{bp(0)}$) and SSS-Noself (π_j^n) satisfy:*

$$\pi_j^{bp(0)} \leq \pi_j^p \leq \pi_j^n \leq \pi_j^s \tag{7}$$

Proof. First, we prove that $\pi_j^n \leq \pi_j^s$. During the computing process of π_j^s by Equation 1, the activation probabilities of $\mathcal{N}_j^{in}$ may include the influence of node j. This extra influence erroneously increases the final value of mode j. *SSS-Noself* algorithm disregards this extra influence during the iteration, hence $\pi_j^n \leq \pi_j^s$.

Second, we prove that $\pi_j^p \leq \pi_j^n$. Although *SSS-Noself* avoids the influence of j when computing π_j^n, it has not eliminated the increase caused by dependent relation and other redundant influence in circuits while applying Equation 1. Hence for some special network structures, π_j^n is still bigger than π_j^p.

Finally, we prove that $\pi_j^{bp(0)} \leq \pi_j^p$. *SSS-Bounded-Path* ($b_0 = 0$) only consider the influence to node j through the shortest path, i.e., it disregards the influence through the paths from ϕ_0 to node j whose lengths are greater than sp_j. However, this is just a fraction of the influence on node j, hence we have $\pi_j^{bp(0)} \leq \pi_j^p$. □

Considering *SteadyStateSpread* and *SSS-Bounded-Path*, we also have the following remarks:

Remark 1. *Given an Independent Cascade model $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$ and a set of initial nodes $\phi_0 \subset \mathcal{V}$, for $\forall j \in \mathcal{V}$, we have $\pi_j^{bp(b)} \leq \pi_j^{bp(b')}$ when $0 \leq b < b'$.*

Obviously, $\pi_j^{bp(b')}$ is computed considering more paths than $\pi_j^{bp(b)}$, i.e., the paths whose lengths are between b and b'. Hence, we have $\pi_j^{bp(b)} \leq \pi_j^{bp(b')}$ when $0 \leq b < b'$.

Remark 2. *Given an Independent Cascade model $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$ and a set of initial nodes $\phi_0 \subset \mathcal{V}$, for $\forall j \in \mathcal{V}$, we have $\lim_{b \to \infty} \pi_j^{bp(b)} = \pi_j^s$ when the same stopping criterion ε^* is used for both SteadyStateSpread and $SSS - Bounded - Path$.*

SSS-Bounded-Path limits the computation iteration to b steps. When b goes to infinity, only the satisfaction of stopping criterion halts the computation. In that case, *SSS-Bounded-Path* is equivalent to *SteadyStateSpread*, hence we have $\lim_{b \to \infty} \pi_j^{bp(b)} = \pi_j^s$.

IV. INFLUENCE MAXIMIZATION

As discussed above, activation probability computation is an important preliminary step for the influence maximization, which consists in maximizing the influence spread by targeting a subset of individuals to adopt an innovation initially. In this section, we give three basic algorithms, named *SelectTopK*, *RankedReplace* and greedy algorithm, which will later be combined with the approaches for activation probability computation in Section 3 to solve the influence maximization problem.

A. *SELECTTOPK* ALGORITHM

In order to select a set of K nodes to maximize the final influence spread, a basic idea is as follows: let each node $j \in \mathcal{V}$ be the single seed node, i.e., $\phi_0 = \{j\}$, then compute

the total activation probabilities by one of algorithms discussed in Section 3. Having known the influence spread when each node is as the single seed node, we select the K nodes with the largest influence spread as seed set ϕ_0. The detail is described in Algorithm 4.

Algorithm 4 *SelectTopK*

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; an integer $K \in \mathbb{N}^+$
Output: Seed set ϕ_0
1: Compute the influence spread $\sigma(\{j\})$ for each node $j \in \mathcal{V}$
2: Select K nodes with the highest value of $\sigma(\cdot)$ as seed set ϕ_0
3: **return** seed set ϕ_0

Let T to be the time complexity required to compute the activation probability for a given net with N nodes. As we have previously discussed, these value has order $O(N \cdot 3^N)$ for *Path Method*, $O(N)$ for *SteadyStateSpread*, $O(N^2)$ for *SSS-Noself* and $O(N)$ for *SSS-Bounded-Path*. Then the time complexity of Algorithm 4 is $O(NT)$ since it computes the influence spread for each node $j \in \mathcal{V}$ as a single seed node.

B. RANKEDREPLACE ALGORITHM

To further improve the SelectTop$K$ algorithm, a number of replacements of seed nodes happen after the selection of initial seed set. As shown in Algorithm 5 [6], the nodes in $\mathcal{V} \setminus \phi_0$ are sorted in descending order of the influence spread value $\sigma(\phi_0)$. Then in each iteration, the nodes in ϕ_0 are sorted in ascending order of influence spread value $\sigma(\phi_0)$. We pick in order the node in $\mathcal{V} \setminus \phi_0$ to replace a node in ϕ_0, if this replacement can increase the final influence spread. Note that in ascending order only the first replacement of a node in ϕ_0 which increases $\sigma(\phi_0)$ is executed.

Algorithm 5 *RankedReplace*

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; an integer $K \in \mathbb{N}^+$
Output: Seed set ϕ_0
1: Compute the influence spread $\sigma(\{j\})$ for each node $j \in \mathcal{V}$
2: Initialize seed set ϕ_0 by K nodes with the highest value of $\sigma(\cdot)$
3: Sort nodes $j \in \mathcal{V} \setminus \phi_0$ in descending order of $\sigma(\{j\})$
4: **for** $j \in \mathcal{V} \setminus \phi_0$ in descending order of $\sigma(j)$ **do**
5: Sort nodes $i \in \phi_0$ in ascending order of $\sigma(\{i\})$
6: **for** $i \in \phi_0$ in ascending order of $\sigma(\{i\})$ **do**
7: **if** $\sigma(\phi_0 \cup \{j\} \setminus \{i\}) > \sigma(\phi_0)$ **then**
8: $\phi_0 = \phi_0 \cup \{j\} \setminus \{i\}$
9: **break**
10: **end if**
11: **end for**
12: **end for**
13: **return** seed set ϕ_0

Let T to be the time complexity required to compute the activation probability for a given net with N nodes. As we have previously discussed, these value has order $O(N \cdot 3^N)$ for *Path Method*, $O(N)$ for *SteadyStateSpread*, $O(N^2)$ for *SSS-Noself* and $O(N)$ for *SSS-Bounded-Path*. Then the time complexity of Algorithm 5 is $O(K(N-K)T)$, with $N = |\mathcal{V}|$ and $K = |\phi_0|$.

C. GREEDY ALGORITHM FOR INFLUENCE MAXIMIZATION

Algorithm 6 describes the general greedy algorithm for influence maximization which can guarantee that the influence spread ϕ_0 is within $(1 - 1/e)$ of the optimal value, as pointed by Kempes et al. [2]. In this algorithm, the node which maximizes the incremental influence spread is selected in each iteration.

Algorithm 6 Greedy Algorithm for Influence Maximization

Input: An independent cascade network $\mathcal{G}_{IC} = (\mathcal{V}, \mathcal{E}, p)$; an integer $K \in \mathbb{N}^+$
Output: Seed set ϕ_0
1: Initialize $\phi_0 = \emptyset$
2: **for** $q = 1$ to K **do**
3: Select $i = argmax_{j \in \mathcal{V} \setminus \phi_0} \{\sigma(\phi_0 \cup \{j\}) - \sigma(\phi_0)\}$
4: $\phi_0 = \phi_0 \cup \{i\}$
5: **end for**
6: **return** seed set ϕ_0

Let T to be the time requiring to compute the activation probability (i.e., $O(N \cdot 3^N)$ for *Path Method*, $O(N)$ for *SteadyStateSpread*, $O(N^2)$ for *SSS-Noself* and $O(N)$ for *SSS-Bounded-Path*), then the time complexity of Algorithm 6 is $O(KNT)$, with $N = |\mathcal{V}|$ and $K = |\phi_0|$.

V. EXPERIMENTAL RESULTS

Our experiment is performed in two parts. First, we compare the influence spread with fixed seed set computed by Monte Carlo simulation, *Path Method*, *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path*. Second, we evaluate the performances of *SteadyStateSpread* and *SSS-Noself* in terms of seed set selection for influence maximization combined with the *SelectTopK*, *RankedReplace* and greedy algorithm.

All approaches are implemented in MATLAB. All experiments are run on a PC with 2.40GHz Intel Core i5 Processor and 8GB memory.

A. ACTIVATION PROBABILITY COMPUTATION

Data Set. We consider two datasets for comparing the different algorithms we have discussed for activation probability computation.

First, we construct a series of bidirectional grid graphs with a parameter m such that the $m-$th grid graph contains m^2 nodes. Fig. 4 shows the grid graphs for $m \in \{2, 3, 4\}$. For each edge (i, j), we uniformly at random select $p_{i,j}$ from the set $\{0.1, 0.2, 0.5\}$. We represent this dataset as **Series-Grid**.

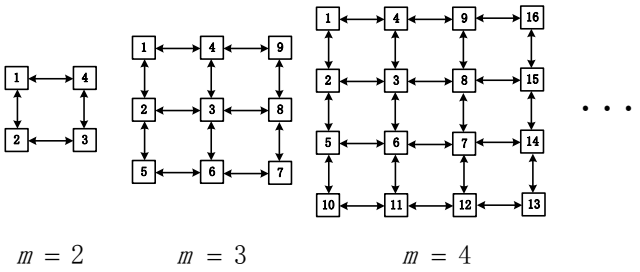


FIGURE 4. Series of grid graphs.

The second dataset is a real-world network — **airportsinUS** [27] which is a benchmark network widely used in social network analysis. It is a weighted network of the 500 airports with the largest amount of traffic from publicly available data in the United States. Nodes represent US airports and edges represent air travel connections among them. There are 5960 edges in total. Based on the weights $w_{i,j}$ of edges, we obtain $p_{i,j}$ by $w_{i,j} \setminus \sum_i w_{i,j}$.

Experimental Setup. We compare *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path* for activation probability computation in terms of effectiveness and efficiency. For each network dataset, the seed set is randomly chosen with a certain size. The tested algorithms are briefly described as follows:

Monte Carlo simulation: the average of 10,000 simulation runs. Kempe et al. [2] showed that the quality of approximation after 10,000 iterations is comparable to that after 300,000 or more iterations. The simulation process is described as: assume node i attempts to activate node j at step t, then generate a random number uniformly distributed in the interval $[0, 1]$. The innovation successfully propagates from node i to node j when the random number does not exceed $p_{i,j}$.

Path Method: the exact computation method proposed in Section 3.1.

SteadyStateSpread: the heuristic [6] described in Section 3.2.

SSS-Noself: the improved algorithm proposed in Section 3.2.

SSS-Bounded-Path: the algorithm proposed in Section 3.2.

Experimental Results. First, we present the computation results of activation probabilities on Series-Grid using *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path*. We randomly select one node as seed node for the grid graphs with $m = \{2, 3\}$ and two nodes for the grid graphs with $m = \{4, 5, 6, 7\}$. We set $\varepsilon^* = 10^{-8}$ for the iterations of *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path*. In order to show the convergence of *SSS-Bounded-Path*, we set $b_0 = \{0, 1, 2, 3, 4, 20, 40, 65, 85, 135\}$. The sum of activation probabilities of nodes on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$ using *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread* and *SSS-Noself* is shown in Table 4. The value for $m = \{4, 5, 6, 7\}$ by the *Path Method* is not given since the running time is more than 8 hours, i.e., out of time (o.o.t). The sum of activation probabilities of nodes on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$ using *SSS-Bounded-Path* is shown in Table 5. As a particular case, we list activation probability of each node for the grid graph with $m = 3$ in Table 6 and Table 7 to show the difference of every node by these five methods.

We can observe that while *SSS-Bounded-Path* ($b_0 = 0$) provides a lower-bound, the result by *SSS-Noself* is always between the exact result by *Path Method* and the result by *SteadyStateSpread* for both activation probability of each node and sum of activation probabilities of all node. It verifies the relationship among these four methods in Proposition 2. We can also figure out that within certain paths, the results computed by *SSS-Bounded-Path* converges to the results computed by *SteadyStateSpread*. This is proved in Remark 2. Moreover, it shows that our *SSS-Noself* algorithm provides more precise results than *SteadyStateSpread*.

TABLE 4. Sum of activation probabilities computed by *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread*, and *SSS-Noself* on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$.

Method	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$	$m = 7$
Monte Carlo simulation	1.4343	1.9055	3.2171	4.5072	4.5527	5.1308
Path Method	1.4428	1.9098	o.o.t	o.o.t	o.o.t	o.o.t
SteadyStateSpread	1.4467	2.0936	5.2166	5.8790	7.0053	13.1089
SSS-Noself	1.4428	1.9502	4.0661	5.0710	5.7296	9.1891

TABLE 5. Sum of activation probabilities computed by *SSS-Bounded-Path* with path bound $b_0 = \{0, 1, 2, 3, 4, 20, 40, 65, 85, 135\}$ on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$.

Path bound	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$	$m = 7$
$b_0 = 0$	1.4396	1.8769	2.8447	3.9560	4.0748	4.5891
$b_0 = 1$	1.4396	1.8769	3.0590	4.1876	4.2843	4.8252
$b_0 = 2$	1.4466	2.0078	3.3066	4.7612	4.8362	5.6117
$b_0 = 3$	1.4466	2.0172	3.5094	4.9387	4.9569	5.7510
$b_0 = 4$	1.4467	2.0566	3.6957	5.2234	5.3128	6.2450
$b_0 = 20$	1.4467	2.0936	5.0722	5.8717	6.8181	10.1654
$b_0 = 40$	1.4467	2.0936	5.2119	5.8790	6.9949	12.5893
$b_0 = 65$	1.4467	2.0936	5.2166	5.8790	7.0050	13.0701
$b_0 = 85$	1.4467	2.0936	5.2166	5.8790	7.0053	13.1045
$b_0 = 135$	1.4467	2.0936	5.2166	5.8790	7.0053	13.1089

TABLE 6. Activation probabilities computed by *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread*, and *SSS-Noself* on Series-Grid with $m = 3$.

Node	1	2	3	4	5	6	7	8	9
Monte Carlo simulation	1	0.5076	0.0962	0.1099	0.1048	0.0302	0.0151	0.0168	0.0249
Path Method	1	0.5032	0.1009	0.1136	0.1049	0.0302	0.0161	0.0169	0.0238
SteadyStateSpread	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320
SSS-Noself	1	0.5049	0.1119	0.1182	0.1093	0.0345	0.0206	0.0227	0.0280

Second, we compare the running time of these five methods for the activation probability computation, shown in Table 8 and Table 9. We can observe that *Path Method* takes exponential time to give exact results as the size of network increases. *SSS-Noself* provides better results than

TABLE 7. Activation probabilities computed by *SSS-Bounded-Path* with $b_0 = \{0, 1, 2, 3, 4, 20, 40, 65, 85, 135\}$ on Series-Grid with $m = 3$.

Node	1	2	3	4	5	6	7	8	9
$b_0 = 0$	1	0.5000	0.0975	0.1000	0.1000	0.0296	0.0161	0.0137	0.0200
$b_0 = 1$	1	0.5000	0.0975	0.1000	0.1000	0.0296	0.0161	0.0137	0.0200
$b_0 = 2$	1	0.5296	0.1189	0.1264	0.1191	0.0432	0.0236	0.0203	0.0266
$b_0 = 3$	1	0.5296	0.1213	0.1315	0.1191	0.0434	0.0238	0.0208	0.0276
$b_0 = 4$	1	0.5354	0.1279	0.1382	0.1264	0.0491	0.0268	0.0233	0.0296
$b_0 = 20$	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320
$b_0 = 40$	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320
$b_0 = 65$	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320
$b_0 = 85$	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320
$b_0 = 135$	1	0.5392	0.1349	0.1475	0.1317	0.0535	0.0292	0.0255	0.0320

SteadyStateSpread with an acceptable increase of computation time for the considered small networks. As b_0 increases, *SSS-Bounded-Path* involves more paths of the network, thus it cost a little more time to compute.

TABLE 8. Running time for activation probability computation by *Monte Carlo simulation*, *Path Method*, *SteadyStateSpread*, and *SSS-Noself* on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$.

Running time (s)	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$	$m = 7$
Monte Carlo simulation	0.48	1.25	0.95	1.40	1.82	1.58
Path Method	0.11	0.31	o.o.t	o.o.t	o.o.t	o.o.t
SteadyStateSpread	0.01	0.02	0.10	0.09	0.34	0.55
SSS-Noself	0.01	0.06	0.66	1.42	4.80	11.41

TABLE 9. Running time for activation probability computation by *SSS-Bounded-Path* with $b_0 = \{0, 1, 2, 3, 4, 20, 40, 65, 85, 135\}$ on Series-Grid with $m = \{2, 3, 4, 5, 6, 7\}$.

Running time (s)	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$	$m = 7$
$b_0 = 0$	0.01	0.01	0.01	0.01	0.01	0.01
$b_0 = 1$	0.01	0.01	0.01	0.01	0.01	0.01
$b_0 = 2$	0.01	0.01	0.01	0.01	0.01	0.01
$b_0 = 3$	0.01	0.01	0.01	0.01	0.01	0.01
$b_0 = 4$	0.01	0.01	0.01	0.01	0.01	0.01
$b_0 = 20$	0.01	0.01	0.01	0.01	0.01	0.02
$b_0 = 40$	0.01	0.01	0.01	0.01	0.02	0.03
$b_0 = 65$	0.01	0.01	0.01	0.02	0.03	0.02
$b_0 = 85$	0.01	0.01	0.01	0.03	0.03	0.04
$b_0 = 135$	0.01	0.02	0.02	0.03	0.05	0.07

Another experiment of activation probability computation is performed on airportsinUS network data. We evaluate the sum of activation probabilities for all nodes by *Monte Carlo simulation*, *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path* given different sizes of seed sets, shown in Fig. 5. We have not given the value computed by *Path Method* since it can not be obtained within limited time for this size of network. The seven seed sets are randomly generate with size $|\phi_0| = \{1, 5, 10, 15, 20, 25, 30\}$. The result of *Monte Carlo simulation* is obtained by the average of 10,000 simulation runs proceeded as described in the first experiment. The stopping criterion is fixed as $\varepsilon^* = 0.01$ for *SteadyStateSpread*, *SSS-Noself* and *SSS-Bounded-Path*. The path bound b_0 is chosen from $\{0, 1, 5, 10, 15, 20, 25, 30\}$ for *SSS-Bounded-Path*. We can observe that the results computed by these approaches are consistent with Equation 7. According to the proved relationship in Proposition 2, although we have not been able to compute the exact value by *Path Method* because of the net size, we can figure out that *SSS-Noself* is more precise than *SteadyStateSpread*. Moreover, we can see that the results of *SSS-Bounded-Path* increase as the path bound b_0's increase. As a lower-bound, the values computed by *SSS-Bounded-Path* ($b_0 = 0$) are the smallest among all approaches under the same seed set size.

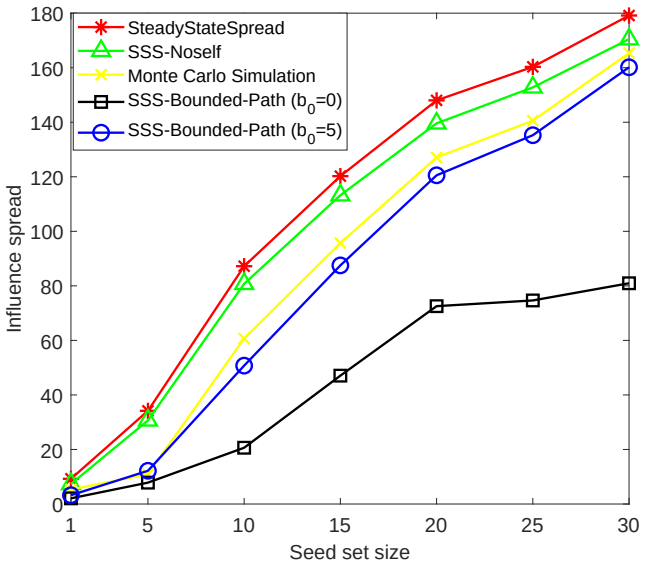


FIGURE 5. Sum of activation probabilities computed by *Monte Carlo simulation*, *SteadyStateSpread*, *SSS-Noself*, *SSS-Bounded-Path* ($b_0 = 0$) and *SSS-Bounded-Path* ($b_0 = 5$) on airportsinUS network data.

However, as shown in Table 10, the running time of *SSS-Noself* is much longer than *SteadyStateSpread* when the size of the network is large since *SSS-Noself* needs one more pass of all nodes in a network than *SteadyStateSpread*. For this reason we think that it may be necessary to further improve the efficiency of *SSS-Noself*. Compared with the running time of *SSS-Bounded-Path* in Table 11, *SSS-Bounded-Path* is obviously faster than *SteadyStateSpread* when $b_0 = \{0, 1, 5, 10, 15, 20\}$. As b_0 increases, the running time of these two approaches will be similar.

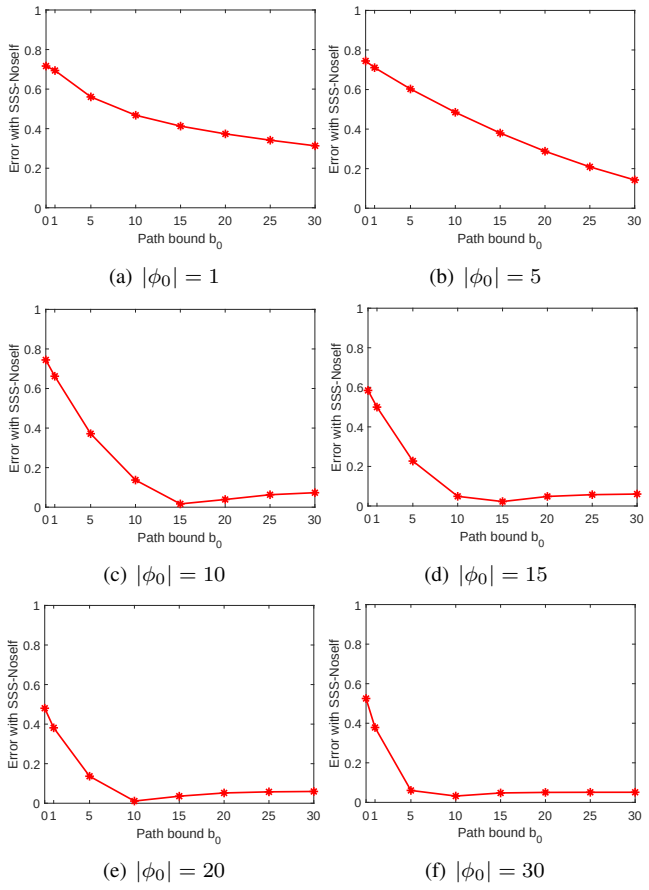
TABLE 10. Running time for activation probability computation by *Monte Carlo simulation*, *SteadyStateSpread* and *SSS-Noself* on airportsinUS network data given different sizes of seed sets $|\phi_0|$.

Running time (s)	$\|\phi_0\| = 1$	$\|\phi_0\| = 5$	$\|\phi_0\| = 10$	$\|\phi_0\| = 15$	$\|\phi_0\| = 20$	$\|\phi_0\| = 25$	$\|\phi_0\| = 30$
Monte Carlo simulation	3.86	6.23	36.39	57.70	68.81	98.59	110.99
SteadyStateSpread	5.05	3.20	1.35	1.18	1.10	0.94	0.82
SSS-Noself	1143.62	892.77	719.44	467.74	512.77	383.86	350.49

TABLE 11. Running time for activation probability computation by *SSS-Bounded-Path* on airportsinUS network data given different sizes of seed sets $|\phi_0|$ with path bound $b_0 = \{0, 1, 5, 10, 15, 20, 25, 30\}$.

Running time (s)	$\|\phi_0\| = 1$	$\|\phi_0\| = 5$	$\|\phi_0\| = 10$	$\|\phi_0\| = 15$	$\|\phi_0\| = 20$	$\|\phi_0\| = 25$	$\|\phi_0\| = 30$
$b_0 = 0$	0.12	0.12	0.11	0.14	0.12	0.11	0.12
$b_0 = 1$	0.16	0.22	0.16	0.18	0.16	0.15	0.16
$b_0 = 5$	0.26	0.28	0.28	0.30	0.27	0.27	0.28
$b_0 = 10$	0.36	0.38	0.38	0.40	0.38	0.37	0.37
$b_0 = 15$	0.61	0.57	0.56	0.58	0.56	0.56	0.57
$b_0 = 20$	0.57	0.64	0.63	0.65	0.64	0.63	0.62
$b_0 = 25$	0.87	0.87	0.86	0.88	0.86	0.86	0.86
$b_0 = 30$	0.90	0.90	0.91	0.91	0.89	0.88	0.89

Fig. 6 shows the error between *SSS-Bounded-Path* and *SSS-Noself* which is measured by $|\sum_{j\in\mathcal{V}\setminus\phi_0}\pi_j^{bp} - \sum_{j\in\mathcal{V}\setminus\phi_0}\pi_j^{n}| \setminus \sum_{j\in\mathcal{V}\setminus\phi_0}\pi_j^{n}$. We do not present the curve for $|\phi_0| = 25$ due to the limit of space, but point out that it is similar to the curves for $|\phi_0| = \{15, 20, 30\}$. We can find that in the beginning the error decreases as the path bound b_0's increases. At certain path bound the error is the smallest and then increases a bit. Results show that the path bound corresponding to the smallest error varies with different seed set size.



(a) $|\phi_0| = 1$ (b) $|\phi_0| = 5$ (c) $|\phi_0| = 10$ (d) $|\phi_0| = 15$ (e) $|\phi_0| = 20$ (f) $|\phi_0| = 30$

FIGURE 6. Error between *SSS-Bounded-t* and *SSS-Noself* on airportsinUS network data given different sizes of seed sets $|\phi_0|$ with path bound $b_0 = \{0, 1, 5, 10, 15, 20, 25, 30\}$.

B. OPTIMAL SEED SET SELECTION

Data Set. In this part, the real-world dataset used in seed set selection for influence maximization is **HighSchool** [26]. It is a directed network, containing friendship links among 73 boys in a small high-school in Illinois. A node represents a boy and an edge from node i to node j shows that the $i-$th boy chose the $j-$th boy as a friend. The activation probability $p_{i,j}$ is randomly selected from the set $\{0.1, 0.2, 0.5\}$.

Experimental Setup. In this part of experiment, we evaluate the performances of *SteadyStateSpread* and *SSS-Noself* in terms of selecting seed set to maximize the influence propagation. During the process of seed set selection by *SelectTopK*, *RankedReplace* or greedy algorithm, we apply *SteadyStateSpread* or *SSS-Noself* to compute the activation probability. After selecting the seed set by these different combination of methods, the influence spread of the selected seed set is evaluated by running *Monte Carlo simulation* for 10,000 times. The tolerance for *SteadyStateSpread* and *SSS-Noself* is fixed as $\varepsilon^* = 0.01$. The tested algorithms are briefly described as following:

Random: Randomly select a set of nodes to be activated.

SelectTopK-SSS: Compute the activation probability of each node by *SteadyStateSpread* and then select K nodes with the largest influence spread as the seed nodes.

SelectTopK-SN: Compute the activation probability of each node by *SSS-Noself* and then select K nodes with the largest influence spread as the seed nodes.

Replace-SSS: Compute the activation probability of each node by *SteadyStateSpread* and then select seed nodes by *RankedReplace*.

Replace-SN: Compute the activation probability of each node by *SSS-Noself* and then select seed nodes by *RankedReplace*.

Greedy-SSS: Select seed nodes by greedy algorithm, in which computing the activation probability by *SteadyStateSpread*.

Greedy-SN: Select seed nodes by greedy algorithm, in which computing the activation probability by *SSS-Noself*.

Experimental Results. We evaluate the algorithms above on the Highschool network under the Independent Cascade model in terms of the influence spread and the running time. The influence spread is denoted with the total activation probabilities of all nodes in the network. After selecting a seed set by any one of the above algorithms, the influence spread is computed by *Monte Carlo simulation*.

The influence spread with a seed set of size $K = \{1, 5, 10, 15, 20, 25\}$ computed by different algorithms is shown in Fig. 7. Regardless of which approach computes the activation probability, either *SteadyStateSpread* or *SSS-Noself*, it is obvious that greedy algorithm performs better than *RankedReplace*, and *RankedReplace* performs better than *SelectTopK* when $K = \{5, 10, 15, 20, 25\}$. Moreover, based on the same algorithm for seed set selection (*SelectTopK*, *RankedReplace* or greedy algorithm), *SSS-Noself* can select a better set of seed nodes, i.e., give a larger influence spread, compared with *SteadyStateSpread*.

Fig. 8 shows the running time for selecting nodes by different algorithms above. *Greedy*-SN takes much longer time than other algorithms. Although *SelectTopK*-SN takes a bit longer time than *SelectTopK*-SSS and as well *Replace*-SN takes a bit longer time than *Replace*-SSS, these four algorithms are efficient enough.

VI. CONCLUSION

In this paper, we analyzed different approaches for influence spread computation in an Independent Cascade model. We initially proposed an approach which can compute the exact

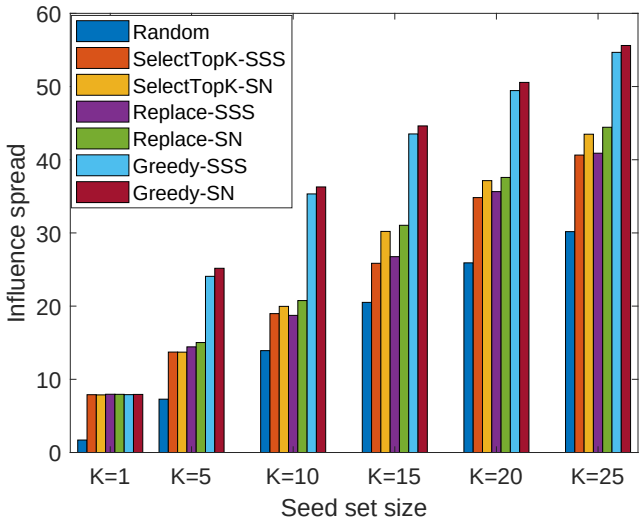


FIGURE 7. Influence spread computed by different algorithms on Highschool network data given size of seed set $K = \{1, 5, 10, 15, 20, 25\}$.

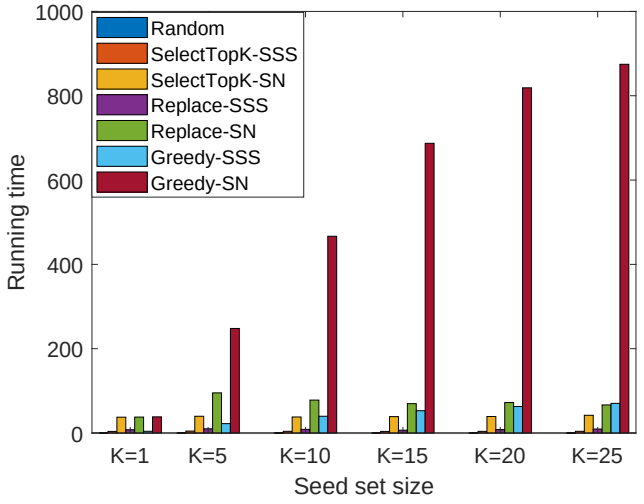


FIGURE 8. Running time for selecting seed nodes by different algorithms on Highschool network data given size of seed set $K = \{1, 5, 10, 15, 20, 25\}$.

value of the influence spread called *Path Method*. We also discussed the convergence properties of the existing algorithm *SteadyStateSpread*, showing that it converges to a fixed solution by fixed-point theory. We also considered the elements resulting in the inaccuracy of *SteadyStateSpread*: the dependent relation between nodes and the existence of circuits. Furthermore, we showed how to compute a lower approximation of activation probabilities by *SSS-Bounded-Path* and proposed an improved algorithm called *SSS-Noself* which partially decreases the error caused by circuits.

Moreover, focusing on the influence maximization problem, we evaluated these approaches in terms of selecting seed set by combining a selection strategy based on *SelectTopK*, *RankedReplace* and greedy algorithm.

Aware of the factors which cause the inaccuracy of *SteadyStateSpread*, proposing new algorithms to improve the effectiveness will be the objective of our future work. Besides, since we observed that *SSS-Noself* is quite time consuming in large-scale networks, another interesting line of research could be to improve the efficiency of the *SSS-Noself*. Besides, opposite to the influence maximization problem, the influence minimization problem has also received much attention recently in the domain of innovation diffusion through social networks. We believe that these computational approaches to determine influence spread can also be applied to solve this problem: our future research will explore this issue.

REFERENCES

[1] D. Rosa and A. Giua, "On the spread of innovation in social networks," in *NecSys*, Rhine-Moselle-Hall, Koblenz, Germany, 2013, pp. 322-327.

[2] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th KDD*, Washington, DC, USA, 2003, pp. 137-146.

[3] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. 16th KDD*, Washington, DC, USA, 2010, pp. 1029-1038.

[4] C. Zhou, P. Zhang, J. Guo, X. Zhu, and L. Guo, "Ublf: an upper bound based approach to discover influential nodes in social networks," in *IEEE 13th ICDM*, Dallas, Texas, 2013, pp. 907-916.

[5] D. Kempe, J. M. Kleinberg, and É. Tardos, "Influential nodes in a diffusion model for social networks," in *Int. Colloq. Autom. Lang., Program.*, Lisbon, Portugal, 2005, pp. 1127-1138.

[6] C. C. Aggarwal, A. Khan, and X. Yan, "On flow authority discovery in social networks," in *Proc. SDM*, Mesa, Arizona, USA, 2011, pp. 522-533.

[7] Y. Yang et al., "On approximation of real-world influence spread," in *ECML-PKDD*, Bristol, United Kingdom, 2012, pp. 548-564.

[8] E. Schechter, "Handbook of analysis and its Foundations," *Academic Press*, 1996.

[9] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proc. 7th KDD*, San Francisco, CA, USA, 2001, pp. 57-66.

[10] B. Ryan and N. C. Gross, "The diffusion of hybrid seed corn in two lowa communities," *Rural Sociology*, vol. 8, no. 1, pp. 15-24, 1943.

[11] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. 15th KDD*, Paris, France, 2009, pp. 199-208.

[12] J. S. Coleman, E. Katz, and H. Menzel, "Medical innovation: a diffusion study," *Bobbs-Merrill Co.*, 1966.

[13] J. Leskovec et al., "Cost-effective outbreak detection in networks," in *Proc. 13th KDD*, San Jose, CA, USA, 2007, pp. 420-429.

[14] S. Kakutani, "A generalization of brouwer's fixed point theorem," *Duke Mathematical Journal*, vol. 8, no. 3, pp. 457-459, 1941, doi: 10.1215/s0012-7094-41-00838-4.

[15] J. C. P. Bus, "Convergence of Newton-like methods for solving systems of nonlinear equations," *Numerische Mathematik*, vol. 27, no. 3, pp. 271-281, 1976, doi: 10.1007/bf01396177.

[16] C. Grosan and A. Abraham, "Multiple solutions for a system of nonlinear equations," *Int. J. Innov. Comput. Inf. Control*, vol. 4, no. 9, pp. 2161-2170, 2008.

[17] J. Goldenberg, B. Libai, and E. Muller, "Talk of the network: a complex systems look at the underlying process of word-of-mouth," *Marketing Lett.*, vol. 12, no. 3, pp. 211-223, 2001.

[18] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks," in *ECML-PKDD*, Berlin, Germany, 2006, pp. 259-271.

[19] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "Celf++: optimizing the greedy algorithm for influence maximization in social networks," in *Proc. 20th WWW*, Hyderabad, India, 2011, pp. 47-48.

[20] C. Borgs et al., "Maximizing social influence in nearly optimal time," in *Proc. 25th ACM-SIAM Symp. Discrete Algorithms*, Porland, Oregon, USA, 2014, pp. 946-957.

[21] J. Tang, X. Tang, and J. Yuan, "Influence maximization meets efficiency and effectiveness: a hop-based approach," in *Proc. IEEE/ACM ASONAM*, Sydney, Australia, 2017, pp. 64-71.

[22] J. Tang, X. Tang, and J. Yuan, "An efficient and effective hop-based approach for influence maximization in social networks," *SNAM*, vol. 8, no. 1, pp. 10, 2018, doi: 10.1007/s13278-018-0489-y.
[23] W. Yang, L. Brenner, and A. Giua, "Computation of activation probabilities in the independent cascade model," in *5th CoDIT*, Thessaloniki, Greece, 2018, pp. 791-797.
[24] F. Ye et al., "Identifying influential individuals on large-scale social networks: a community based approach," *IEEE Access*, vol. 6, pp. 47240-47257, 2018, doi: 10.1109/access.2018.2866981.
[25] Q. Yu, H. Li, Y. Liao, and S. Cui, "Fast budgeted influence maximization over multi-action event logs," *IEEE Access*, vol. 6, pp. 14367-14378, 2018, doi: 10.1109/access.2018.2809547.
[26] J. S. Coleman, *Introduction to Mathematical Sociology*, London, Collier-Macmillan, 1964.
[27] V. Colizza, R. Pastor-Satorras, and A. Vespignani, "Reaction-diffusion processes and metapopulation models in heterogeneous networks," *Nature Physics*, vol. 3 no. 4, pp. 276, 2007, doi: 10.1038/nphys560.
[28] X. Deng, Y. Dou, T. Lv, and Q. V. H. Nguyen, "A novel centrality cascading based edge parameter evaluation method for robust influence maximization," *IEEE Access*, vol. 5, pp. 22119-22131, 2017, doi: 10.1109/access.2017.2764750.
[29] H. Wu et al., "LAIM: a linear time iterative approach for efficient influence maximization in large-scale networks," *IEEE Access*, vol. 6, pp. 44221-44234, 2018, doi: 10.1109/access.2018.2864240.