A fast algorithm for finding most influential people based on the linear threshold model

Khadijeh Rahimkhani
University of Tehran
3 PUBLICATIONS **113** CITATIONS
SEE PROFILE

Abolfazl Aleahmad
University of Tehran
37 PUBLICATIONS **628** CITATIONS
SEE PROFILE

Maseud Rahgozar
University of Tehran
153 PUBLICATIONS **789** CITATIONS
SEE PROFILE

Ali Moeini
University of Tehran
126 PUBLICATIONS **984** CITATIONS
SEE PROFILE

Expert Systems with Applications 42 (2015) 1353–1361

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A fast algorithm for finding most influential people based on the linear threshold model

Khadije Rahimkhani [a,1], Abolfazl Aleahmad [a,1,*], Maseud Rahgozar [a], Ali Moeini [b]

[a] Database Research Group, Control and Intelligent Processing Center of Excellence, School of Electrical and Computer Engineering, University of Tehran, Iran
[b] Faculty of Engineering Science, School of Engineering, University of Tehran, Iran

ARTICLE INFO

Article history:
Available online 30 September 2014

Keywords:
Social networks
Influential people retrieval
Influence maximization
Linear threshold model

ABSTRACT

Finding the most influential people is an NP-hard problem that has attracted many researchers in the field of social networks. The problem is also known as influence maximization and aims to find a number of people that are able to maximize the spread of influence through a target social network. In this paper, a new algorithm based on the linear threshold model of influence maximization is proposed. The main benefit of the algorithm is that it reduces the number of investigated nodes without loss of quality to decrease its execution time. Our experimental results based on two well-known datasets show that the proposed algorithm is much faster and at the same time more efficient than the state of the art algorithms.

1. Introduction

Social networks play an important role in our new world. After invention of online social networks, people are able to influence each other much more easily. This fact caused many researchers to look for an efficient method for finding top-k most influential people through social networks. This problem is also known as influence maximization (IM) and has many applications such as: opinion propagation, studying acceptance of political movements or acceptance of technology in economics. For example, suppose that we need to advertise a product in a country or we need to propagate news. For this purpose, we need to choose some people as a starting point and maximize the news or the products influence in the target society.

Finding most influential people has been found useful for many applications such as: developing recommender systems (Morid, Shajari, & Hashemi, 2014; Song, Tseng, Lin, & Sun, 2006), choosing useful weblogs (Leskovec et al., 2007) and finding influential twitters (Bakshy, Hofman, Mason, & Watts, 2011; Weng, Lim, Jiang, & He, 2010). The problem was introduced in (Domingos & Richardson, 2001) for the first time. After that in (Kempe, Kleinberg, & Tardos, 2003) the authors formalized the problem as follows: given a weighted graph in which nodes are people and edge weights represent influence of the people on each other, it is desired to find K starting nodes that their activation leads to maximum propagation based on a chosen influence maximization model.

Inspired from humanities science, two well-known influence maximization models were presented for the first time in (Kempe et al., 2003): Linear Threshold (LT) and Independent Cascade (IC). Also, different algorithms have been proposed based on IC model (Morid et al., 2014; Wang, Cong, Song, & Xie, 2010; Zhang, Zhou, & Cheng, 2011) and LT model (Brin & Page, 1998; Chen et al. 2010; Goyal, Bonchi, & Lakshmanan, 2011; Leskovec et al., 2007). In this paper, a new algorithm will be introduced for finding top-k most influential people based on the LT model. Also, through experimental results on two real world datasets, we will show that the proposed algorithm is faster and even more efficient than the state of the art algorithms, so it would be applicable to larger social networks.

In this paper, we focus on selecting the most important nodes that lead to maximum influence based on the LT model. For this purpose, community structures of the input graph are identified first and the most influential communities are selected among them; then a number of representative nodes are chosen from the resulted communities and form the final output of the proposed algorithm. Our experiments show that the proposed algorithm is very efficient in finding the most influential nodes with minimum execution time. In other words, in this paper we will focus on optimizing both the influence spread and the execution time.

* Corresponding author.
E-mail addresses: rahimkhani_khadije@yahoo.com (K. Rahimkhani), a.aleahmad@ece.ut.ac.ir (A. Aleahmad), Rahgozar@ut.ac.ir (M. Rahgozar), moeini@ut.ac.ir (A. Moeini).

[1] The first two authors contributed equally to this work.

http://dx.doi.org/10.1016/j.eswa.2014.09.037
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

The rest of the paper is organized as follows: the next section reviews the related works; the proposed algorithm is presented in Section 3; comparison and analysis of the results are presented in Section 4 and finally, Section 5 concludes the paper.

2. Related works

As stated in the previous section, two different information diffusion processes are presented in (Kempe et al., 2003): IC and LT models. In IC model, when a node is activated in time t, it has only one chance to activate its neighbors in time $t+1$. In LT model, in-links of each node are weighted with a value less than one and each person has an activation threshold between zero and one. Then, if total influence rate of neighboring nodes of a node is greater than its threshold, the node will be activated.

The influence function is monotone and sub-modular in both models. The function set f: $2^U \rightarrow R^{+}$ is monotone if $f(S) \leqslant f(T)$ for all $S \subseteq T \subseteq U$. Also, if $f(S \cup \{w\}) - f(S) \geqslant f(T \cup \{w\}) - f(T)$ is true for all $S \subseteq T$ and $w \in U \setminus T$, then the function is sub-modular. So, both IT and IC models of influence maximization are NP-hard problems. Furthermore, in (Lu, Zhang, Wu, Kim, & Fu, 2012) the authors study the complexity of the influence maximization problem in deterministic linear threshold model. They show that there is no $n^{1-\epsilon}$ – factor polynomial time approximation for the problem in the deterministic linear threshold model unless P = NP. Also, they show that the exact computation of the influence given a seed set can be solved in polynomial time.

The rest of this section reviews LT and IC based influence maximization algorithms. Then the most important community detection algorithms are discussed and finally a formalization of linear threshold model will be presented.

2.1. IC-based influence maximization algorithms

There exist different solutions for Independent Cascade (IC) model of the problem. In (Guo, Zhang, Zhou, Cao, & Guo, 2013), the authors present a dynamic selection approach referred to as the Item-based top-k influential user Discovering Approach (IDA). IDA identifies the top-k influential users for a given topic based on real-world diffusion traces and on-line relationships. In (Kim, Kim, & Yu, 2013), the authors propose a scalable influence approximation algorithm named Independent Path Algorithm (IPA). In (Li et al., 2014), the polarity-related influence maximization (PRIM) problem is proposed which aims to find the seed node set with maximum positive influence or maximum negative influence in signed social networks. In (Liu et al., 2013), the authors propose GS algorithm for quick approximation of influence spread by solving a linear system, based on the fact that propagation probabilities in real-world social networks are usually quite small. Furthermore, for better approximation, they study the structural defect problem existing in networks, and correspondingly, propose enhanced algorithms, GSbyStep and SSSbyStep, by incorporating the Maximum Influence Path heuristic. In (Yang et al., 2012), the authors propose to measure the seed's independent influence by a linear social influence model. In (Li et al., 2014), a novel conformity aware cascade model is proposed which leverages on the interplay between influence and conformity in obtaining the influence probabilities of nodes from underlying data for estimating influence spreads. Also, in (Ohsaka, Akiba, Yoshida, & Kawarabayashi, 2014), the authors propose a new method that produces solutions of high quality base on Monte Carlo simulation.

2.2. LT-based influence maximization algorithms

In (Leskovec et al., 2007), the authors introduced a sub-modularity factor and presented a lazy-forward greedy algorithm named CELF. Then Goyal et al. presented CELF++ that is an extension of CELF which reduces the number of iterations of the algorithm considerably (Goyal, Lu, & Lakshmanan, 2011a).

There exists different works on influence models such as: (Chen et al. 2010; Goyal, Lu, & Lakshmanan, 2011b; Kimura & Saito, 2006). Simpath algorithm (Goyal et al., 2011b) estimates activation probability of nodes by investigating paths that exists between seed nodes and other nodes in the input network. Kempe et al. used a greedy algorithm to add K nodes with maximum marginal gain to the seeds set (Kempe et al., 2003). Exact calculation of marginal nodes is #P-hard in both IC and LT models (Bakshy et al., 2011; Chen et al., 2010). Hence, they are estimated by use of Monte Carlo (MC) simulation. Unfortunately, the greedy algorithm has the following drawbacks:

- MC simulation should be run for many times (e.g. 10,000 times).
- This greedy algorithm should use MC for $n \times k$ times. Where n is the number of nodes and k is the number of selected nodes.

Chen et al. came to the conclusion that based on LT models, influence maximization problem is #P-hard (Chen et al., 2010) and it can be executed on directed acyclic graphs (DAGs) in linear time. They supposed that each node can only influence a limited number of its neighbors. So, a local DAG (LDAG) is considered for each node and its influence is investigated for its LDAG. They showed that this heuristic algorithm is faster and even more efficient than its greedy counterparts. However, their idea has the following limitations:

- Finding LDAGs is an NP-hard problem, so a greedy heuristic is employed to discover a good LDAG in (Chen et al., 2010). Using a greedy LDAG may introduce loss in quality of the seed set.
- It is supposed that a node can influence others through the paths that exist in its LDAG, so its influence on other paths is ignored.
- Storage of all LDAGs needs a huge memory in large networks.

This algorithm is very efficient for LT model of influence maximization. It is faster than greedy algorithms like CELF but it does not propose high quality starting nodes. Another algorithm is presented in (Leskovec et al., 2007) that considers another condition for simple greedy algorithms. They presented an optimization for CELF that reduces the number of MC iterations considerably. Despite this optimization, the greedy algorithm needs nearly 700 iterations (Leskovec et al., 2007). So, this algorithm is still slow and cannot be used for large graphs (Goyal et al., 2011). Generally, LDAG-based algorithms perform better than optimized CELF algorithms (Chen et al., 2010).

Simpath is also a lazy-forward optimized version of CELF (Goyal et al., 2011b). The algorithm inspects all paths from the initial seeds to maximize influence in the LT model. The problem of computing a simple path is #P-hard (Valiant, 1979). Simpath uses the vertex cover method to reduce the number of iterations of the algorithm and the resulted nodes are inspected for influence maximization. In this method, the number of selected nodes is increased after each iteration of the algorithm, so the influence maximization process becomes very slow. Therefore, further optimization is applied to reduce the run time of the iterations.

More recently, in (Zhou, Zhang, Guo, & Guo, 2014), the authors derive an upper bound for the spread function under the LT model. They propose an efficient UBLF algorithm by incorporating the bound into CELF. Experimental results demonstrate that UBLF, compared with CELF, reduces Monte Carlo simulations and reduces the execution time when the size of seed set is small. In (Narayanam & Narahari, 2010), the authors proposed a Shapley value based heuristic SPIN for the LT model. SPIN evaluates the influence spreads of seeds set but it does not use important features of the LT model.

CELF and CELF++ algorithms have high execution time, LDAG and High-Degree algorithms could not provide an optimized set of seeds. In contrast to greedy algorithms, the quality of influence spread of Simpath is not optimized in more than 63 percent of its seeds set (Li et al., 2014). Hence, we provide a way to select nodes that have both good quality and reduced execution time.

2.3. Community detection algorithms

Generally there exist two categories of community detection algorithms: overlapping and non-overlapping. As overlapping community detection is a main part of our proposed algorithm, the related works will be explored in this section briefly. Overlapping community detection algorithms can be divided into five different categories:

– Clique percolation
– Line graph
– Link partitioning
– Local expansion and optimization
– Agent-based and dynamic algorithms

Among the categories, click percolation algorithms are the most efficient that try to find strongly connected sub-graphs to estimate the actual communities. Link partitioning algorithms try to divide the input sub-graph into some sub-graphs iteratively in such a way that the final sub-graphs present the graph communities. Local expansion and optimization algorithms work based on extending natural communities or a partial community (Lancichinetti, Fortunato, & Kertesz, 2009). Most of these methods focus on a local profit function that tries to find the nodes that are densely connected to each other. Label propagation algorithms (Raghavan, Albert, & Kumara, 2007) distribute different labels on the input graph based on some predefined roles and finally, the nodes with the same label are considered as a community.

In this paper, we use SLPA algorithm (Xie, Szymanski, & Liu, 2011) for community detection. SLPA simulates information diffusion process based on listener-speaker interaction. It propagates labels between nodes based on some rules that govern the interaction. In contrast to the methods of in Raghavan et al. (2007) and Gregory (2010) they in which a node forgets the information that it received from the previous steps, SLPA considers a memory for each node to store the labels that it receives in different steps of the algorithm. Membership degree of a node is considered as the probability of observing a community label in a node's memory. SLPA is very fast and at the same time it does not need to know the number of communities.

2.4. Linear threshold model

Consider the graph $G = (V, E, b)$ in which $(u, v) \in E$ is an edge and each edge has a weight $b_{u,v}$ that shows the influence of u on v. in (Kempe et al., 2003) they proved that the Linear Threshold (LT) model is equal to Live-Edge model, if each node $v \in V$ chooses at most one edge with a probability that is equal to the weight of the edge. If an edge is selected, it is live else it is dead. Then a diffusion process is carried out on the live edges. The influence of a node is equal to the number of nodes that are accessible from the node. Let X be the accessible space and let $\delta_X(S)$ be the total number of nodes that are accessible from members of S; then influence of S is calculated as follows:

$$\delta(S) = \sum_X \Pr[X].\delta_X(S) \tag{1}$$

Also, $\delta_X(S)$ is calculated as follows:

$$\delta_X(S) = \sum_{v \in V} I(S, v, X) \tag{2}$$

Where $I(S,v,X)$ is an index function that is equal to 1 if a live path exists between all nodes of S and v, else it is equal to 0. So, we have:

$$\delta(S) = \sum_{v \in V} \sum_X \Pr[X] \cdot I(S, v, X) = \sum_{v \in V} \gamma_{S,v} \tag{3}$$

In which, $Y_{s,v}$ is the activation probability of node v if S is considered as the starting nodes set that shows the influence weight of S on v.

In this model, only directed paths are considered. Let $P = \langle v_1, v_2, \ldots, v_m \rangle$ be a path and $(v_i, v_j) \in P$ be an edge of the path. The activation probability of a path P is :

$$\Pr[P] = \prod_{(v_i, v_j) \in P} b_{v_i, v_j}$$

Therefore:

$$\gamma_{u,v} = \sum \prod_{P \in P(u,v)} \Pr[P] \tag{4}$$

As an example, the influence of node x on node z is $\gamma_{x,y} = 0.3 * 0.2 + 0.4 = 0.46$ in Fig. 1.

Finding an optimized path with k members is a NP-Complete problem. So, we should look for a method that provides a good estimation of the solution within an acceptable period of time.

Let us show sub-graphs with superscripts in the following formulas. So, let $\gamma_{u,v}^{V-S}$ be the total influence of node u on node v in the sub-graph $V - S$. So, $V - S + u$ means $((V/S) \cup \{u\})$. ComPath works based on the following theory in (Goyal et al., 2011b):

Theory 1. *In the LT model, the influence spread of a node depends on the influence spread of its out-neighbors linearly:*

$$\delta(v) = 1 + \sum_{u \in N^{out}(v)} b_{v,u}.\delta^{V-v}(u) \tag{5}$$

Where, $N^{out}(v)$ is the set of out-neighbors of v. Based on this theory, the influence spread of x in Fig. 1 is calculated as follows:

$$\begin{aligned} \delta(x) &= 1 + b_{x,y}.\delta^{V-x}(y) + b_{x,z}.\delta^{V-x}(z) \\ &= 1 + 0.3.(1 + 0.2) + 0.4.(1 + 0.5) = 1.96 \end{aligned}$$

This theory is proven in (Goyal et al., 2011b). Based on this theory, $\gamma_{v,y}$ can be rewritten as follows:

$$\gamma_{v,y} = \sum_{u \in N^{out}(v)} b_{v,u}.\gamma_{u,y}^{V-v} \tag{6}$$

3. The proposed algorithm

The main steps of the proposed algorithm are summarized in Fig. 2. First, community structures of the input graph are used to form a new network. Each node of the new network represents a community. Then the most central nodes of the new network are detected based on the betweenness centrality measure. Each node of the new network contains the nodes of its corresponding community. So, proportional to the size of the community, a number of nodes are selected from each community based on degree

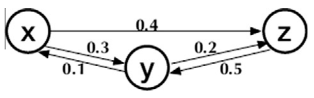


Fig. 1. A sample network (Goyal et al., 2011b).

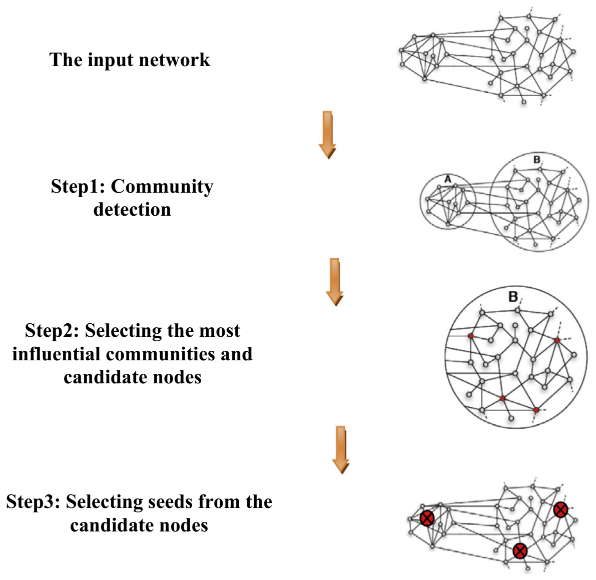

The input network
Step1: Community detection
A
B
Step2: Selecting the most influential communities and candidate nodes
B
Step3: Selecting seeds from the candidate nodes

Fig. 2. The main steps of the proposed algorithm: In step1, the input network is reduced into a new small network by use of a community detection algorithm, in step 2, some candidate nodes are .selected from each community and in step3, final seeds are selected from the candidate nodes.

centrality to from candidates set. Then top-k most influential nodes are chosen from the candidates set based on the paths that exist in the network. Details of the algorithm are depicted in Fig. 3 and will be discussed in the following subsections.

Also, the following figure summarizes the steps of the proposed algorithm, named *ComPath*:

The inputs of the algorithm are: path length L and the input network N. Details of the algorithm will be discussed step by step in the following subsections.

3.1. Step 1: creating a new network

In the first step of the algorithm, community structures of the input network are identified and then a reduced network is built as follows: first SLPA algorithm (Xie et al., 2011) is used to detect communities of the input network then each community is considered as a node in a new graph $G = (V,E)$. Where V is the set of nodes (communities) and E is the directed edges that exist between the graph nodes. So, a directed edge (A, B) shows that members of the community A are linked to members of B in the original network (see Fig. 4).

3.2. Step 2: detecting nodes with maximum influence

After construction of the communities graph in the previous section, the graph nodes are sorted based on their betweenness centrality measure. This measure shows the importance of each node in the input network. A node has a high betweenness centrality value if it is situated between many other nodes of the input graph. In this way, those communities that can potentially extend communication with others are selected.

After choosing important communities, some representative nodes should be selected from each of them. Degree centrality is a simple and efficient measure for this purpose. High degree centrality value of a node means it has more communication links to the other nodes. Now, a question may arise: how many nodes should be chosen from each community? A community that contains more nodes should play a more important role in the final result. The following formula is applied for this purpose:

$$\left\lceil \left(\frac{N_i}{Max - Min}\right) * \beta \right\rceil + \alpha \qquad (7)$$

In which *Max* is the number of nodes of the largest community, *Min* is the number of nodes of the smallest community and N_i is the number of nodes of the ith community. The value of $(\frac{Ni}{Max-Min})$ falls between zero and one, so it is multiplied by a constant β to amplify its effect. Also, α is a constant that ensures the selection of at least α nodes from each community. The appropriate value of α depends on the input network. Our experiments on different datasets show that $\beta = 10$ and $\alpha = 2$ are suitable for most usual networks.

Let *Selected_Nodes* be the set of selected nodes that are candidates for the final output of our algorithm. Larger seeds set needs investigation of more communities that leads to a larger *Selected_Nodes*. The last step of the algorithm is devoted to choosing k nodes from the *Selected_Nodes* set.

3.3. Step 3: finding nodes with maximum influence

This step selects k nodes from the output of the previous step in such a way that their activation leads to maximum influence in the input network. In order to find the final set of seed nodes, all paths from each node of the network to the nodes in *Selected_Nodes* are inspected and the nodes with maximum influence are obtained. First, the path length is considered equal to 2 and seed nodes with maximum influence are selected. Then the path length is increased and the seeds set is calculated again. If average influence of the seed nodes is increased compared to the previous path length, then the path length is increased and the seeds set is calculated again.

After increasing the path length, some new nodes may be found that are not present in the *Selected_Nodes* set. So, these nodes are added to a new set v' that will be inspected later. The new set's size depends on *Selected_Nodes* and the chosen path length. So, v' set is defined as follows:

$$V' = (Selected\_Nodes) \cup \left\{N^{out}(Selected\_Nodes)\right\}^{L}, \quad 1 \leqslant L \leqslant n \qquad (8)$$

Where L is the path length, N^{out} is the set of neighbors of a node that are not present in *Selected_Nodes* set. If i number of seeds are found till the ith step then all seeds of the ith step are removed from v' to find a seed in the $i+1$th step. Then influence of all $u \in Selected\_Nodes–S_i$ that are present in paths with different length is computed (where S_i is the set of selected nodes till the ith step). As an example consider the network of Fig. 1 and let $S_1 = \{x\}$. In order to find the second node, x is removed from S_1 and if the two remaining nodes are in *Selected_Nodes*, their influence will *be* $\gamma_{y,z} = 0.2$ *and* $\gamma_{z,y} = 0.5$. Then if the path length is equal to one, z will be selected as the next seed.

For calculating top-k nodes in the 3rd step of the algorithm, the following *Backtrack* algorithm will be computed for each node that was proposed by the previous step. The *Backtrack* algorithm is a lazy-forward algorithm that is presented in (Kempe et al., 2003) for path finding. The three inputs of the algorithm are: u, a node from *Selected_Nodes* set, w, a part of the input network that is under investigation and L that is the path length.

Table 1
Characteristics of Epinion and NetHEPT datasets.

	Epinions	NetHEPT
Number of nodes	76 K	15 K
Number of edges	509 K	58 K
Average degree	13.4	4.12
Maximum out-degree	3079	64
No. of connected components	11	1781
Size of the largest component	76 K	6794
Average component size	6.9 K	8.6

Table 2
The algorithms used in our experiments.

Algorithm	Description
Maximum degree (Kempe et al., 2003)	A heuristic based on the degree centrality measure. Seeds set contains top-k nodes with maximum degree
Page rank (Brin & Page, 1998)	This algorithm is commonly used for ranking webpages based on their importance in the web graph. This algorithm is ran with parameter $p = 0.85$ and $\varepsilon = 0.001$ and then top-k nodes of the ranking are selected
LDAG (Chen et al., 2010)	This algorithm is ran on the input graph with the parameter $\theta = 1.32$ to control creation of local DAG for each node
MC-CELF (Leskovec et al., 2007)	A greedy algorithm that is actually an optimized version of CELF. In this algorithm, Monte Carlo simulation is run for 10,000 times to obtain the final seeds
CELF++ (Goyal et al., 2011a)	This algorithm is also a faster and more efficient version of CELF

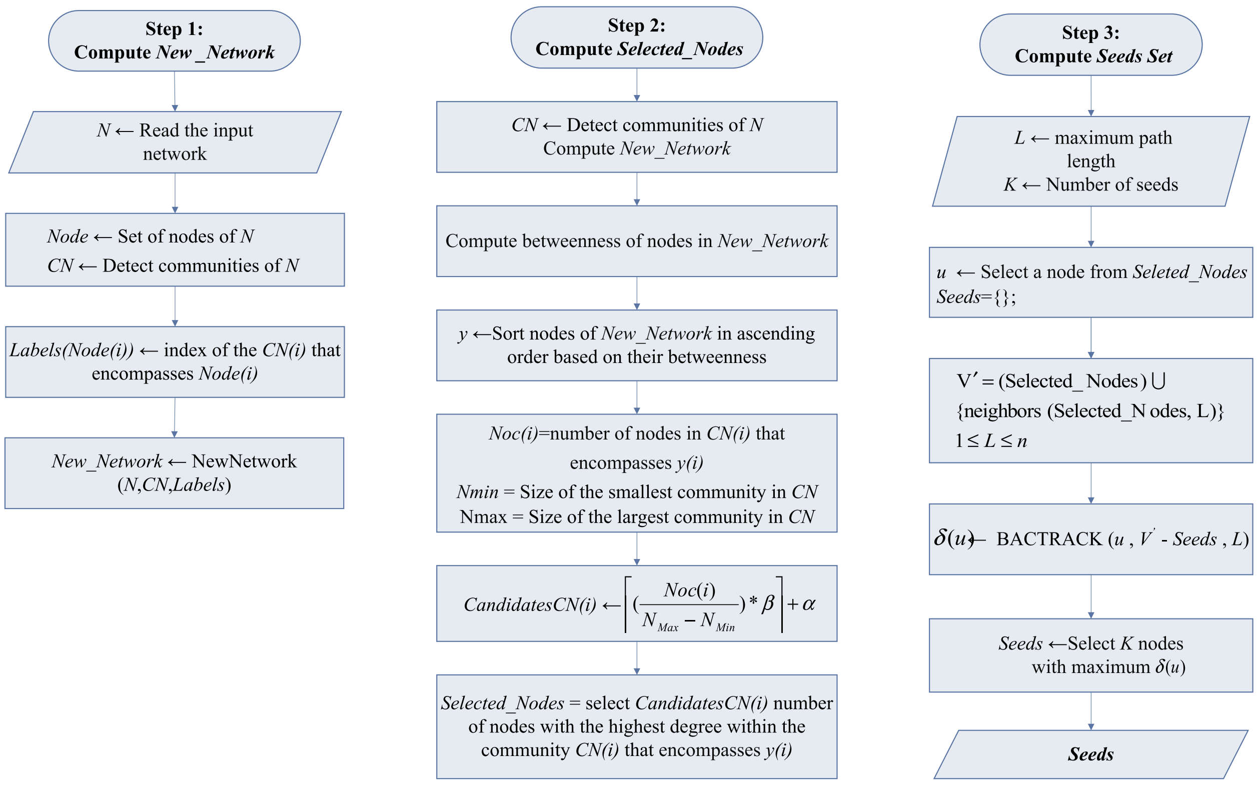

Step 1: Compute New_Network
N ← Read the input network
Node ← Set of nodes of N
CN ← Detect communities of N
Labels(Node(i)) ← index of the CN(i) that encompasses Node(i)
New_Network ← NewNetwork (N,CN,Labels)
Step 2: Compute Selected_Nodes
CN ← Detect communities of N
Compute New_Network
Compute betweenness of nodes in New_Network
y ←Sort nodes of New_Network in ascending order based on their betweenness
Noc(i)=number of nodes in CN(i) that encompasses y(i)
Nmin = Size of the smallest community in CN
Nmax = Size of the largest community in CN
$CandidatesCN(i) \leftarrow \left\lceil \left(\frac{Noc(i)}{N_{Max} - N_{Min}}\right) * \beta \right\rceil + \alpha$
Selected_Nodes = select CandidatesCN(i) number of nodes with the highest degree within the community CN(i) that encompasses y(i)
Step 3: Compute Seeds Set
L ← maximum path length
K ← Number of seeds
u ← Select a node from Seleted_Nodes
Seeds={};
V′ = (Selected_Nodes) ∪ {neighbors (Selected_N odes, L)}
1 ≤ L ≤ n
δ(u)← BACTRACK (u , V′ - Seeds , L)
Seeds ←Select K nodes with maximum δ(u)
Seeds

Fig. 3. Steps of the proposed algorithm.

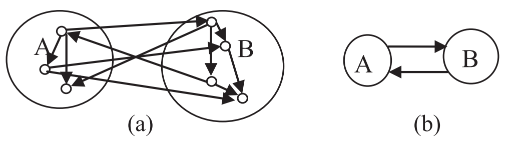


Fig. 4. (a) A sample input graph with two communities and (b) its corresponding reduced graph.

Algorithm 1. Pseudo code of *Backtrack* algorithm

```
BACKTRACK(u, W, L)
1:  Q ← {u}; spd ← 1; pp ← 1; D ← null;
2:  While Q ≠ ∅ DO
3:     [Q, D, spd, pp] ← Forward (Q, D, spd, pp, W, L).
4:     u ← Q.last(); Q ← Q-u; delete D[u]; v ← Q.last().
5:     pp ← pp/$b_{v,u}$.
6:  return spd
```

All paths starting from u are identified in this algorithm. For this purpose, *Backtrack* uses the *Forward* algorithm that is designed to find all paths of length L. During the path finding process, weight of edges between different nodes in the path are multiplied to find the value of $\gamma_{u,v}$, using formula 3.

Algorithm 2. Pseudocode of Forward algorithm

```
Forward(Q ,D, spd, pp , W, Len)
1:  x = Q.last(). L ← 0;
2:     While ∃ y ∈ $N^{out}$(x), L⩽Len : y ∉ Q , y ∉ D[x], y ∈ W DO
3:     Q.add(y);
4:     pp ← pp.$b_{x,y}$; spd ← spd + pp.
5:     D[x].insert(y); x ← Q.last().
6:     L ← L + 1;
7:  return [Q, D, spd , pp].
```

The inputs of the *Forward* algorithm are: Q is a stack that contains nodes of the found path, *pp* is the influence weight of the path, *spd* is the influence rate of *u* in the input sub-graph *W*, and *L* is the path length. Also, *D[x]* is the set of neighbors of *x* that have an input link from x and have not been investigated yet. When a node is removed from the stack, *pp* is updated. Then $b_{v,u}$ will be the weight of the edge between u and v.

Please note that the original *Forward* algorithm of (Goyal et al., 2011b) is changed here to reduce its execution time. The path length is reduced as much as possible to decrease the number of investigated nodes. In *Forward* algorithm, all neighbors of x are added to the stack if:

– They are not investigated yet ($y \notin D[x]$)
– They contain no cycle ($y \notin Q$)
– They are present in the sub-graph W ($y \in W$)
– Their path to x is less than *Len* ($L \leqslant Len$)

Then *pp* and *spd* variables are updated. This process is continued till the maximum path length is achieved. After this step the influence of the nodes in the proposed set is computed. *spd* contains the value of $\gamma_{u,v}$ between u and v in formula 3. The node with maximum influence is selected as a member of the seeds set.

Analysis of this step of the algorithm shows that it is a greedy algorithm that tries to select the node x with maximum $\delta(x)$. In order to find the ith seed, if x is a candidate node, all $i-1$ number of already selected seeds are eliminated from *v'* and the influence function is computed for the remaining nodes:

$$\delta(v) = 1 + \sum_{u \in N^{out}(v)} b_{v,u}.\delta^{V'-S_i-v}(u) = \sum_{u \in V'} \gamma_{v,u}^{V'-S_i}, \quad v \in Selected\_Nodes \tag{9}$$

In other words, $\delta(x)$ is the influence of all nodes in the *Selected_Nodes* set on other nodes of *v'*. Finally the node x with the maximum influence Max ($\delta(x)$) on other nodes is chosen as a seed for the next step:3

$$\delta(x) = Max\{\delta(x_1), \delta(x_2), \ldots, \delta(x_n)\}, \quad x_i \in Selected\_Nodes \tag{10}$$

$$S_{i+1} = S_i \cup x \tag{11}$$

Therefore, it is needed to call the Backtrack function (algorithm 1) for each node in the *Selected_Nodes* set to obtain the final seed nodes. Each node with maximum influence spread is detected after running the Backtrack function for n times (n = |*Selected_Nodes*|). Then this node is omitted from the *Selected_Nodes* set. This process runs for k times to obtain the final seeds set (k = |*seeds*|).

4. Evaluation and experimental results

In this section, the proposed algorithm is evaluated based on two well-known datasets and results are compared with 5 other state of the art algorithms.

4.1. Datasets

Two real networks are used in our experiments (see Table 1):

– NetHEPT: a well-known dataset that has been used in many studies (Chen et al., 2010; Chen, Wang, & Yang, 2009; Chen et al., 2010; ; Goyal et al., 2011b; Kempe et al., 2003). It is created by use of *"High Energy Physics (Theory)"* part of *arXiv3* which is the collaboration network of paper authors. The users' actions is publishing papers and *A(u,v)* is the number of papers that u and v have published together.
– Epinion (Richardson, Agrawal, & Domingos, 2003): In this network, nodes are members of a site and an edge from v to u, means v trusts u (i.e. u influences v).

In order to calculate edges weight ($b_{u,v}$) for NetHEPT graph, we use the same method that were used in (Goyal, Bonchi, & Lakshmanan, 2010; Goyal et al., 2011b; Kempe et al., 2003):

$$b_{u,v} = A(u,v)/N(v) \tag{12}$$

In which, *A(u,v)* is the mutual actions of u and v and *N(v)* is a normalization factor that is calculated as follows:

$$N(v) = \sum_{u \in N^{in}(v)} A(u,v) \tag{13}$$

Where $N^{in}(v)$ is the set of nodes that has an edge to v. Also, the influence weight of edges to v is considered as $1/M(v)$.

4.2. Comparison and analysis of the results

There exist some well-known algorithms that will be compared with our proposed algorithm in this sub-section. The algorithms are listed in Table 2:

At first, the optimum number of chosen communities and candidate nodes that are selected from each of them should be identified. For this purpose different experiments are carried out. More than 2800 communities are identified for NetHEPT by SLPA algorithm (Xie et al., 2011). The number of candidate nodes from each community depends on the number of selected communities and the number of nodes in each community. Fig. 5 shows the number of communities and the number of nodes selected from 255 communities with maximum betweenness centrality in the network. We need to select 50 nodes as the final seeds set. As the figure shows, 12 nodes are selected from the community with highest betweenness centrality measure.

Also, the number of communities and size of *Selected_Nodes* in NetHEPT and Epinion are shown in Table 3 and 4 respectively. As Table 3 shows, if 5, 50 or 100 seeds are needed, only 39, 437, 870 nodes are inspected in the proposed algorithm while the total number of nodes of the network is 15233. Also, for Epinion dataset, a few numbers of communities are investigated that helps our algorithm runs much faster in the next step.

Different number of nodes will be activated for different values of path length. Table 5 summarizes the algorithms results on NetHEPT dataset for path length of 3, 4 and 5 and seeds set size of 50 and 100.

Our experiments for path length greater than 5 showed negligible improvement. So, the experiments are reported for the path length of 5.

Figs. 6 and 7, compare ComPath with other algorithms for different size of seeds set based on the two datasets. It can be observed that the proposed method performed better than the others.

Because of very high execution time of the algorithms on NetHEPT, their execution is not possible for 100 seeds and no results are reported for this number of seeds. The following figures compare execution time of the algorithms:

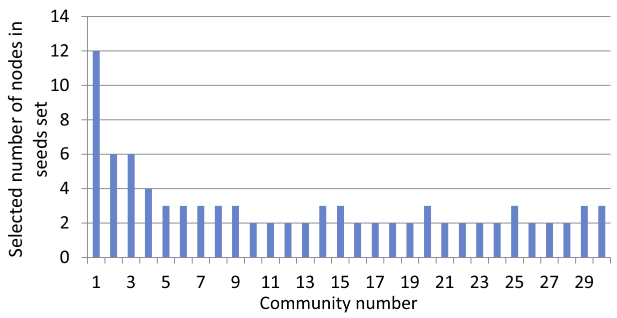


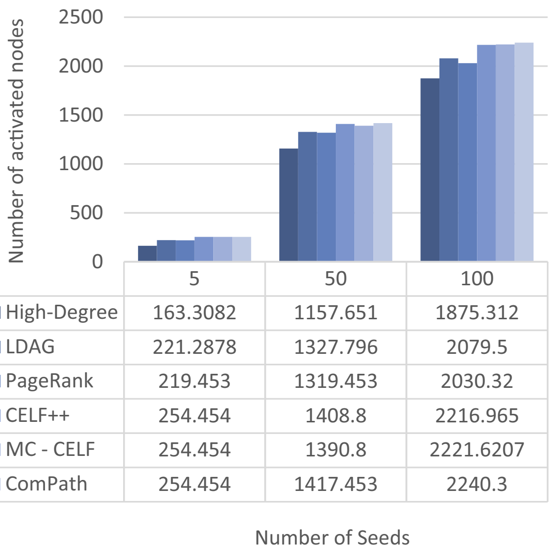
Fig. 5. The number of candidate nodes selected from each community for $k = 50$. Since all communities in rank 30th or higher have 2 representative nodes in the final seeds set, they are not presented in this figure.

Table 3
Communities size and no. of selected nodes in experiments on NetHEPT.

Size of *Selected_Nodes*	No. of communities with maximum betweenness	Size of seeds set
39	7	5
437	225	50
870	500	100

Table 4
Communities Size and No. of Selected nodes in experiments on Epinion.

Size of *Selected_Nodes*	No. of communities with maximum betweenness	Size of seeds set
6	1	5
165	1	50



	5	50	100
■ High-Degree	163.3082	1157.651	1875.312
■ LDAG	221.2878	1327.796	2079.5
■ PageRank	219.453	1319.453	2030.32
■ CELF++	254.454	1408.8	2216.965
■ MC - CELF	254.454	1390.8	2221.6207
■ ComPath	254.454	1417.453	2240.3

Fig. 6. Comparison of ComPath with other algorithms on NetHEPT dataset.

As the figures show, the execution time of ComPath is much less than the others. The following table compares execution time improvement of ComPath compared to the other algorithms:

The experiments show that ComPath not only selects better initial seeds but also it is much faster than the other comparable state of the art algorithms. LDAG is comparable with ComPath in terms of execution time but it is not accurate in selecting high quality initial seeds, especially if a large number of initial seeds are needed (See Figs. 8 and 9).

4.3. Comparison of the algorithms

Our experimental results show that CELF++ and MC-CELF are very time-consuming. The problem of the CELF algorithm is the number of times it uses Monte Carlo simulation which makes it very slow in terms of execution time. Both Simpath and CELF++ algorithms calculate the following equation in each step:

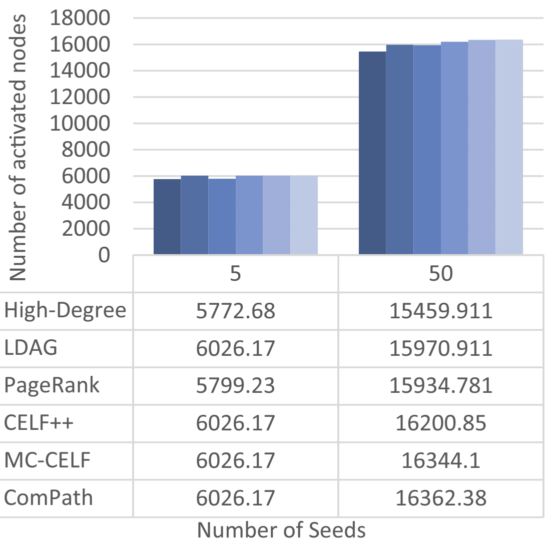
$$\delta(S_i + x + u) = \delta^{V-S_i-u}(x) + \delta^{V-x}(S_i + u)$$

Where, S_i is the seeds set after the ith step ($i = |S_i|$). In order to identify the $i+1$th seed, Simpath calculates both $\delta(S_i + x)$ and $\delta^{V-x}(S_i + u)$ simultaneously and if x is chosen as a seed then only $\delta^{V-S_i-u}(x)$ is needed to be calculated. But in CELF++ x is considered unknown in each iteration of the algorithm and all the calculations are carried out multiple times.

In this research, we tried to identify the initial nodes among the members of the *Selected_Nodes* set. While CELF and CELF++ carry out the calculations among all nodes of the input graph. Also, Simpath uses Vertex Cover Optimization for the first initial node and Look Ahead Optimization for the rest of them that leads to reducing the number of Backtrack iterations (Goyal et al., 2011b). Simpath uses a threshold η to balance a trade-off between influence spread and execution time; e.g. smaller η means larger influence spread and larger execution time. This parameter specifies which paths should be investigated in the algorithm. In other words, if the influence weight between two nodes is less than the η

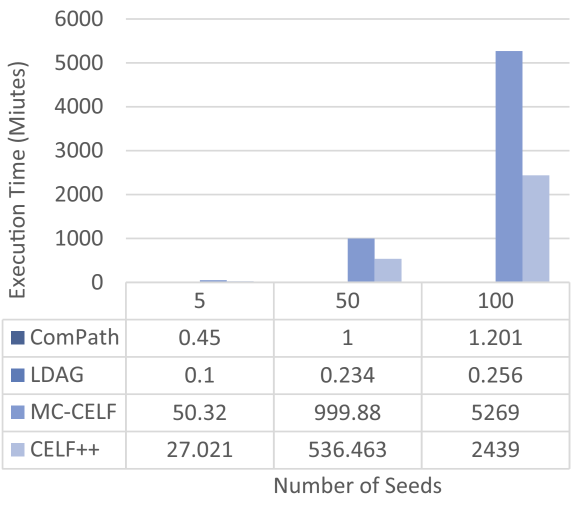
Table 5
Comparison of the proposed algorithm for different path length and number of seeds parameters based on NetHEPT.

Path Length	Average influence for 50 seeds	Average influence for 100 seeds
3	1409.4	2220.544
4	1416.453	2230.643
5	1416.8	2240.3



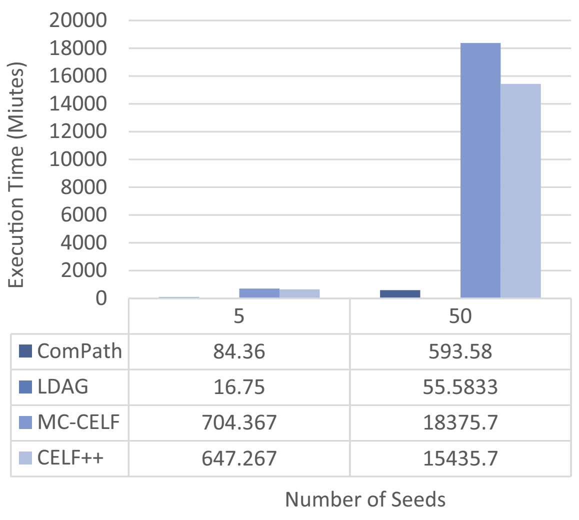
	5	50
■ High-Degree	5772.68	15459.911
■ LDAG	6026.17	15970.911
■ PageRank	5799.23	15934.781
■ CELF++	6026.17	16200.85
■ MC-CELF	6026.17	16344.1
■ ComPath	6026.17	16362.38

Fig. 7. Comparison of ComPath with other algorithms on Epinion dataset.



	5	50	100
■ ComPath	0.45	1	1.201
■ LDAG	0.1	0.234	0.256
■ MC-CELF	50.32	999.88	5269
■ CELF++	27.021	536.463	2439

Fig. 8. Execution time comparison of the algorithms on NetHEPT.



	5	50
■ ComPath	84.36	593.58
■ LDAG	16.75	55.5833
■ MC-CELF	704.367	18375.7
■ CELF++	647.267	15435.7

Fig. 9. Execution time comparison of the algorithms on Epinion.

Table 6
Over performance percentage of ComPath based on the number of activated nodes.

Algorithm	CELF++	MC-CELF	LDAG	High degree
Over performance on NetHEPT	0.56%	1%	6.7%	22.3%
Over performance on Epinion	0.99%	0.11%	2.45%	5.8%

threshold, the edge between them is ignored by the algorithm. Determining an optimized value of η may be very hard for many graphs. On the other hand, our proposed algorithm uses a path length parameter that can be easily estimated in real world networks (the optimized path length value is 3, 4 or 5 for most graphs). Our experimental evaluation of ComPath showed that by computing each node's influence on its neighbors with a path length less than 5, most influential nodes of the network can be identified by the algorithm. Also, the proposed algorithm only requires to obtain the number of nodes in the *Selected_Nodes* set with maximum value of $\delta(x)$, therefore the algorithm requires to execute the Backtrack function for *j*–*i* times where *j* is equal to the number of nodes in the *Selected_Nodes* set, and *i* is the cardinality of the *Seed* set. This fact helps to reduce the number of inspected nodes and so the proposed algorithm runs much faster than the other algorithms (See Tables 6 and 7).

Another benefit of the proposed algorithm, compared to other algorithms like UBLF (Zhou et al., 2014) and Simpath (Goyal et al., 2011b) is that it identifies the most influential communities of the network. In many occasions it is highly desirable to focus on a small sub-graph that can maximize the influence spread all over the input graph. For example, let us consider the problem of advertising a product in a social network. In our real world it is very hard to start advertising from some people that are distributed all over the network. So, advertisement companies try to focus on a community (or communities) that can maximize their advertisement effectiveness. They would like to know which communities (and which people in each community) are better to be considered as their advertisement target. The proposed algorithm can also provide a good answer for this kind of influence maximization problem.

Table 7
ComPath improvement of execution time compared to other algorithms.

	CELF++	MC-CELF	LDAG
Improvement on NetHEPT dataset	98.33%	99.10%	No Improvement
Improvement on Epinion dataset	86.96%	88.02%	No Improvement

5. Conclusion

In this research, an efficient algorithm, named ComPath, was proposed based on the linear model to find top-k most influential people in social networks. Our experiments showed the algorithm provides a good balance between execution time and efficiency. The algorithm detects the communities of the input network at first and investigates a limited number of them to reduce the execution time. Also, ComPath is able to suggest a limited number of communities to be investigated from the input network. This is very useful in our real world because it enables us to choose a few number of communities as our target and reduce the expenses for different applications such as product advertisement, etc.

As mentioned before, for computing the influence spread of nodes in the linear threshold model, the formula $\delta(v) = 1 + \sum_{u\in N^{out}(v)} b_{v,u} \cdot \delta^{V-v}(u)$ is used. In this research, we introduced two modifications to this formula. The first one was to limit the number of nodes that are examined that is accomplished by using the centrality measures, community detection and computing the *Selected_Nodes* set. The second one was to limit the search space of the input graph in the set *v'*. The size of *v'* depends on the *Selected_Nodes set* and the chosen path length. Finally, the formula used in this paper is:

$$\delta(v) = 1 + \sum_{u\in N^{out}(v)} b_{v,u} \cdot \delta^{V'-S_i-v}(u) = \sum_{u\in V'} \gamma_{v,u}^{V'-S_i}, \quad v \in Selected\_Nodes$$

As the experimental results showed, the mentioned modifications improved the execution time and the information spread.

Several challenges and future directions remain to investigate: one of them would be to use other centrality measures such as PageRank or K-Center to calculate the member of the *Selected_Nodes* set. Another challenge would be to use different algorithms for detecting communities and to investigate the parameters that affect the community detection algorithms.

References

Bakshy, E., Hofman, J. M., Mason, W. A., & Watts, D. J. (2011). Everyone's an influencer: Quantifying influence on twitter. In *Paper presented at the fourth ACM international conference on web search and data mining*. Hong Kong, China.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems, 30*(1–7), 107–117. http://dx.doi.org/10.1016/s0169-7552(98)00110-x.

Chen, W., Wang, C., & Wang, Y. (2010a). Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Paper presented at the 16th ACM SIGKDD international conference on knowledge discovery and data mining*. Washington, DC, USA.

Chen, W., Wang, Y., & Yang, S. (2009). Efficient influence maximization in social networks. In *Paper presented at the 15th ACM SIGKDD international conference on knowledge discovery and data mining*. Paris, France.

Chen, W., Yuan, Y., & Zhang, L. (2010). Scalable influence maximization in social networks under the linear threshold model. In *Paper presented at the 2010 IEEE international conference on data mining*.

Domingos, P., & Richardson, M. (2001). Mining the network value of customers. In *Paper presented at the seventh ACM SIGKDD international conference on knowledge discovery and data mining*. San Francisco, California.

Goyal, A., Bonchi, F., & Lakshmanan, L. V. S. (2010). Learning influence probabilities in social networks. In *Paper presented at the third ACM international conference on web search and data mining*. New York, New York, USA.

Goyal, A., Bonchi, F., & Lakshmanan, L. V. S. (2011). A data-based approach to social influence maximization. *VLDB Endowment, 5*(1), 73–84. http://dx.doi.org/10.14778/2047485.2047492.

Goyal, A., Lu, W., & Lakshmanan, L. V. S. (2011a). CELF++: Optimizing the greedy algorithm for influence maximization in social networks. In *Paper presented at the 20th International conference companion on world wide web*. Hyderabad, India.

Goyal, A., Lu, W., & Lakshmanan, L. V. S. (2011b). SIMPATH: An efficient algorithm for influence maximization under the linear threshold model. In *Paper presented at the 2011 IEEE 11th International Conference on Data Mining*.

Gregory, S. (2010). Finding overlapping communities in networks by label propagation. *New Journal of Physics, 12*(10), 103018.

Guo, J., Zhang, P., Zhou, C., Cao, Y., & Guo, L. (2013). Item-based top-k influential user discovery in social networks. In *2013 IEEE 13th international conference on Data mining workshops (ICDMW)* (pp. 780–787). IEEE.

Kempe, D., Kleinberg, J., & Tardos, E. (2003). Maximizing the spread of influence through a social network. In *Paper presented at the ninth ACM SIGKDD international conference on knowledge discovery and data mining*. Washington, D.C.

Kimura, M., & Saito, K. (2006). Tractable models for information diffusion in social networks. In *Paper presented at the 10th european conference on principle and practice of knowledge discovery in databases*. Berlin, Germany.

Kim, J., Kim, S. K., & Yu, H. (2013). Scalable and parallelizable processing of influence maximization for large-scale social networks? In *2013 IEEE 29th international conference on data engineering (ICDE)* (pp. 266–277). IEEE.

Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., & Glance, N. (2007). Cost-effective outbreak detection in networks. In *Paper presented at the 13th ACM SIGKDD international conference on knowledge discovery and data mining*. San Jose, California, USA.

Lancichinetti, A., Fortunato, S., & Kertesz, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics, 11*(3), 033015.

Liu, Q., Xiang, B., Zhang, L., Chen, E., Tan, C., & Chen, J. (2013). Linear computation for independent social influence. In *2013 IEEE 13th international conference on data mining (ICDM)* (pp. 468–477). IEEE.

Lu, Z., Zhang, W., Wu, W., Kim, J., & Fu, B. (2012). The complexity of influence maximization problem in the deterministic linear threshold model. *Journal of Combinatorial Optimization, 24*(3), 374–378.

Li, H., Bhowmick, S. S., Sun, A., & Cui, J. (2014a). Conformity-aware influence maximization in online social networks. *The VLDB Journal*, 1–25.

Li, D., Xu, Z. M., Chakraborty, N., Gupta, A., Sycara, K., & Li, S. (2014b). Polarity related influence maximization in signed social networks. *PloS One, 9*(7), e102199.

Morid, M. A., Shajari, M., & Hashemi, A. R. (2014). Defending recommender systems by influence analysis. *Information Retrieval, 17*(2), 137–152. http://dx.doi.org/10.1007/s10791-013-9224-5.

Narayanam, R., & Narahari, Y. (2010). A shapley value-based approach to discover influential nodes in social networks. *IEEE Transactions on Automation Science and Engineering, 99*, 1–18.

Ohsaka, N., Akiba, T., Yoshida, Y., & Kawarabayashi, K. I. (2014). Fast and accurate influence maximization on large networks with pruned monte-carlo simulations 10.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physics and Society, 76*(3), 036106.

Richardson, M., Agrawal, R., & Domingos, P. (2003). Trust management for the semantic web. In D. Fensel, K. Sycara, & J. Mylopoulos (Eds.). *The 2nd international semantic web conference (ISWC2003)* (Vol. 2870, pp. 351–368). Berlin Heidelberg: Springer.

Song, X., Tseng, B. L., Lin, C. -Y., & Sun, M. -T. (2006). Personalized recommendation driven by information flow. In *Paper presented at the 29th annual international ACM SIGIR conference on research and development in information retrieval*. Seattle, Washington, USA.

Valiant, L. G. (1979). The complexity of enumeration and reliability problems. *SIAM Journal on Computing, 8*(3), 410–421.

Wang, Y., Cong, G., Song, G., & Xie, K. (2010). Community-based greedy algorithm for mining top-K influential nodes in mobile social networks. In *Paper presented at the 16th ACM SIGKDD international conference on knowledge discovery and data mining*. Washington, DC, USA.

Weng, J., Lim, E. -P., Jiang, J., & He, Q. (2010). TwitterRank: Finding topic-sensitive influential twitterers. In *Paper presented at the third ACM international conference on web search and data mining*. New York, New York, USA.

Xie, J., Szymanski, B. K., & Liu, X. (2011). SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: *Paper presented at the 2011 IEEE 11th international conference on data mining workshops*.

Yang, Y., Chen, E., Liu, Q., Xiang, B., Xu, T., & Shad, S. A. (2012). On approximation of real-world influence spread. In *Machine learning and knowledge discovery in databases* (pp. 548–564). Berlin Heidelberg: Springer.

Zhang, Y., Zhou, J., & Cheng, J. (2011). Preference-based top-k influential nodes mining in social networks. In *Paper presented at the 2011 IEEE 10th international conference on trust, security and privacy in computing and communications*.

Zhou, C., Zhang, P., Guo, J., & Guo, L. (2014). An upper bound based greedy algorithm for mining top-k influential nodes in social networks. In *Proceedings of the companion publication of the 23rd international conference on world wide web companion* (pp. 421–422). International World Wide Web Conferences Steering Committee.