Influence Maximization in Social Network

515030910594
Xiuwen Qin

***Abstract*—Influence maximization is a pop problem nowadays. Against the limited influence scope and high time complexity of existing influence maximization algorithms in social networks, a k-core filtered algorithm was introduced, its rank of nodes does not depend on the entire network. Secondly, pre-training was carried out to find the value of k which has the best optimization effect on existing algorithm but has no relation with the number of selected seeds. Finally, the nodes and edges that do not belong to the k-core subgraph were filtered by computing the k-core of the graph, then the existing influence maximization algorithms were applied on the k-core subgraph, thus reducing computational complexity. What's more, a new influence maximization algorithm named GI was proposed. It has more influence area and low time complexity.**

***Keywords*—social network, influence maximization, k-core, GI.**

I. Introduction

Influence maximization problem is to find t initial nodes and make the information can spread the largest area. Kempe define it a discrete optimization problem and prove that the influence maximization problem is NP-hard under independent cascade model. There has two typical spread model: Independent Cascade model and Linear Threshhold model. On the basic of these two models, people propose several algorithms, such as CELF, CCA, PMIA, IRIE and so on.

However, these algorithms still have a gap compared with the optimal solution. To improve existing algorithms, I propose a algorithm based on the k-core. It can apply in existing algorithms and enlarge the influence area and lower the time complexity. What,s more, I find a new improved algorithm GI which can influence more area and execute less time.

II. Experiment

A. Problem Definition

- **Definition 1:** $\boldsymbol{u}_n$: In directed graph G=(V,E), given an input t. The influence maximization problem is to find a node subset $S' \in V$, the influence area $\sigma(S')$ satisfy:

$$\sigma(S') = max\{\sigma(S)||S| = t, S \subseteq V\}$$

$$s.t.|S'| = t$$

- **Definition 2:** $\boldsymbol{u}_n$: A aggregate function f: $2^V \to R$ is monotonous, if $f(S) \leq f(T)$ satisfy for all $S \subseteq T$.
- **Definition 3:** $\boldsymbol{u}_n$: A aggregate function f: $2^V \to R$ is submodel, if $f(S \cup |u|) - f(S) \geq f(T \cup |u|) - f(T)$ satisfy for all $S \subseteq T \subseteq V \subseteq$ and $u \in V$.

B. k-core

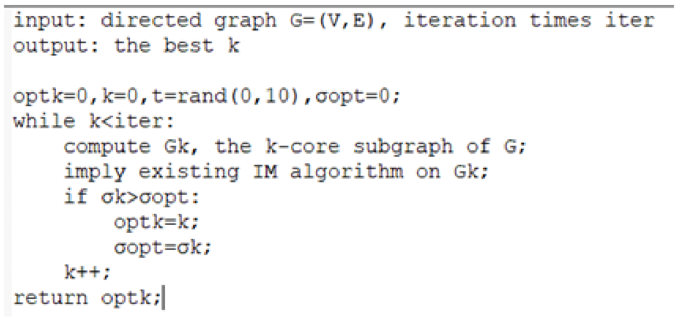
Any node v in the set $V_k \subseteq V$ has a degree which is not less than k. The maximum induced subgraph $G_k(V_k, E_k)$ which is inferred by V_k is called k-kernel.

Since the nodes which has large degree may be scattered, the k-core can gather these large degree nodes and then produce large network influence.

C. k-core filter algorithm

K-core filter algorithm is not a independent influence maximization algorithm. It combines with existing algorithm to enlarge the influence area and shorten the execution time. The basic thought is to pre-train k and find a constant k value which has best optimizing result. When given the node counts t, we can calculate the k-core of the graph and filtrate the unnecessary edges and nodes. Then we apply the influence maximization algorithm on the k-core subgraph and achieve the target that shorten the execution time.

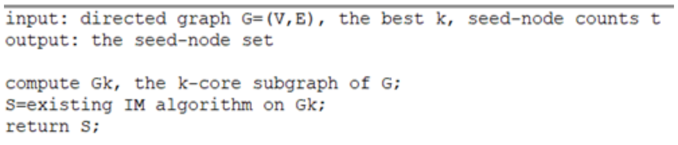
K-core filter algorithm has two procedures:1)pre-train the value k which can produce the best optimizing result. 2)calculate the k-core of the network and apply into existing algorithms.



```
input: directed graph G=(V,E), iteration times iter
output: the best k

optk=0,k=0,t=rand(0,10),σopt=0;
while k<iter:
    compute Gk, the k-core subgraph of G;
    imply existing IM algorithm on Gk;
    if σk>σopt:
        optk=k;
        σopt=σk;
    k++;
return optk;
```

Fig. 1. pre-train k



```
input: directed graph G=(V,E), the best k, seed-node counts t
output: the seed-node set

compute Gk, the k-core subgraph of G;
S=existing IM algorithm on Gk;
return S;
```

Fig. 2. k-core optimize existing algorithm

The pre-training procedure can find the best k. We can learn that when we choose different node counts t, we can get a constant k value which leads to a best optimizing value.

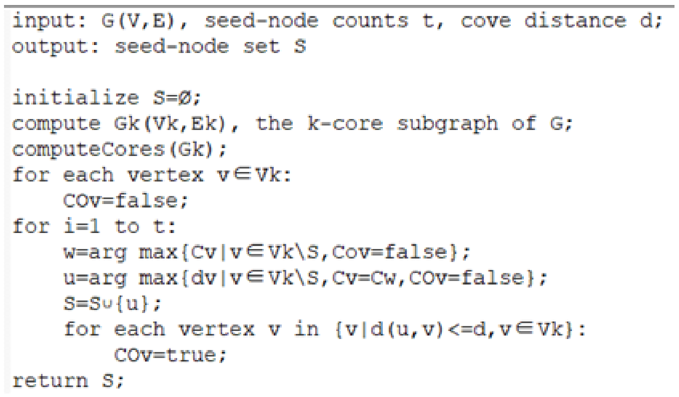
Then we use the seed nodes set S which is calculated by the second algorithm. The execution time will decrease a lot and can influence more area.

D. k-core optimizing existing algorithms

Among the existing influence maximization algorithm, PMIA and IRIE algorithm have better results. IRIE costs large memory and IRIE need to execute a long time. Here, we combine k-core filter algorithm with CCA and PMIA and get KCoreCCA and KCorePMIA. The new algorithms can spread farther and finished quickly.

1) KCoreCCA:

CCA algorithm is based on the the spread distance to choose t nodes which has biggest cores as seed nodes. Core k shows that the node belongs to k-core but not (k+1)-core. Pruning low-degree nodes will not influence the spread area but will shorten the execution time a lot.



```
input: G(V,E), seed-node counts t, cove distance d;
output: seed-node set S

initialize S=Ø;
compute Gk(Vk,Ek), the k-core subgraph of G;
computeCores(Gk);
for each vertex v∈Vk:
    COv=false;
for i=1 to t:
    w=arg max{Cv|v∈Vk\S,Cov=false};
    u=arg max{dv|v∈Vk\S,Cv=Cw,COv=false};
    S=S∪{u};
    for each vertex v in {v|d(u,v)<=d,v∈Vk}:
        COv=true;
return S;
```

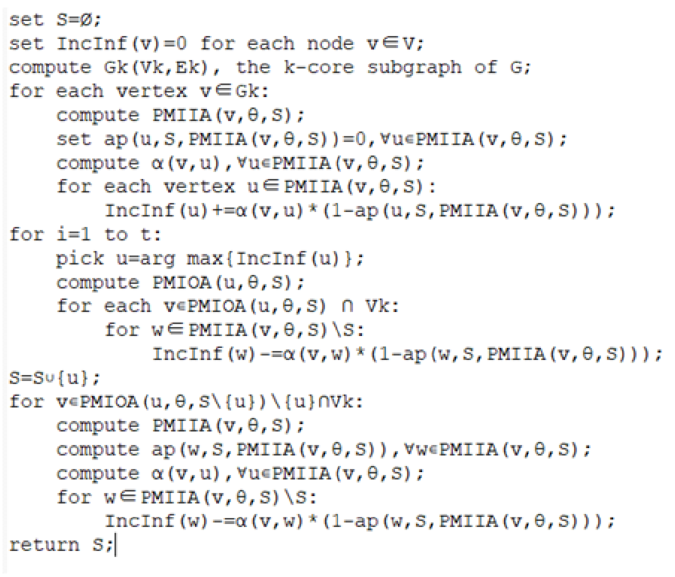
Fig. 3. KCoreCCA

C_v is the node cores, CO_v is the node cover property, $d_{u,v}$ is the distance between node u and v. This algorithm is to apply k-core filter algorithm into CCA. The time complexity of CCA is O(tm). Because of pruning many nodes and decrease calculated amount, the time complexity of KCoreCCA is $max\{O(m), O(tm_k)\}$.

2) KCorePMIA:

PMIA algorithm firstly calculate the local tree structure PMIIA and PMIOA of each node $\forall v \in V$. Based on the local tree structure PMIIA to calculate the influence $ap(u, S, PMIIA(v, \theta))$ caused by each node u. Then calculate the influence coefficient $\alpha(v, u)$ based on the influence linearity property and choose t nodes which have maximum influence. PMIA algorithm need to calculate the local tree structure of each node and it costs much memory. We use k-core filter algorithm to find the maximum influence nodes and then only need to calculate the local tree structure of these nodes. This can decrease much calculated amount.

The time complexity of PMIA algorithm is $O(nt_{i\theta} + tn_{o\theta}n_{i\theta}logn)$. $n_{i\theta}$ is the PMIIA local tree structure of all nodes. $n_{o\theta}$ is the PMIOA local tree structure of all nodes. $t_{i\theta}$



```
set S=Ø;
set IncInf(v)=0 for each node v∈V;
compute Gk(Vk,Ek), the k-core subgraph of G;
for each vertex v∈Gk:
    compute PMIIA(v,θ,S);
    set ap(u,S,PMIIA(v,θ,S))=0,∀u∈PMIIA(v,θ,S);
    compute α(v,u),∀u∈PMIIA(v,θ,S);
    for each vertex u∈PMIIA(v,θ,S):
        IncInf(u)+=α(v,u)*(1-ap(u,S,PMIIA(v,θ,S)));
for i=1 to t:
    pick u=arg max{IncInf(u)};
    compute PMIOA(u,θ,S);
    for each v∈PMIOA(u,θ,S) ∩ Vk:
        for w∈PMIIA(v,θ,S)\S:
            IncInf(w)-=α(v,w)*(1-ap(w,S,PMIIA(v,θ,S)));
S=S∪{u};
for v∈PMIOA(u,θ,S\{u})\{u}∩Vk:
    compute PMIIA(v,θ,S);
    compute ap(w,S,PMIIA(v,θ,S)),∀w∈PMIIA(v,θ,S);
    compute α(v,u),∀u∈PMIIA(v,θ,S);
    for w∈PMIIA(v,θ,S)\S:
        IncInf(w)-=α(v,w)*(1-ap(w,S,PMIIA(v,θ,S)));
return S;
```

Fig. 4. KCorePMIA

is the longest time that calculate the PMIIA local tree structure of all nodes. Because of the calculation of all nodes' PMIIA and PMIOA tree structure, the time complexity of PMIA must be larger than O(m). However, the KCorePMIA only need to calculate the local tree structure on the k-core subgraph. Its time complexity is $max\{O(m), O(n_k t_{i\theta k} + tn_{o\theta k}n_{i\theta k}logn_k)\}$

E. GI algorithm

The GI algorithm extends the algorithm model based on tree. It decrease the time complexity and ensure the influence area.

The path $P_{u,v} = (n_1 = u, n2, ..., n_m = v)$ is a path from node u to node v Under the independent cascade model, the probability of the node u activation node v by the path P is:

$$pp(P_{u,v} = \prod_{i=1}^{m-1} p(n_i, n_{i+1})$$

Node u can influence node v by many paths, we use the maximum influence path MPP(u,v) to stand for the probability of u influence v:

$$pp(u, v) = pp(MPP(u, v)) = pp(argmax(pp(P_{u,v})))$$

By calculate the influence probability of each node to node u, we can make a reverse propagation tree ITree(u). In a similar way, we can calculate the influence probability of node u to each other node. Then we can make maximum propagation tree Otree(u).

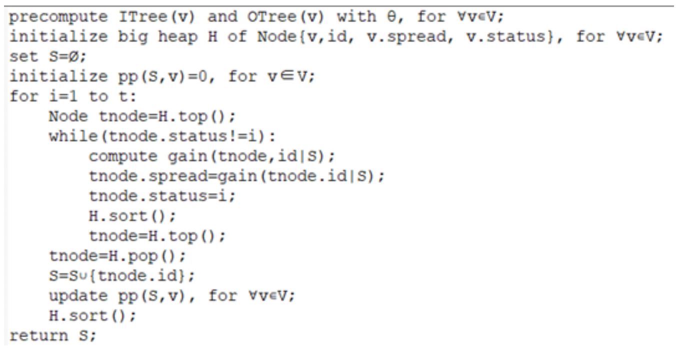
When add a node active node u in the seed node set S, the influence gain of v from seed node set gain(u,—S,v) is:

$$gain(u, |S, v) = pp(u, v)(1 - pp(S, v))$$

Suppose every node influence other nodes independently. Then the total influence gain after adding a new active node u into seed node set S gain(u—S) is:

$$gain(u|S) = \sum_{v \in V} pp(u,v)(1 - pp(S,v))$$

Based on the submodel property of influence maximization, we can choose the node which has maximum gain(u—S) into the seed node set S until we choose t nodes.



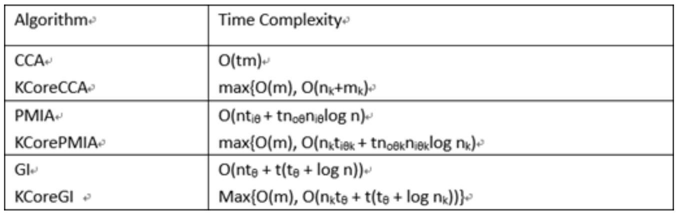
```
precompute ITree(v) and OTree(v) with θ, for ∀v∈V;
initialize big heap H of Node{v,id, v.spread, v.status}, for ∀v∈V;
set S=∅;
initialize pp(S,v)=0, for v∈V;
for i=1 to t:
    Node tnode=H.top();
    while(tnode.status!=i):
        compute gain(tnode,id|S);
        tnode.spread=gain(tnode.id|S);
        tnode.status=i;
        H.sort();
        tnode=H.top();
    tnode=H.pop();
    S=S∪{tnode.id};
    update pp(S,v), for ∀v∈V;
    H.sort();
return S;
```

Fig. 5.   GI

The time complexity of GI algorithm is $O(nt_\theta + t(t_\theta + logn))$. If we apply k-core filter algorithm on it and get KCoreGI algorithm. Then the time complexity is $max\{O(m), O(n_k t_\theta + t(t_\theta + logn_k))\}$

III. Experimental Evaluation

By analyzing the above algorithm, we can get the time complexity of every algorithm and k-core filter algorithm. We can find that the complexity decrease obviously.



Algorithm	Time Complexity
CCA	$O(tm)$
KCoreCCA	$max\{O(m), O(n_k+m_k)$
PMIA	$O(nt_{i\theta} + tn_{o\theta}n_{i\theta}log\ n)$
KCorePMIA	$max\{O(m), O(n_k t_{i\theta k} + tn_{o\theta k}n_{i\theta k}log\ n_k)$
GI	$O(nt_\theta + t(t_\theta + log\ n))$
KCoreGI	$Max\{O(m), O(n_k t_\theta + t(t_\theta + log\ n_k))\}$

Fig. 6.   Time complexity comparison of different IM algorithms and their k-core filtered versions

IV. Conclusion

Concerned on the influence maximization problem in social network, we propose a new GI algorithm. It can influence larger area and have less execution time. What's more, by k-core calculating the nodes which have small influence area and pruning them, we get a k-core filter algorithm which can enlarge the influence area of existing algorithms and shorten the time. However, because of lack of time and data, I don't continue the next experience. In the future, I will find some proper data set and execute the k-core filter algorithm to get the exact data. By the data, we can compare the different algorithm more intuitively.

V. References

[1] aaai Burst Time Prediction in Cascades

[2] aaai Distinguishing between Topical and Non-topical Information Diffusion

[3] aaai Extracting Influential Nodes for Information Diffusion on a Social Network

[4] aaai Learning User-specific Latent Influence and

[5] ijcai15 Maximizing the Coverage of Information Propagation in Social Networks

[6] Information Diffusion in Computer Science Citation Networks

[7] Uncover Topic-Sensitive Information Diffusion Networks