Information Processing and Management 52 (2016) 1188–1199

Contents lists available at ScienceDirect

Information Processing and Management

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

INCIM: A community-based algorithm for influence maximization problem under the linear threshold model

Arastoo Bozorgi[a,*], Hassan Haghighi[a], Mohammad Sadegh Zahedi[b], Mojtaba Rezvani[c]

[a] Electrical and Computer Engineering, Shahid Beheshti University, G.C., Tehran, Iran
[b] Electrical and Computer Engineering, Tehran University, Tehran, Iran
[c] Research School of Computer Science, Australian National University, Canberra, Australia

ARTICLE INFO

Article history:
Received 19 November 2014
Revised 24 March 2016
Accepted 19 May 2016
Available online 27 May 2016

Keywords:
Influence spread
Linear threshold model
Community detection
Influence maximization
Social network

ABSTRACT

With the proliferation of graph applications in social network analysis, biological networks, WWW and many other areas, a great demand of efficient and scalable algorithms for graph mining is rising. In many applications, finding the most influential nodes in the network is informative for the network analyzers in order to track the spread of information, disease and rumors. The problem of finding the top k influential nodes of a directed graph $G = (V, E)$ such that the influence spread of these nodes will be maximized has long been exposed and many algorithms have been proposed to deal with this problem. Despite the useful characteristics of community structure in social networks, only a few works have studied the role of communities in the spread of influence in social networks.

In this paper we propose an efficient algorithm (which has an acceptable response time even for large graphs) for finding the influential nodes in the graph under linear threshold model. We exploit the community structure of graph to find the influential communities, and then find the influence of each node as a combination of its local and global influences. We compare our algorithm with the state-of-the-art methods for influence maximization problem and the results of our experiments on real world datasets show that our approach outperforms the other ones in the quality of outputted influential nodes while still has acceptable running time and memory usage for large graphs.

1. Introduction

A social network is modeled as a graph $G = (V, E)$ where V and E are the set of nodes and edges of G. In a real world social network, people are modeled as the set of nodes V and the relationship between them, e.g., friendship or being co-tagged in a photo is modeled as the set of edges E of the graph. Information can propagate via links between people which leads to word-of-mouth advertising and its famous application, viral marketing. In viral marketing, the owner of a product, gives free or discounted samples of a product to a group of people to gain a large number of adoptions through the word-of-mouth effect. The influence maximization problem is motivated by the idea of viral marketing.

* Corresponding author.
E-mail addresses: a.bozorgi@mail.sbu.ac.ir (A. Bozorgi), h_haghighi@sbu.ac.ir (H. Haghighi), sadeghzahedi@ut.ac.ir (M. Sadegh Zahedi), mojtaba.rezvani@anu.edu.au (M. Rezvani).

http://dx.doi.org/10.1016/j.ipm.2016.05.006
0306-4573/

Kempe, Kleinberg, and Tardos (2003) defined the influence maximization problem in a graph $G = (V, E)$ as finding a subset of $S \subseteq V$ containing of k nodes, such that, as the spread of influence starts from S, the set of nodes that get activated, will be maximized.

Two propagation models are defined in Kempe et al. (2003), namely, Linear Threshold (LT) and Independent Cascade (IC) model. Also, different algorithms have been proposed based on IC model (Chen, Wang, Wang, 2010; Cheng, Shen, Huang, Chen, & Cheng, 2014; Guo, Zhang, Zhou, Cao, & Guo, 2013; Kim, Lee, Park, & Lee, 2013) and LT model (Chen, Yuan, Zhang, 2010; Goyal, Lu, & Lakshmanan, 2011a; 2011b). Our proposed algorithm in this paper is based on Linear Threshold model.

The greedy algorithm (Kempe et al., 2003) needs to calculate the influence of every node in the graph which is very time consuming for large graphs. In recent years, several attempts have been made to solve the influence maximization problem more efficiently. In CELF (Leskovec et al., 2007), first, the spread of each node is calculated and in the next iterations, based on the sub-modularity of spread function, only the spread of some nodes needs to be updated that causes CELF algorithm runs more faster than greedy algorithm. However, the time complexity bottleneck of CELF (Leskovec et al., 2007) is the first iteration.

In some papers such as Goyal, Lu, and Lakshmanan (2011b); Yu, Kim, and Kim (2013) and Kim et al. (2013), the authors try to compute the spread of nodes in smaller subgraphs to have a faster first iteration than CELF algorithm (Leskovec et al., 2007) and also, improve the overall running time of their algorithms. In IPA (Yu et al., 2013), the influence paths starting from each node v to other nodes are found and each influence path is considered as an influence evaluation unit and the node with maximum influence propagation probability over influence paths is a candidate for a seed node. However, the results of our experiments in Section 5 show that the quality of seeds in IPA is not satisfying.

In SIMPATH (Goyal et al., 2011b), to avoid calculating spread of every node in the graph, the vertex cover of the graph is computed and the spread of each node in the vertex cover is calculated. In this paper, we improve the running time of the SIMPATH algorithm (Goyal et al., 2011b) by using the community structure of the input graph.

Despite useful and meaningful characteristics of communities in social networks, very little attention has been made to incorporate the role of communities in influence maximization problem. In real social networks, people live in a cluster with whom they have strong relations. Information circulates at a high velocity within these clusters and each person tends to know what the other people know. Therefore, the spread of information on new ideas and opportunities must come through the weak ties that connect people in separate clusters. The weak ties so often ignored by social scientists are in fact a critical element of social structure. Weak ties are essential to the flow of information that integrates otherwise disconnected social clusters into a broader society (Granovetter, 1973). Furthermore, some communities in social networks play a significant role, as they are central and other communities monitor them to get the updates. Moreover, real networks contain a huge number of nodes, and computing the influence of each node is very expensive. Calculating the spread of each node locally inside its community can be done very quickly which improves the running time.

Kim et al. (2013) approached the influence maximization problem from a community based perspective and their main reason is to limit the search space to some nodes inside communities and decrease the running time. In Kim et al. (2013), first the graph is clustered into a set of communities, and then the most influential node in each community is considered as the seed candidate for the influence maximization problem. Finally, only k nodes from candidate nodes are chosen as final seed nodes. By the way, in this model, the role of communities is ignored. For example, if some communities are very small and have no influence on other communities, the quality of seeds in that communities is expected to be lower.

Contribution. Our major contribution in this research is summarized as follows:

- We design a new framework to incorporate community structure in the influence maximization problem.
- We devise an efficient algorithm for finding the top-k influential nodes in a social network based on communities in the graph.

Methodology. In this paper, we incorporate the role of communities of the social network in a meaningful manner and propose an algorithm under Linear Threshold model. In our algorithm, first we find the local spread of each node inside its community. Then we construct the graph of communities, where each community is a vertex and there is an edge between two vertices if there is at least one edge between the corresponding communities in the actual graph. After that, we compute the global influence that is the spread of each community in the graph of communities and finally calculate the final spread of each node as a combination of its local spread and its global spread. So, a particular node that belongs to a community with higher spread, has more chance to be chosen as a seed node candidate.

We conduct the empirical studies on large real graphs. Extensive performance studies demonstrate that the proposed algorithm significantly outperforms the state-of-the-art algorithms in term of the quality of outputted seeds while still has an acceptable running time and memory usage.

Organization. The rest of this paper is organized as follows. In Section 2, we summarize the related works, and in Section 4, we devise a new algorithm for finding the top influential nodes in the graph called INCIM. Section 5 stands for introducing the datasets which are used to examine our algorithm and presenting the experimental results. The last section is devoted to conclusions and some directions for future work.

2. Related work

In the previous section, we mentioned some works closer to ours and in this section, these works and other related works are reviewed in a more detailed way. The greedy algorithm presented in Kempe et al. (2003) can guarantee a good quality of seeds, but it is very time consuming for large graphs. Therefore, efficiently and accurately finding the influential nodes in social networks, under the LT and IC models, has recently drawn a great deal of attentions.

CELF algorithm presented in Leskovec et al. (2007) owes to the fact that the influence spread function is a sub-modular function, which means the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding the same element to a superset of S (Kempe et al., 2003). Therefore, CELF reduces the number of calls to the spread estimation function (because only the spread of some nodes needs to be updated in each iteration) and improves the running time of the greedy algorithm. However, the time complexity bottleneck of CELF is the first iteration, where the spread of every node is calculated and thus the CELF's first iteration is the same as the greedy's first iteration.

In the heuristic approach in Chen et al. (2010), for each node v, a DAG (Directed Acyclic Graph) is constructed and the spread of nodes is computed locally within the resulting DAGs and the seed nodes are selected based on the greedy algorithm. This algorithm has a reasonable running time, but it takes a lot of memory to store a DAG for each node in the graph. The main idea of the algorithm presented in Chen et al. (2010) is making local trees with edges starting or ending at each node called MIA; then the probability of activating a node by other nodes in each tree is computed locally. Finally a node with maximum probability is chosen as seed node.

In SIMPATH (Goyal et al., 2011b), to reduce the number of calls to the influence estimation function, first the vertex cover of G is computed, and only the spreads of nodes within the vertex cover are calculated. Having the spread values, the seed nodes are found in k iterations. In IMRank (Cheng et al., 2014), an initial rank is assigned to every node in the graph which is computed from a known heuristic; then, based on these initial ranks, the spreads of nodes are estimated, while in the greedy algorithm the spreads are computed exactly in each iteration. In IPA (Yu et al., 2013), for each node v in the graph, the algorithm finds the influence paths starting from v to other nodes and computes an influence propagation probability for each node. The nodes with maximum influence propagation probability are selected as seed nodes. IPA algorithm runs very fast and also is parallelizable to runs faster in multicore systems, but it achieves seed nodes with low quality over different datasets comparing to other algorithms.

In Guo et al. (2013), influence maximization problem is viewed from a new perspective. More specifically, k nodes are chosen as seed sets that have maximum influence over a set of target nodes to obtain a personalized set of influential nodes. The personalized influence is specific to its own applications where the seeds are selected from a general social network for a specific type of product.

Very closely related to our work is Kim et al. (2013), where communities of the graph are constructed by Markov Clustering algorithm. Then, in each community, a node with maximum spread is chosen as the seed node candidate. Finally, k nodes with bigger influence spread are chosen from candidate nodes. The main problem with this algorithm is that, it does not consider the role of each community as a unit to spread the influence and the size of each community.

3. Preliminaries

In this part, we introduce fundamental concepts of influence calculation which are needed for better understanding of the paper.

Community detection. Communities are subsets of nodes in the graph, with more edges between them and fewer edges between nodes in different communities (Luce & Perry, 1949). Community detection is formulated as a clustering problem. That is, given the full graph $G = (V, E)$, partition the vertex set into k subsets $S_1, S_2, \ldots, S_K$, such that $\bigcap_{i=1}^{k} S_i = \emptyset$ and $\bigcup_{i=1}^{k} S_i = V$. A quality metric $Q(S_1, \ldots, S_K)$ is defined over the partitions and a community detection algorithm will try to find a partitioning that maximize or minimize Q depending on its nature. This is for non-overlapping community detection and one can simply remove the constraint $\bigcap_{i=1}^{k} S_i = \emptyset$ to get the overlapping version (Hu, Chow, & Lau, 2014).

Linear threshold model. In this model which is defined in Kempe et al. (2003), activation probability of a node depends on a uniform function of its neighbors that are activated before. More specifically, a weight function f_v maps the neighbors of v to [0, 1], then assigns a threshold value $\theta_v \in [0, 1]$ to each node v uniformly at random. Node v would be activated in time t if $f_v(S) > \theta_v$, which S is the subset of neighbors v that have been activated in time $t - 1$. Based on Kempe et al. (2003), the value of f_v is initialized as Eq. (1).

$$f_v(S) = \sum_{u \in S} b_{v,u} \tag{1}$$

$b_{v,u}$ is the weight of edge (v, u) and sum of all edges between v and its neighbors should be less than 1 as Eq. (3).

$$\sum_{u \ neighbors \ of \ v} b_{v,u} <= 1 \tag{2}$$

In some cases, the weight of the edges are not set in the input graph and they should be computed. To do so, the number of edges between two nodes i and j are defined as w_{ij}. Also, the number of input edges to node j is defined as d_j. Finally

the weight of edge (i, j) in LT model is as follows:

$$\frac{w_{ij}}{d_j} \tag{3}$$

4. INCIM algorithm

In this section, we devise a new algorithm for the influence maximization problem from a community-based perspective. First, we present a general overview of the algorithm and then we explain each part in more details.

4.1. Algorithm overview

Given a set of mutually disjoint communities $C = \{C_1, C_2, \ldots, C_l\}$ in a graph $G = (V, E)$, such that $C_1 \cup C_2 \cup \cdots \cup C_l = V$ and for each i and j, $C_i \cap C_j = \emptyset$, the intuition of INCIM can be explained using the characteristics of community structure in G. Our algorithm proceeds in two phases. In the first phase (preprocessing), we find the communities of the main graph by a partition algorithm. In the second phase, we first find the influence of each community among other communities based on the links between their nodes and then find the degree to which each node spreads the influence inside its own community. We determine the final spread of each node in the graph as a combination of its community influence and its influence inside its community.

Our innovations (which will be explained in Section 4.2 in more details) to decrease the running time and increase the efficiency of our algorithm are as follows:

- Computing the spread of nodes locally in the communities which causes a decrease in overall running time of the algorithm. Also, by determining those communities whose nodes spread should be updated, we decrease the number of calls to the SIMPATH algorithm which again results in decreasing the running time.
- By using the idea of local and global spreads of nodes and their combination, we track the role of communities and also, the influence of a node in its own community.
- Using a combination of the SIMPATH (Goyal et al., 2011b) and CELF (Leskovec et al., 2007) algorithms; more precisely, using the SIMPATH algorithm to compute the spread of nodes and storing and updating these spread values by the idea of the CELF algorithm cause our algorithm to find the seed nodes which have the most influence spread in the graph in a reasonable time and less memory usage.

4.2. Algorithm description

INCIM contains the following steps:

1. *Preprocessing:* In this step, we partition the input graph into its communities. The partition algorithm we have used is SLPA (Xie, Szymanski & Liu, 2011).
2. *Making the graph of communities:* We make the graph of communities whose nodes are the communities of the graph.
3. *Computing spread of nodes:* We use the manner of the SIMPATH algorithm (Goyal et al., 2011b) to compute the spread of nodes which is done in 3 steps as follows:
   (a) Computing global spread which is the spread value of each node in the graph of communities, i.e., the spread values of the communities.
   (b) Computing local spread which is the spread value of the nodes inside their communities.
   (c) Computing final spread which is a combination of local and global spreads per each node.
4. *Making CELF lists:* We store the resulting spread values of the nodes by the idea of the CELF (Leskovec et al., 2007) algorithm. For each community, we have a list which the number of its elements is equal to the number of nodes in that community.
5. *Determining communities to be updated:* We determine which communities should be updated based on the nodes in the seed set.
6. *Selecting seed nodes:* In this step, INCIM iterates k times to finds the k most influential nodes.

Now, we can describe INCIM algorithm by explaining the details of each step.

4.2.1. Preprocessing step

To partition the input graph into communities, we use the SLPA algorithm as the partition algorithm which is proposed in Xie, et al. (2011). The SLPA is categorized as a dynamic and agent-based algorithm and is an extension of the Label Propagation Algorithm (LPA). Both SLPA and LPA algorithms tries to maximize the modularity measure Q. Modularity measure Q is defined as the difference between the observed density of edges within communities and the expected density of edges within the same communities (Newman & Girvan, 2004). In SLPA, each node can be a listener or a speaker. The roles are switched depending on whether a node serves as an information provider or information consumer. The input of the SLPA algorithm is the input graph containing the id of the nodes and the edge weights in the LT model. The weight of the edges are considered when a listener wants to accept a label from its neighbors. For example, A listener x, has three neighbors

$y1$, $y2$ and $y3$, with edge weights 0.1, 0.2 and 0.3 respectively. When node x consider the labels from its neighbors, it will weight each label as $0.1/(0.1+0.2+0.3)$ for node $y1$, $0.2/(0.1+0.2+0.3)$ for node $y2$ and $0.3/(0.1+0.2+0.3)$ for node $y3$. Then, the listener accepts one label from the collection of labels received from neighbors following certain listening rule, such as selecting the most popular label from what it observed in the current step.

The main reasons to choose this algorithm are its less time consuming than the other ones in finding communities and its high quality in finding communities of the graph; the experiments done in Xie, et al. (2011) confirm such specifications of SLPA.

4.2.2. Computing spread of nodes

In our framework, we use the approach of the SIMPATH algorithm (Goyal et al., 2011b) to compute the spread of a node in the input graph. As mentioned in Goyal et al. (2011b), the spread of a node v is computed by summing the weights of all simple paths starting from v. Also the spread of a seed set S is the sum of that of nodes $u \in S$ in subgraphs induced by $V - S + u$. So, by considering simple paths starting from nodes of set S, we can compute the spread of set S in different subgraphs. To find simple paths in the graph, SIMPATH uses BACKTRACK algorithm that is presented in Johnson (1975); Kroft (1967). As the problem of enumerating all simple paths is #P-hard (Valiant, 1979), in the SIMPATH algorithm, only the paths are considered which their weights decreases rapidly as the length of the path increases. Thus, the influence can be captured by exploring the paths within a small neighborhood, where the size of the neighborhood can be controlled by a threshold value.

The SIMPATH algorithm is shown in algorithm 1. In algorithm 1, S is the seed set, η is the threshold value to control the size of the neighborhood, U is the set of nodes of the input graph and $\delta(S)$ is the spread value of S.

Algorithm 1 SIMPATH-Spread.

Input: S, η, U
Output: $\delta(S)$
1: $\delta(S) \leftarrow 0$
2: **for** each $u \in S$ **do**
3: $\quad \delta(S) \leftarrow \delta(S) + BACKTRACK(u, \eta, V - S + u, U)$
4: return $\delta(S)$

Algorithm 1 shows how the SIMPATH algorithm computes the spread value of a seed set. In an iterative manner in line 3 of the algorithm, all the nodes of S are given to the BACKTRACK algorithm and their spreads are computed. The final value returned by algorithm 1 is the sum of the spread values of all nodes in set S.

As we mentioned in Section 2, CELF algorithm (Leskovec et al., 2007) uses the sub-modularity of the influence spread function. In our algorithm, we take the advantages of CELF algorithm to reduce the number of calls to the SIMPATH algorithm. To achieve this goal, in our algorithm the spread values of nodes computed by SIMPATH are stored in a list sorted in decreasing order. Then, in each iteration of CELF algorithm, only the marginal gain of the top node of the list is re-computed and if needed, the list is resorted. if a node remains at the top, it is picked as the next seed (Leskovec et al., 2007). As we compute the spread of nodes in the communities of the input graph, we have lists separated per each community. By using the CELF idea to store the spread values and update the lists, we improve the running time of our algorithm.

Next, we formalize the notions of *local spread* and *global spread* of each node and describe the way we determine them in more details.

Global spread. Once we identify the communities, we can determine which communities are the central ones since they are monitored by individuals and other communities that look for ideas, news and innovations. For instance, consider a big community of researchers working on social networks; it can play a central role in spreading ideas and topics for other researchers as they follow this community to monitor the recent trends. Therefore, it makes sense to consider the position of each community inside these research groups and find the degree to which each community can influence other communities. In this way, we can determine central communities and find the influential nodes in these communities.

To determine central communities, we build the graph $G_c = (V_c, E_c)$ of communities and compute the spread of each of its nodes. Each community C_i is a node of G_c and if there is a directed edge from node $u \in C_i$ to node $v \in C_j$ in graph G, then there is a directed edge from the corresponding node C_i to C_j in G_c. We define W_{ij} as the maximum weight of the edges between two communities C_i and C_j. We also count the number of input edges to community C_i shown by d_{C_i} as the degree of this community. Similar to LT model, the weight of edge (C_i, C_j) in G_c is considered as $\frac{W_{ij}}{d_{C_j}}$. As we mentioned before, the model of information propagation which is used for the main graph is LT model, so we should have the same information spread model for the graph of communities, too. That's why we use the LT edge weight computation manner for graph of communities.

The spread of nodes in graph $G_c = (V_c, E_c)$ is computed by the SIMPATH algorithm which takes $G_c = (V_c, E_c)$ graph as its input. We call this spread as δ.

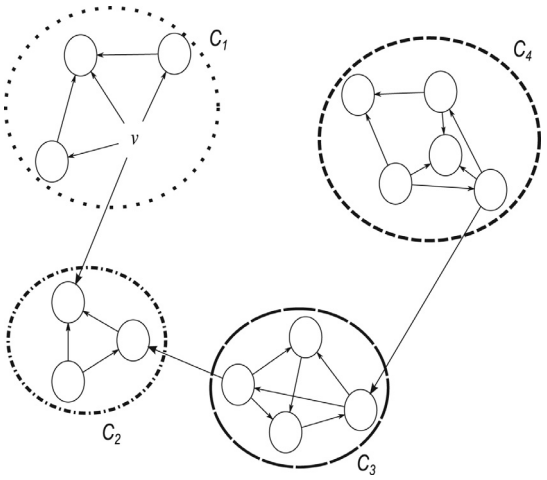

Fig. 1. A sample network with its communities.

Local spread. Suppose that there is a social network in which communities are constructed based on research interests in the academia. An influential node in a community has more influence on the nodes of its community than it does on the nodes of other communities with different research interests; so, in INCIM algorithm, each node has a local spread which is its influence inside its community. By computing the spread of nodes locally in the communities, we have an improvement in running time and also, we choose the most influential nodes in their own communities which results in choosing the most influential nodes as the algorithm's final output when combining the local spread with the global spread as will be explained below. The results of our experiments in Section 5 confirm this improvements.

In the first iteration of the algorithm, the spread of each node v in a community is computed as $\delta_l(v)$ and stored in list δ based on CELF optimization. For each community i, we have a list δ_i whose size is equal to the number of nodes in that community. In other word, the input nodes of the SIMPATH algorithm are the ones inside the communities separately for each community, and SIMPATH finds simple paths in the communities locally. As Eq. (4), the total number of elements of all δ_i is equal to the number of all nodes in graph G, so the total memory needed by INCIM is the same as CELF algorithm.

$$\sum_{i=1}^{l} |\delta_i| = |V| \tag{4}$$

In Eq. (4), l is the number of communities, $|\delta_i|$ is the size of δ_i and $|V|$ is the number of nodes in graph G.

Since the communities in a network are not necessarily well separated, there are some edges between them. Therefore, individuals in the same community can have different influence on the neighbor communities. To tackle this issue, we consider the notion of border nodes in the graph. A *border node* in a community has at least one edge that connects it to another community in the network and influence can enter or leave a community through border nodes. So, it is important to compute the spread of border nodes over the graph. We call SIMPATH with graph G as its input to compute the border node's spread and store the spread values in a vector called α with an element per each border node. These values will be combined with local and global spreads to track border node's influence above nodes from their linked communities. In Fig. 1, node v is a border node as it has a link to another node from another community.

Final spread, combination of local spread and global spread. Final spread of a node is a combination of its local spread and spread of its community. By final spread, we determine which nodes are the most influential nodes over the graph.

Definition 1. Given a social network $G = (V, E)$ and a set of communities $\{C_1, C_2, \ldots, C_l\}$, where $\delta_l(v)$ is the local spread of node v in community C_i and $\bar{\delta}(v)$ is the global spread of community C_i which v belongs to, along with $\alpha(v)$ which is the spread values of border nodes, the final Spread of node v is defined as follows:

$$\delta(v) = \delta_l(v) + \alpha(v).\bar{\delta}(v) \tag{5}$$

In Eq. (5), $\alpha(v)$ is equal to 1 if v is a non-border node; if v is a border node, $\alpha(v)$ is equal to the influence spread from v to the nodes inside its linked communities. As Goyal *et. al.* showed in Goyal et al. (2011b), the majority of the influence to a node flows in from a small neighborhood and can be captured by enumerating paths within that neighborhood. Also, Wang *et. al.* discussed in their community-based greedy approach (Wang, Cong, Song, & Xie, 2010) that the difference between the node's influence degree in its community and its influence degree in the whole network is small. In INCIM, we compute the spread of nodes locally in their own communities, too. But as we discussed earlier, the communities can also influence each other by the links which exist between them and we cannot ignore the influence which flows between the nodes

belong to different communities. Such influence is considered by combining the global spread with the local spread of each node. The global spread of a community is computed by summing up the edge weights of all simple paths starting from the community. Also, the local spread of a node is computed in the same way, too. So to combine the local spread and global spread of a node, we should sum up them together, as we did in Eq. (5). In Section 5.3, we will show that by using Eq. (5), we can achieve a good approximation for spread computation near to the approximation achieved by simple greedy algorithm (Kempe et al., 2003) which is $(1 - 1/e - \varepsilon)$.

4.2.3. Seed nodes selection

After finishing the first iteration of the algorithm, the node with maximum spread in community i takes the first place of δ_i, and the node with maximum spread between all first elements of all δ_i is chosen as the first seed node.

After choosing the first seed node in the first iteration, the algorithm will be executed $k-1$ times to find the $k-1$ remaining seed nodes. In each iteration, the SIMPATH algorithm is called and the δ_i lists would be updated based on the new spread values of nodes. Since the seed nodes are chosen from different communities, the spread of nodes in graph of communities $G_c = (V_c, E_c)$ should be also updated. So, in the beginning of each iteration, SIMPATH is called to compute the spread of nodes of graph $G_c = (V_c, E_c)$.

Suppose that Fig. 1 is a part of a network whose node v is chosen as a seed node at time t. As node v belongs to community C_1, the spread of other nodes in community C_1 should be updated for the next iteration. Since there is a simple path from node v to a node in community C_2, so the spread of nodes in community C_2 should be updated, too. But for communities C_3 and C_4, such updates are not required because there is no simple path from v to C_3 and C_4.

The INCIM algorithm calls SIMPATH to compute the spread of nodes in only those communities determined by the *getCommunitiesToUpdate* subroutine; in this way, the number of calls to the SIMPATH algorithm will be decreased remarkably in each iteration. This is one of the contributions for decreasing the running time of the algorithm. The *getCommunitiesToUpdate* subroutine is shown in algorithm 2. In algorithm 2, S is the seed set and $G_c(V_c, E_c)$ is the graph of communities.

Algorithm 2 getCommunitiesToUpdate.

Input: $S, G_c(V_c, E_c)$
Output: $UpdLst$
1: $UpdLst \leftarrow \emptyset$
2: $Neighbours \leftarrow \emptyset$
3: **for** each $v \in S$ **do**
4: $C_i =$ the community containing v
5: $UpdLst = UpdLst \cup C_i$
6: $Neighbours =$ every u that there is a simple path from v to u
7: **for** each $u \in Neighbours$ **do**
8: $C_j =$ the community containing u
9: $UpdLst = UpdLst \cup C_j$
10: return $UpdLst$

The *UpdLst* is the list of communities which should be updated and returned by algorithm 2. In the loop of line 3, the nodes which there is a simple path starting from nodes of set S to them are considered. Then the communities of such nodes and the communities of the nodes of set S are considered as the output of the algorithm.

In the framework proposed in this paper, we consider the role of each community by its global spread. Also, we track the influence spread achieved by each node in its community as local spread. By combining global and local spreads, we choose nodes as seed nodes that have most influence spread in their community and this influence can spread as more as possible to other communities. This is why we claim that our algorithm chooses the most influential nodes with higher quality comparing to other state-of-the-art algorithms. Also, by computing the spread of nodes locally in the communities and using the idea of CELF algorithm (Leskovec et al., 2007), we have improvements in running time and memory usage. The experimental results of Section 5 confirm our claims.

INCIM algorithm is shown in algorithm 3. In algorithm 3, $G(V, E)$ is the input graph and S is the set of size k containing seed nodes returned by INCIM algorithm.

In lines 3 and 4 of algorithm 3, the graph of communities is constructed based on the communities discovered by the SLPA algorithm. Then in lines 5-7, the spread values of the nodes in the graph of communities are computed. The influence spread of border nodes is computed in lines 9–11. In line 11, where $\bar{\delta}(v)$ is the global spread of v's community, we multiply the spread value of each border node with the global spread value of its community. In the beginning of each iteration starting from line 12, we call algorithm 2 to determine which communities should be updated in line 14 and store them in list *CP*. In the first iteration, after calling algorithm 2, *CP* contains all the communities of the graph because there is no seed node in the beginning of the algorithm. In lines 15-24, for all nodes of the communities which should be updated, we compute their spread values by the SIMPATH algorithm, and the node with the maximum spread is chosen as the seed node. In line 17, $\delta^{V-x}(S)$ is the spread value of seed set S in the graph containing nodes in set V except node x and its edges.

Algorithm 3 INCIM.

Input: $G(V, E), k$
Output: S

$S \leftarrow \emptyset$
$CommunitiesSet \leftarrow \emptyset$
call SLPA algorithm to find the communities of G and store them in $CommunitiesSet$
make graph of communities $G_c(V_c, E_c)$ such that each community is a graph node
for each $u \in G_c$ **do**
  call algorithm 1 to compute $\bar{\delta}(u)$
  add $\bar{\delta}(u)$ to $\bar{\delta}$ which is ordered decreasingly
find border nodes and store them in $BorderNodesSet$
for each $v \in BorderNodesSet$ **do**
  call algorithm 1 to compute $\delta(v)$
  $\delta(v) = \delta(v) * \bar{\delta}(v)$
while $|S| < k$ **do**
  $CP \leftarrow \emptyset$
  call algorithm 2 to determine the communities which should be updated and store them in CP
  for each $c \in CP$ **do**
    $maxSpread = 0$
    call algorithm 1 to compute $\delta^{V-x}(S), \forall x \in \delta_c$
  for each $x \in \delta_c$ **do**
    call algorithm 1 to compute $\delta^{V-S}(x)$
    update δ_c based on the new spread value of x
    u = top node of δ_c
    if $\delta(u) > maxSpread$ **then**
      $maxSpread \leftarrow \delta(u)$
  $S = S \cup u$
return S

Table 1
Specifications of real standard datasets.

	NetHEPT	Slashdot	Amazon	DBLP
#nodes	15K	82K	262K	914k
#edges	62K	948K	1.2M	6.6M
Average out-degree	4.12	9.8	9.4	7.2
Maximum out-degree	64	1527	425	950
#connected components	1781	1	1	41.5K
Largest component size	6794	82K	262K	789K

5. Evaluation

To compare our algorithm with other approaches, we have done our experiments on four real datasets and compared the algorithms based on running time, the quality of seed nodes and memory usage. The code is written in C++ and all experiments are run on a Linux (Ubuntu 10.4) machine with 3.2 GHz Intel Xeon CPU and 128GB memory.

5.1. Datasets

We have used four real-world datasets to run our experiments on; their specifications are shown in Table 1 and available on the SNAP library of Stanford University website[1].Using SLPA algorithm, we find the communities of the datasets to be used by INCIM algorithm. The specifications of the found communities of the datasets are shown in Table 2. In both Tables 1 and 2, the # sign indicates the number of elements.

5.2. Algorithms to compare

INCIM: The algorithm presented in this paper.
LDAG: The approach proposed in Chen et al. (2010). We set the parameter $\theta = 1/320$ as recommended by authors.
SIMPATH: The algorithm in Goyal et al. (2011b) running with parameters $\eta = 10^{-3}$ and $l = 4$ as recommended by authors.

[1] http://snap.stanford.edu/

Table 2
Specification of communities of the datasets.

	NetHEPT	Slashdot	Amazon	DBLP
#Communities	2901	12K	32674	73489
#Nodes in the biggest community	407	6050	2852	771
#Edges in the biggest community	2938	16294	7169	3193

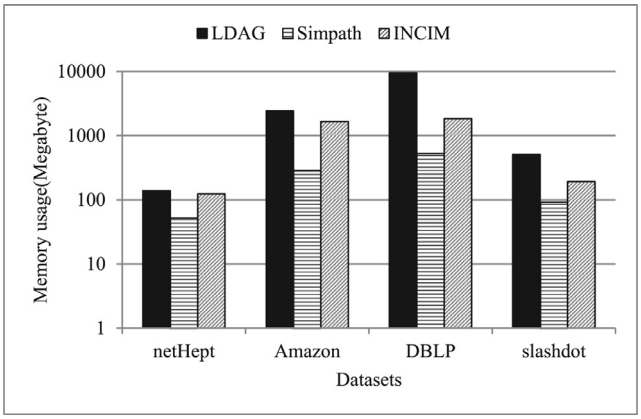

Fig. 2. Comparison based on memory usage in different algorithms.

IPA: The proposed algorithm in Yu et al. (2013) running with $threshold = 0.005$ as recommended by authors.

PageRank: The algorithm proposed in Brin and Page (1998). In this paper, nodes with maximum ranking are chosen as seed nodes. The algorithm stops when the score vectors from two consecutive iterations differ by at most 10^{-6} as per $L1 - norm$.

HighDegree: This algorithm (Kempe et al., 2003) chooses the nodes with maximum out-degree as seed nodes.

Our reasons to compare INCIM with the aforementioned algorithms are as follows: PageRank and HighDegree are two well-known and basic algorithms which are compared with most of the other works. SIMPATH is an algorithm with good results in running time, memory usage and quality of seed nodes. Also, LDAG has good results in quality of seed nodes and reasonable response time. At last, IPA uses the idea of communities to find the most influential nodes.

5.3. Experimental results

Comparison of memory usage The memory usage of HighDegree and the PageRank algorithms is almost zero because they don't need to store any structures when they are running, but LDAG has the most memory usage among compared algorithms because this algorithm make a DAG for each node in the graph. INCIM uses higher amount of memory than SIMPATH but lower than LDAG. The results are shown in Fig. 2 for 50 seed sets. In this figure, the y-axis has a logarithmic scale in base 10.

Comparison of seed set quality An algorithm with higher quality is the algorithm that has higher influence spread. Based on the experimental results shown in Fig. 3, the INCIM algorithm has the highest quality seed set among other algorithms. While INCIM uses the SIMPATH algorithm to compute the spread of nodes, its spread of seed set is a little better than SIMPATH in most of the cases because INCIM uses a combination of local and global spreads and thus tracks both the effect of each node in its community and the effect of each community.

The IPA algorithm has lower quality of seed sets than other algorithms in all datasets, except the slashdot dataset. In netHept and Amazon, IPA has the lowest quality of seed set and in DBLP, its quality is higher than the PageRank and HighDegree algorithms but lower than INCIM, SIMPATH and LDAG. As we can see, IPA has an uncertain behavior because for some datasets, the quality of its seed nodes is higher than other algorithms while for most others, it results in lower quality; hence, the IPA algorithm is not so reliable.

Comparison of running times Fig. 4 shows the results of comparisons based on running times. In the NetHept dataset, the running time of PageRank and HighDegree are considerably low, so the plots for these algorithms in NetHept dataset are omitted. IPA has better running time than INCIM, SIMPATh and LDAG and runs in a time close to that of PageRank and HighDegree. The INCIM algorithm has the best running time among other algorithms, except IPA, PageRank and HighDegree.

As we see in Fig. 4a and 4d, INCIM performs slower than SIMPATH at the beginning of its running time. For instance, in Fig. 4a, the time needed to find seed nodes until the number of seed nodes reaches 15, is higher than the time needed in the SIMPATH algorithm. The reason is that, the INCIM algorithm finds the communities of the graphs in the preprocessing step. Also, in the first iteration, INCIM computes the spread of the nodes of all communities, but in other iterations, only

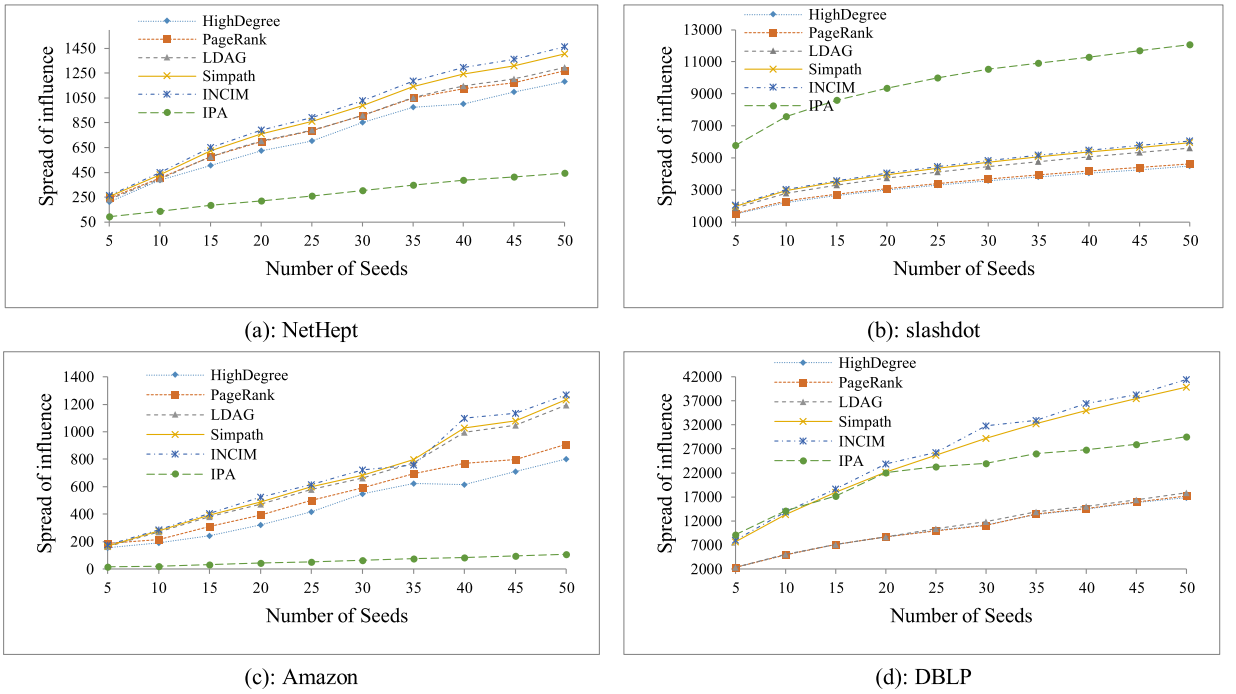

Fig. 3. Influence spread achieved by various algorithms.

Table 3
Speed up in different datasets .

Algorithms compared	Datasets			
	DBLP	Amazon	Slashdot	NetHept
Simpath	29%	27%	68%	43%
LDAG	58%	53%	86%	54%

the communities which are considered by algorithm 2, should be updated. Table 3 shows the running time improvement of our proposed algorithm for choosing 50 seed nodes comparing with SIMPATH and LDAG which have reasonable running time and high seed set quality. As we can see, our approach has an improvement in running time between 27% and 86% based on different datasets.

The effect of communities As we mentioned in this paper, in the INCIM algorithm, seed nodes are chosen using communities of the input graph. In each iteration of the algorithm, computing the spread of each node is done locally in its community. Also, when node v is chosen from community C_i, the spread of other nodes of community C_i along with the spread of nodes in communities with a simple path from C_i, should be updated. Thus, the spread of only a limited number of nodes would be updated in each iteration, instead of updating all of the nodes in the graph. By this contribution, the algorithm runs faster in other iterations. Fig. 5 shows the number of communities that are updated in each iteration. The y-axis has a logarithmic scale in base 2.

INCIM in its first iteration, computes the spread of all nodes in all communities, but in other iterations, the number of communities to be updated is decreased, and the number of spread computation calls is decreased, too. As the number of seed nodes is increased, there are more simple paths from seed nodes to other nodes in other communities. So, after some iterations, the number of communities which should be updated, is increased in comparison to the second iteration where there is only one seed node; but the number of communities in later iterations is anyway less than that of the first iteration.

The efficiency of final spread computation To study how efficiently we compute the spread of nodes in our approach, we run INCIM and simple greedy algorithm (Kempe et al., 2003) which uses Monte Carlo simulations(MC) on two moderate and large datasets (Amazon and DBLP respectively) and compare the spread achieved by them. We choose 5 different randomly selected set of nodes as seed sets of size 10, 20, 30, 40 and 50 and run INCIM and MC simulations 10,000 times to compute the spread of the seed sets. The results are shown in Fig. 6.

As we can see in Fig. 6, the values which are computed by INCIM are very close to the values computed by Monte Carlo simulations for different sets of nodes. For Amazon dataset, the differences between the values computed by INCIM and MC are 0.68%, 0.5%, 0.8%, 0.62% and 0.85% for sets of size 10, 20, 30, 40 and 50 respectively. Also, For DBLP dataset, the differences between the values are 0.8%, 0.69%, 0.73%, 0.32% and 0.47% for sets of size 10, 20, 30, 40 and 50 respectively.

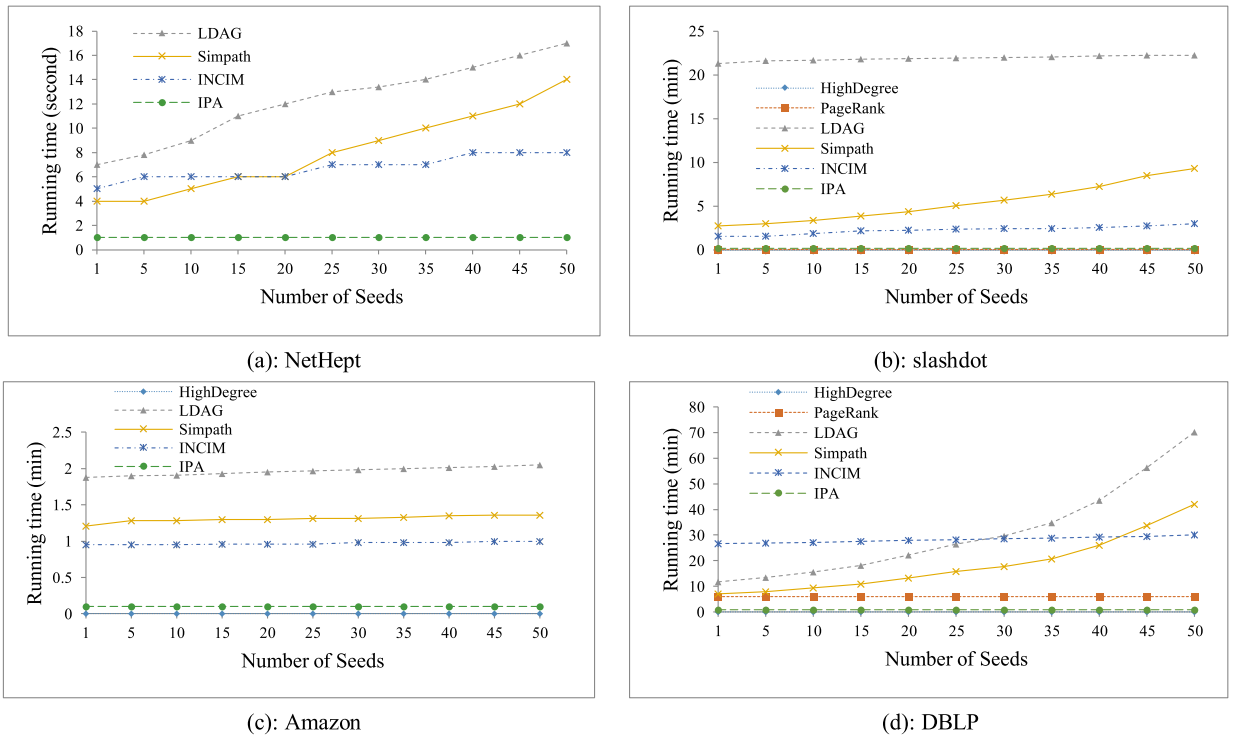

Fig. 4. Comparison of running time of different algorithms.

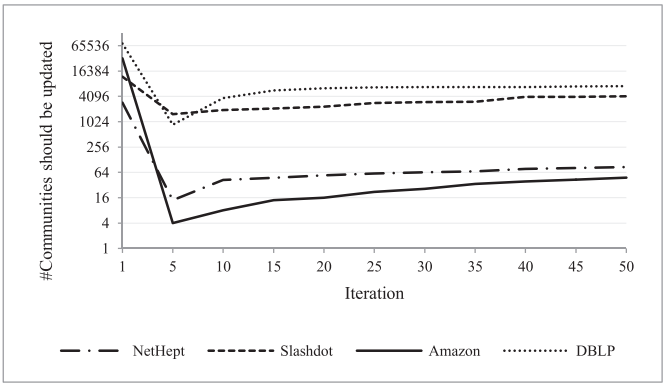

Fig. 5. Number of communities that should be updated in each iteration.

These results show that our approach computes the spread values efficiently by combining the local spread and global spread of each node.

6. Conclusion and future work

We can categorize the recent works in influence maximization problem in three groups. Some papers such as Chen et al. (2011); He, Song, Chen, and Jiang (2012); Liow, Cheng, and Lau (2012); Shirazipourazad, Bogard, Vachhani, Sen, and Horn (2012) study the influence maximization problem by considering competitors. In such papers, a competitive model is described which is in most cases an extension of linear threshold or independent cascade model. Some papers such as Chen, Lu, and Zhang (2012) studies the influence maximization problem with temporal constraints. The main goal of such works is that influence propagation will be done in a limited time. Other works try to propose an algorithm to solve the influence maximization problem in a reasonable time with lower memory usage and higher quality of seed sets. INCIM, our proposed algorithm, finds the influential nodes in communities of the graph. In this algorithm, spread computation of nodes is done locally in communities of the graph which causes a reasonable decrease in running time. Also, in each iteration of the algorithm, the marginal gain of only a limited number of nodes is computed, based on the communities they belong to,

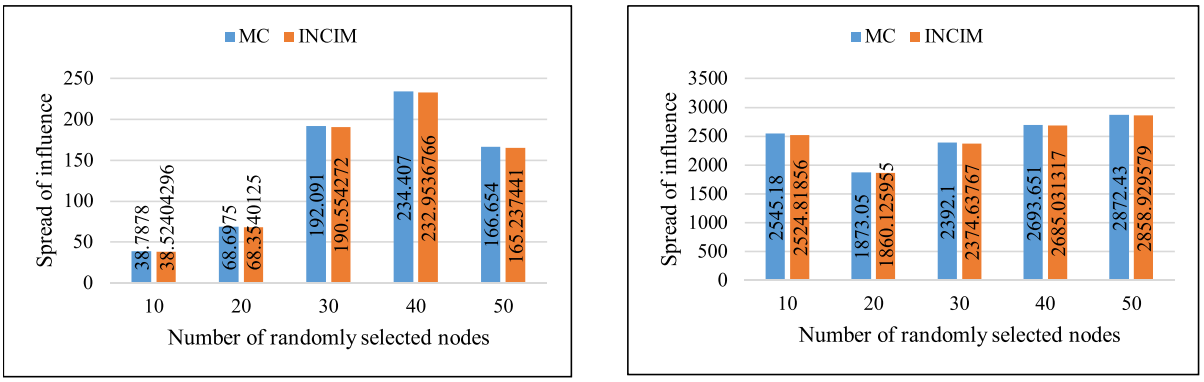

Fig. 6. The spread achieved by different randomly selected seed sets.

which causes the number of calls to the subroutine computing the spread of nodes, to be decreased, and thus the running time is decreased, too. Experimental results show the performance of our algorithm and its efficiency on large networks.

For future works, we are interested in studying the influence maximization problem by considering the graph content and choosing the influential nodes based on the topic given as input.

References

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems, 30*(1), 107–117.

Chen, W., et al. (2011). Influence maximization in social networks when negative opinions may emerge and propagate. In *Proceedings of SIGKDD, 2010* (pp. 379–390). ACM.

Chen, W., Lu, W., & Zhang, N. (2012). Time-critical influence maximization in social networks with time-delayed diffusion process. In *Proceedings of SIGIR, 2014* (pp. 475–484).

Chen, W., Wang, C., & Wang, Y. (2010a). Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of SIGKDD* (pp. 1029–1038). ACM.

Chen, W., Yuan, Y., & Zhang, L. (2010b). Scalable influence maximization in social networks under the linear threshold model. In *Proceedings of ICDM* (pp. 88–97). IEEE.

Cheng, S., Shen, H.-W., Huang, J., Chen, W., & Cheng, X.-Q. (2014). Imrank: Influence maximization via finding self-consistent ranking. In *Proceedings of SIGIR* (pp. 475–484).

Goyal, A., Lu, W., & Lakshmanan, L. V. (2011a). Celf++: Optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on world wide web* (pp. 47–48). ACM.

Goyal, A., Lu, W., & Lakshmanan, L. V. (2011b). Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Proceedings of ICDM* (pp. 211–220). IEEE.

Granovetter, M. S. (1973). The strength of weak ties. *American Journal of Sociology, 78*(6), 1360–1380.

Guo, J., Zhang, P., Zhou, C., Cao, Y., & Guo, L. (2013). Personalized influence maximization on social networks. In *Proceedings of CIKM* (pp. 199–208). ACM.

He, X., Song, G., Chen, W., & Jiang, Q. (2012). Influence blocking maximization in social networks under the competitive linear threshold model. *SDM* (pp. 463–474). SIAM.

Hu, P., Chow, S. S., & Lau, W. C. (2014). Secure friend discovery via privacy-preserving and decentralized community detection. arXiv: 1405.4951.

Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing, 4*(1), 77–84.

Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of SIGKDD* (pp. 137–146). ACM.

Kim, C., Lee, S., Park, S., & Lee, S.-g. (2013). Influence maximization algorithm using Markov clustering. In *Database systems for advanced applications* (pp. 112–126). Springer.

Kroft, D. (1967). All paths through a maze. *Proceedings of the IEEE, 55*(1), 88–90.

Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., & Glance, N. (2007). Cost-effective outbreak detection in networks. In *Proceedings of SIGKDD* (pp. 420–429). ACM.

Liow, L.-F., Cheng, S.-F., & Lau, H. C. (2012). *Niche-seeking in influence maximization with adversary* (pp. 107–112).

Luce, R. D., & Perry, A. D. (1949). A method of matrix analysis of group structure. *Psychometrika, 14*(2), 95–116.

Newman, M. E., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E, 69*(2), 026113.

Shirazipourazad, S., Bogard, B., Vachhani, H., Sen, A., & Horn, P. (2012). *Influence propagation in adversarial setting: How to defeat competition with least amount of investment* (pp. 585–594).

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 *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1039–1048). ACM.

Xie, J., Szymanski, B. K., & Liu, X. (2011). *SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process* (pp. 344–349).

Yu, H., Kim, S.-K., & Kim, J. (2013). Scalable and parallelizable processing of influence maximization for large-scale social networks. In *Proceedings of ICDE* (pp. 266–277). IEEE Computer Society.