Physica A 420 (2015) 124–133

Contents lists available at ScienceDirect

Physica A

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

SMG: Fast scalable greedy algorithm for influence maximization in social networks

Mehdi Heidari*, Masoud Asadpour, Hesham Faili

School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

HIGHLIGHTS

- SMG changes the way Monte-Carlo simulation is calculated in greedy approach.
- SMG makes huge improvement in the times that nodes of graph should be traversed.
- SMG prevents recreation of graph instances needed for Monte-Carlo simulation.
- SMG brings scalability to any greedy method which uses Monte-Carlo simulation (sub-modular NP-Compelete problems like influence maximization).
- We show improvement of SMG in both time complexity and experiments.

ARTICLE INFO

Article history:
Received 8 September 2014
Available online 7 November 2014

Keywords:
Social networks
Influence maximization
Scalable greedy
State machine Monte-Carlo
Propagation model
Viral marketing

ABSTRACT

Influence maximization is the problem of finding k most influential nodes in a social network. Many works have been done in two different categories, greedy approaches and heuristic approaches. The greedy approaches have better influence spread, but lower scalability on large networks. The heuristic approaches are scalable and fast but not for all type of networks. Improving the scalability of greedy approach is still an open and hot issue. In this work we present a fast greedy algorithm called State Machine Greedy that improves the existing algorithms by reducing calculations in two parts: (1) counting the traversing nodes in estimate propagation procedure, (2) Monte-Carlo graph construction in simulation of diffusion. The results show that our method makes a huge improvement in the speed over the existing greedy approaches.

1. Introduction

The goal of the influence maximization problem is to find k nodes in a social network which, activating them, results in activation of the maximum number of individuals in the network. This problem has many applications especially in marketing, advertisement and elections. For example, in viral marketing, the goal is to maximize the number of individuals influenced via face to face advertisements of a particular product, in this case, k free samples of the product is given to k individuals hoping that they try it and upon satisfaction start recommending the product to their friends. If the product has rather better quality compared to its competitors it is expected that the recommendation circulate among friends and friends of friends, etc., and those people get convinced to buy it. The number of seed set, k, is limited to the budget considered for advertisement.

To solve the influence maximization problem in a network, we should have the influence cascade model, i.e. the way recommendations or influences are propagated from the receivers of free samples to their friends, and from friends to friends of friends, etc. The work by Kempe, Kleinberg and Tardos [1] provided the first formulation of influence maximization as an

* Corresponding author.
E-mail addresses: heidari_mehdi@ut.ac.ir (M. Heidari), asadpour@ut.ac.ir (M. Asadpour), hfaili@ut.ac.ir (H. Faili).

http://dx.doi.org/10.1016/j.physa.2014.10.088
0378-4371/© 2014 Elsevier B.V. All rights reserved.

optimization problem. Their work is based on two basic influence cascade models: the Independent Cascade (IC) model and the Linear Threshold (LT) model.

In both models, a social network is modeled as a weighted directed graph $G = (V; E)$, where the vertices of V represent individuals and edges in E represent relationships, and the orientations of the edges indicate the direction of influence. In the IC model each edge has an activation probability and influence is propagated by activated nodes independently; these nodes activate their inactive neighbors based on the activation probabilities of their edge. In the LT model, in addition to the weights on edges, each vertex has a threshold too, and a vertex will be activated if the sum of weights that it receives from its active neighbors exceeds its threshold.

2. Previous works

Domingos and Richardson [2] studied influence propagation as an algorithmic problem to optimize marketing decisions. They studied movie customers network and proposed a probabilistic model. They modeled customers network as a graph and used a Markov random field to calculate influence propagation among them. So they triggered algorithmic study of influence maximization.

Kemp et al. [1] in 2003 formulated the influence maximization as an optimization problem. They proved NP-hardness and sub-modularity of influence maximization under two presented models in their work. They used the greedy hill climbing algorithm as an approximate solution to this problem. The greedy approximation had already been proved that is $(1 - 1/e)$ approximation on sub-modular functions [3]. Their algorithm adds k nodes to answer vector S in k steps of greedy choice. In each step of seed selection the algorithm calls a propagation estimator function (F) for each node to calculate their marginal gain by Monte-Carlo simulation. After the marginal gain estimation, the node with maximum marginal gain is chosen and added to seed set S. In practice, the algorithm is too slow and is not scalable to large networks.

Leskovec et al. [4] proposed a lazy forward version of the greedy algorithm. Their algorithm try to reduce the number of marginal gain calculations when it is not needed. Adding the first node to seed set needs calling propagation estimator function for all n nodes of graph. But in the second seed-selection step, since marginal gains of the previous step is available and also since, according to sub-modularity, the marginal gain of a particular node does not increase in the next step, the next seed element could approximately be found without re-calculating marginal gain for the $n - 1$ remaining nodes. So, to improve the speed, this method targets the reduction in the number of calls to propagation estimator function F. Of course, there is no guarantee how much it could be reduced.

Chen et al. [5] improved the speed of greedy algorithm on Independent Cascade (IC) model. Since in the IC model each pair of vertices only coexists in a small portion of the graphs, there is no need to increase the number of simulations to compensate the correlation effect, so an improvement can be gained by reducing the number of simulations. Although this considerably improves the running time, but it only works for the IC model and not in general.

Chen [6] changed the way simulation is calculated in the greedy algorithm by using a threshold parameter to gain a local tree around the node which is being evaluated. So simulation could be performed locally. As the value of the threshold reduces, the local simulation result gets closer to global simulation. So this method is only an approximation of the greedy algorithm but not as exact as them.

Goyal [7] presented an algorithm called CELF++. This algorithm tries to reduce the recalculation of propagation estimator function by using a data structure which stores next step marginal gain for a particular node. But it is highly probable that the stored marginal gain has not been calculated based on the last selected seed. So, in this case the stored marginal gain does not help and should be calculated again in the next step. CELF++ could not considerably reduce recalculations of propagation. On the methodology section we describe how our algorithm reduces these recalculations.

There are many works that have targeted scalability by using heuristic methods. But, there is no approximation guarantee available for them. Although they are not as dependable as the greedy methods however, they are very fast. Most of them use graph centrality measures and assume those measures are proportional to the amount of influence [5,8–11]. Some heuristic methods work wiser and consider a discount to handle sub-modularity too [5,8]. The discount could be the probability of a node being activated by the seed set [6,8] or another measure which has a direct relation with that probability [5]. Out work is focused on the greedy approach and the heuristic approaches are not related to our work so we do not go into detail in this paper.

In the next section we present our method that improves the speed of greedy algorithms from 2 aspects: (1) it prevents recalculations of the propagation estimator function. Note that this aspect was targeted by CELF++ but it could not efficiently prevent recalculations and just had a small improvement. (2) It minimizes the amount of Monte-Carlo graph constructions. It should be mentioned that our method could be incorporated into all previous greedy methods.

3. Our proposed method

3.1. The general greedy algorithm

Algorithm 1 shows the general greedy algorithm. The outer loop iterates k steps. In each iteration the most influential node is found and added to seed set S. Finding the most influential node, needs the inner loop which iterates over candidate nodes and evaluate them by estimating their marginal gain. Marginal gain estimation is done by function F. F gets a set of nodes as input and estimates their influence propagation by Monte-Carlo simulation. The output of F is the number of nodes that are activated upon activation of the input set. Propagation of elements of $S \cup \{w\}$ minus propagation of elements of S gives marginal gain of candidate node w. After the inner loop, the node with maximum marginal gain is added to S.

Algorithm 1: The General Greedy Algorithm

```
$S \leftarrow \phi$ ;
$Selected \leftarrow \phi$ ;
$S_\sigma \leftarrow 0$ ;
for $i \leftarrow 1$ to $k$ do
    foreach node $w$ of seed candidates do
        $MarginalGain[w] \leftarrow F(S \cup \{w\}) - S_\sigma$ ;
    end
    $Selected \leftarrow \{w | MarginalGain[w] maximized\}$ ;
    $S = S \cup Selected$ ;
    $S_\sigma \leftarrow S_\sigma + MarginalGain[Selected]$ ;
end
```

Assume that the above algorithm is in a state where j nodes have already been added to the seed set S and we need to find the next most influential node. Adding a new node to S needs calculation of F(S $\cup$ $\{w\}$) for each remained node w and then finding the node with maximum marginal gain. The existing algorithms spend a huge amount of time in this step, as they recalculate propagation for all j nodes of S in addition to calculation of propagation for the next candidate, w. We will show how our algorithm called, State Machine Greedy (SMG), prevents these recalculations.

To understand which recalculations would happen, assume that we are in a state shown in Fig. 1, where two nodes have already been added to S and five another nodes have been activated by two seed nodes. To find the third seed node in order to add it to S, the algorithm needs to simulate the diffusion process and estimate the marginal gain of each candidate node, w. To calculate the marginal gain for the current w in Fig. 1 it calls F(S $\cup$ $\{w\}$). The general greedy algorithms traverse all nodes of the graph again and again. But if we save this state as the last state, then F have to traverse only the nodes in the Traversing Area and successfully estimate the marginal gain of w.

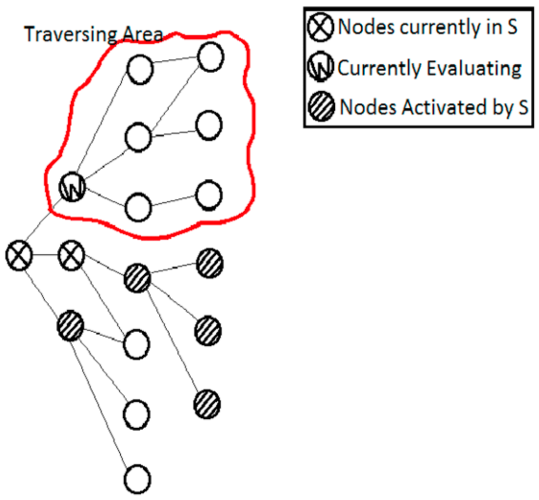

Fig. 1. Example of a Monte-Carlo graph saved as last state.

3.2. *State Machine Greedy*

The flow chart of our proposed method, State Machine Greedy, is presented in Fig. 2. It works as a single state machine which saves calculated propagation of i nodes of S in all Monte-Carlo graphs as last state and uses it to prevent recalculation of these propagation in the next step.

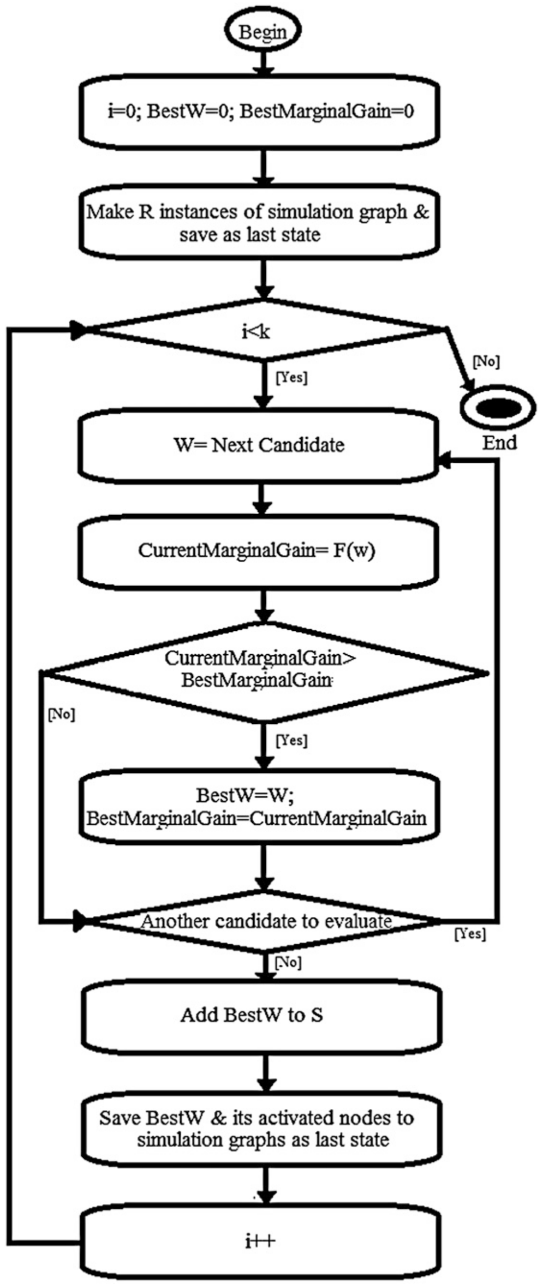
Begin
i=0; BestW=0; BestMarginalGain=0
Make R instances of simulation graph & save as last state
i<k
[No]
End
[Yes]
W= Next Candidate
CurrentMarginalGain= F(w)
CurrentMarginalGain> BestMarginalGain
[No]
[Yes]
BestW=W; BestMarginalGain=CurrentMarginalGain
Another candidate to evaluate
[Yes]
[No]
Add BestW to S
Save BestW & its activated nodes to simulation graphs as last state
i++

Fig. 2. Flowchart of SMG.

This state machine gets R graphs with inactive nodes at the beginning and evaluates their propagation in order to find the one with maximum marginal gain. The R graphs are generated in the Monte-Carlo simulation in order to sample the space of possible graphs. After adding the most influential node to S, the algorithm saves all activated nodes by the best node in each simulation graph as the last state and gets ready to find the next most influential node in the next step. The best node here is the node that can activate the maximum number of nodes upon its activation. As a result, in any step i, we have R graphs which have recorded propagation of all $(i-1)$ nodes in S as the last state. So to evaluate each new node, w, the function F only calculates the increased propagation of w instead of calculating propagation for $i+1$ elements in $S \cup \{w\}$. In the flow chart of SMG, shown in Fig. 3, the function F is called only with one node $\{w\}$ and not $S \cup \{w\}$. In addition, inside function F, the number of nodes which should be traversed is reduced again by knowing which nodes have been activated by S till now.

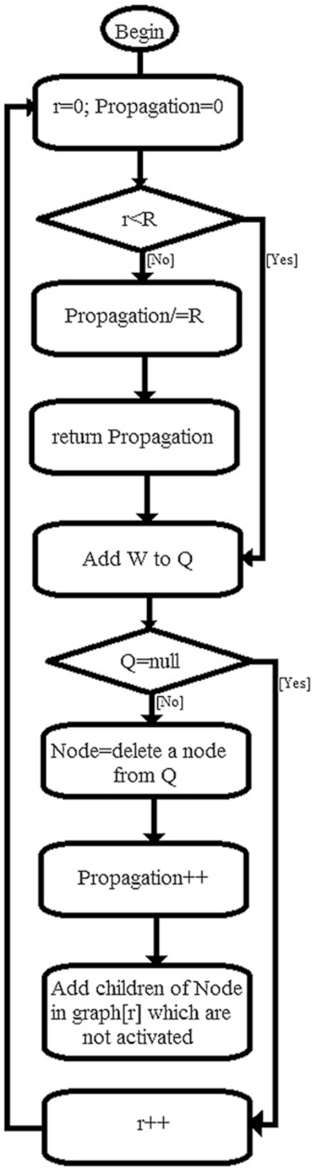

Fig. 3. Flowchart of function F in SMG.

Fig. 3 shows the flow chart of function F. Implementation of F varies depending on the propagation model. Here we present the flowchart based on the Independent Cascade Model.

3.3. Time complexity analysis

The idea presented as SMG can be incorporated to any existing greedy algorithm. To compare their running time we analyze them from two point of views: first, based on the number of nodes which are put in the startup queue by F, till their triggered propagation is calculated. The number of these nodes is equal to R multiplied by the number of all input nodes of F where R is the Monte-Carlo factor. We call it Complexity of Trigger Nodes. Second, based on the number of graph instances which is built for Monte-Carlo simulation at the total runtime of algorithm, we call it Complexity of Graph Construction. Our algorithm has an improvement in the number of nodes which are traversed by F too, but this cannot be shown by time complexity and we show it later by experiments.

3.3.1. Complexity of trigger nodes

In both algorithms, the number of iterations of the first loop is k. This is the number of elements which should be chosen as the final seed set (in each step one node is chosen under the greedy mechanism).

The number of iterations in the second loop in both algorithms is the same. This loop iterates on the nodes which are selected as a candidate for adding to the seed set. In each iteration one candidate (w) is evaluated by calculating marginal gain for it. The number of candidate nodes could be all n nodes of the graph, so time complexity of this loop is $O(n)$.

The function F is where our algorithm performs better. Inputs of F are what triggers propagation simulation in F. Stateless greedy algorithms call this function with i nodes as input parameters where i is the sum of elements already in S (in step i, S contains $i - 1$ nodes) plus the candidate node w which is being evaluated to find ith element of S. But our algorithm calls F with just one node as the input parameter and it is done by holding propagation of $i - 1$ elements in the seed set as the last state.

The last point to calculate aforementioned time complexity is the inner loop in function F. Each time F is called, it calculates propagation for its parameters R times, so all trigger nodes we have talked till now is multiplied by R. Now we can summarize these words in a table shown in Table 1.

Table 1
Time complexity of trigger nodes.

	SMG	Stateless greedy
Steps of seed selection	K	K
Candidated nodes in step i	O(N)	O(N)
Number of simulations	R	R
Trigger nodes of F in step i	1	i
Trigger nodes complexity	$O(K * N * R)$	$O(K*N*R*(K+1)/2)$

3.3.2. Complexity of graph construction

SMG builds R instances of graph at the beginning and saves them to memory as the primary state. Then in each step of greedy choice, it uses the same graphs to find the most influential node of that step, then saves marginal gain of the new seed and use them as the last state for the next step. So SMG totally builds $O(R)$ instance of simulation graphs, but other greedy algorithms make R graphs each time they call function F. Since, function F is called $O(K * N)$ times so they totally build $O(K * N * R)$ instances of simulation graphs.

As a short conclusion, State Machine Greedy has $O(k)$ times less complexity in trigger nodes and $O(K * N)$ times less complexity in graph construction. Note that in addition to these two improvement, saving last state helps to reduce total traversing nodes. As it is shown in Fig. 1, function F just traverses the nodes in the traversing area. This restricted area is formed by knowing which node has been activated by S and should not be added to the traversing queue. Amount of this third type of improvement depends on the network and could not be formulated easily. Instead, we show the average number of nodes which have been traversed by F in the experiment section.

4. Experiments

For experiments we used Arxiv and Wiki-Vote graphs available in SNAP Dataset's page (http://snap.stanford.edu/). Arxiv is an undirected graph containing co-authorship graph of authors whose papers have been published in Quantum Cosmology and General Relativity Categories. Nodes represent authors and the edge between two nodes represents their contribution in at least one paper. For example if 3 authors have contributed in one paper they form a complete sub graph of 3 nodes. Wiki-Vote is a directed graph containing voting network of Wikipedia editors. Wikipedia common editors can request to upgrade their role to administrator, the acceptance of their request is specified by voting among other editors. Nodes of Wiki-Vote represent editors and a directed edge from node i to node j means node j has been voted by node i. Some structural characteristics of the two described datasets are shown in Table 2.

Table 2
Structural characteristics of the datasets.

	Arxiv	Wiki-Vote
Nodes	5242	7115
Edges	14 496	103 689
Diameter	17	7
Number of triangles	48 260	608 389
Fraction of closed triangles	0.36	0.04
Clustering coefficient	0.53	0.14

Tests are implemented using the Independent Cascade (IC) model. In this model edges are directed and weighted. Edge weights are probabilities which specify the probability that the source node – if it is already active – activates the target node. Activation of an edge is independent from the other edges. So, to simulate the diffusion process, it is easier to decide in advance which edges are active and which are inactive based on the probability denoted by the edge weights. This way the dynamic process of diffusion is changed to a static process. In order to have a good sample of the outcomes of the possible graphs, R different networks are generated in which, as mentioned before, existence of an edge depends on the outcome of a Bernoulli trial. If it wins then the edge is created between the source and the target nodes. Now, outcome of the simulation of the diffusion process is specified by counting the number of active nodes. A node is active either if it belongs to the active seed set or there exists a path from any node in the seed set to it. So to calculate the propagation from a seed set we just keep edges which have won the Bernoulli trial and then count nodes of graph reachable from seed set. Using IC in our tests does not mean State Machine Greedy could not be adjusted to other diffusion models. It can be used with any diffusion model. Note that complexities calculated in the methodology section are also independent of the diffusion model.

The datasets we use for our experiments are not weighted. We use Weighted Cascade model to assign weights to the edges; in this model, we assign weight $w_{i,j} = 1/inDeg(j)$ to edge $E_{i,j}$, where j is the target node of the edge and $inDeg(j)$ is the input degree of j (see Fig. 4 for an example).

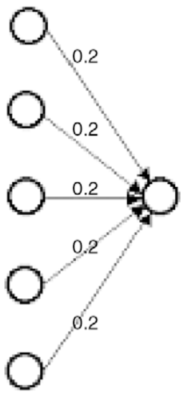

Fig. 4. Example of Weighted Cascade model.

The comparison of running time of CELF and SMG on two Arxiv and Wiki-Vote datasets is presented in Figs. 5 and 6. The horizontal axis shows seed set size ($K = 1, \ldots, 50$) and the vertical axis shows the running time of algorithms in minutes. It is seen that the running time of SMG is negligible compared to CELF. Note that the vertical axis is in the logarithmic scale. This makes SMG applicable to larger problems.

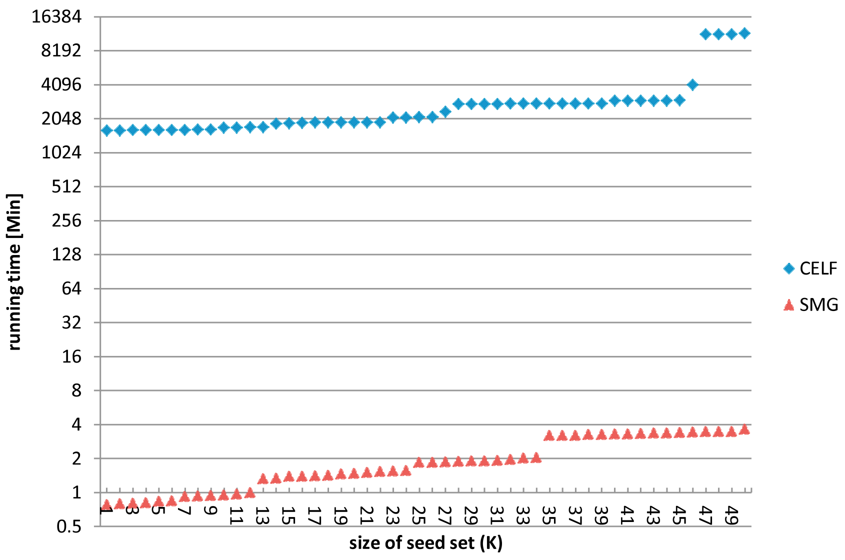

Fig. 5. Running time on the Arxiv dataset.

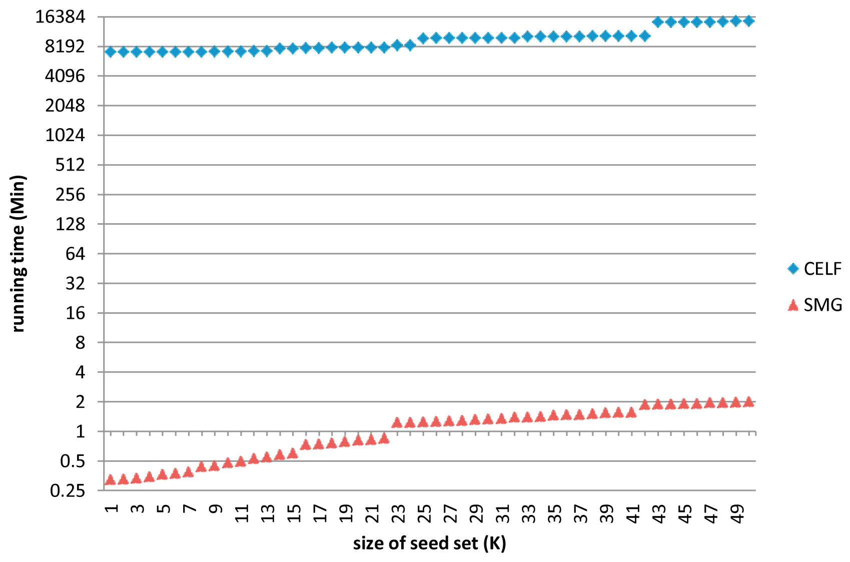

Fig. 6. Running time on the Wiki-Vote dataset.

Table 3 shows the average running time of CELF and SMG when Monte Carlo simulation is done with $R = 2500$ samples and the average has been obtained on 50 different seed set size from 1 to 50. The average running time for SMG is around two minute on Arxiv and one minute on Wiki-vote which is quite astonishing compared to CELF that needs more than two days to complete.

Table 3
Average running time of CELF and SMG.

Dataset	Running time (min)	
	CELF	SMG
Arxiv	2987	2
Wiki-Vote	9629	1

Another interesting measure is the number of nodes that are traversed for diffusion simulation. This gives a sense on the complexity of the algorithms in action. To obtain this we calculate the average number of nodes which are put in the queue by F in order for their influence propagation to be calculated. This measure is calculated by counting the number of nodes traversed by F divided by the total number of times F is called. Figs. 7 and 8 show this average for different seed set sizes.

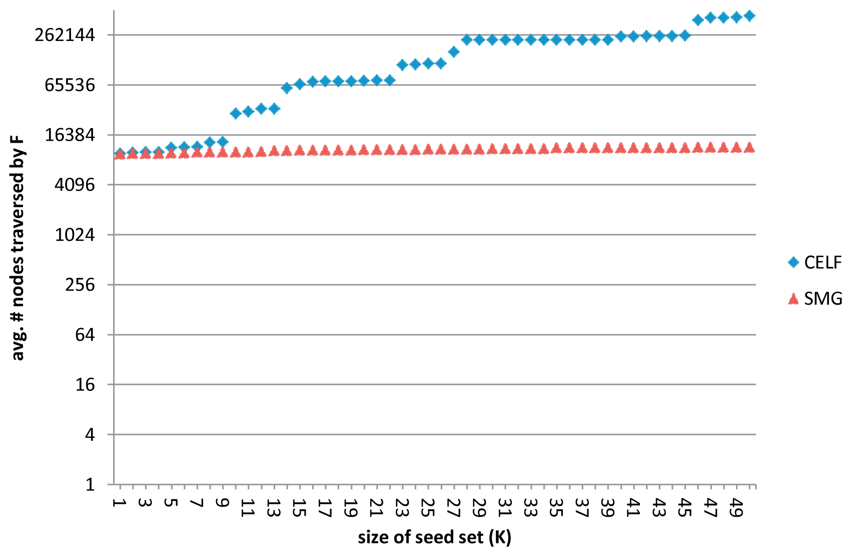

Fig. 7. Average traversed nodes by F on the Arxiv dataset.

At the beginning when the seed set is small the difference between the two algorithms is negligible. But as the seed set size gets larger the difference becomes bigger. Here is where saving the state of the graph helps a lot in reducing the burden of marginal gain estimation. Note that the vertical axis is in the logarithmic scale. Complexity for SMG grows with a very slow rate such that it is almost a horizontal line in the logarithmic scale. This means SMG is scalable to bigger problems.

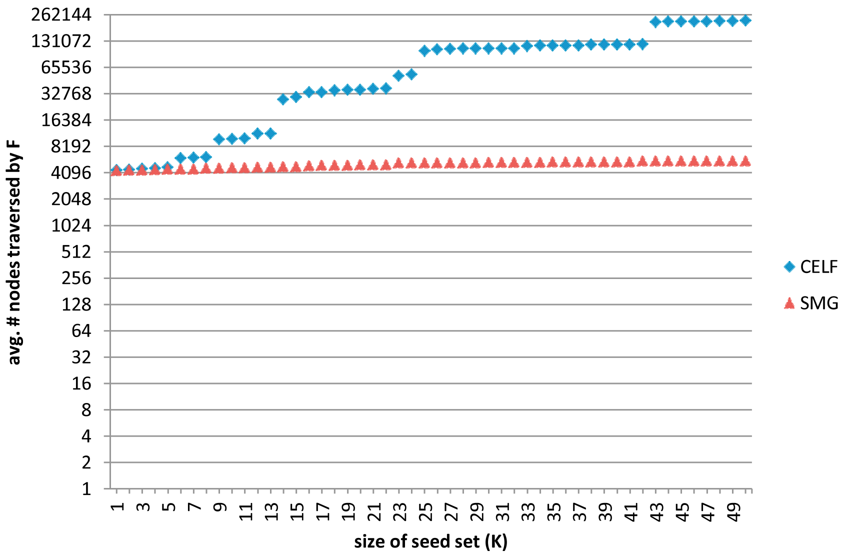

Fig. 8. Average traversed nodes by F on the Wiki-Vote dataset.

Overall speaking, SMG works great however there are some minor issues that should be noted here. The amount of improvement depends on to the structure of dataset and the amount of prevention in node re-traversing. As seen in Table 3, improvement of our method over CELF is bigger for Wiki-Vote. This is partially due to the fact that Wiki-Vote has more nodes and edges than Arxiv. So, when the algorithm is in step i, $(i - 1)$ nodes are in S and when CELF calls F it should traverse all nodes reachable from i nodes of $S \cup \{w\}$. So, there are many more nodes getting retraversed by CELF in Wiki-vote than Arxiv. So, SMG can have more improvement by preventing retraversals on Wiki-Vote.

Another reason for the improvement of SMG over CELF is the structure of the network. An evidence can be obtained by comparing Fig. 7 with Fig. 8 and observing that the average number of traversed nodes by SMG is higher in Arxiv than Wiki-Vote. Taking a look at the structural measures of datasets in Table 2 shows that Arxiv has a quite bigger clustering coefficient. This means Arxiv has more closed triangles and denser communities. This could cause more node traversal in total (even when there is no node re-traversal in SMG) because nodes in dense communities are very close and directly reachable from other nodes of that community. When F is estimating propagation of a particular node w it has to traverse most of the nodes of the community that w belongs.

5. Conclusion

In this work we presented the State-Machine Greedy algorithm that solves the important influence maximization problem. We showed SMG highly improves the speed of greedy algorithms by preventing recalculations done by older methods. We showed that by saving the last state of simulations at the end of each step of greedy choice and using it to efficiently prevent recalculations in estimation of marginal gain for candidate nodes in selection of the next seed, a great improvement in running time of the greedy algorithms is obtained. We showed SMG improves previous greedy algorithms in three aspects: first, it improves the time complexity of trigger nodes which are put in the startup queue by F, from $O(K * K * N * R)$ to $O(K * N * R)$. Second it reduces the time complexity of graph construction from $O(K * N * R)$ to $O(R)$. And third, it is preventing retraverse of nodes which are already activated by S. The improvement made by the third aspect was shown in experiments. We compared SMG with CELF in experiments and our empirical studies on two datasets shows that SMG greatly improves both running time and the average number of nodes traversed by F.

For future we would like to study the effect of structure on the time complexity of influence maximization algorithms and try to invent some heuristics and tricks to reduce the amount of calculations.

References

[1] D. Kempe, J. Kleinberg, E. Tardos, Maximizing the spread of influence through a social network, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 137–146.

[2] P. Domingos, M. Richardson, Mining the network value of customers, in: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, pp. 57–66.
[3] G. Cornuejols, M. Fisher, G. Nemhauser, Exceptional paper-location of bank accounts to optimize float: an analytic study of exact and approximate algorithms, Manage. Sci. (1977) 789–810.
[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, N. Glance, Cost-effective outbreak detection in networks, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp. 420–429.
[5] W. Chen, Y. Wang, S. Yang, Efficient influence maximization in social networks, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 199–208.
[6] W. Chen, C. Wang, Y. Wang, Scalable influence maximization for prevalent viral marketing in large-scale social networks, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 1029–1038.
[7] A. Goyal, L. Amit, L.V. Lakshmanan, CELF++: optimizing the Greedy algorithm for influence maximization in social networks, in: Proceedings of the 20th International Conference Companion on World Wide Web, 2011, pp. 47–48.
[8] K. Jung, W. Heo, W. Chen, IRIE: a scalable influence maximization algorithm for independent cascade model and its extensions, 2011. Arxiv Preprint.
[9] M. Kimura, K. Saito, Tractable models for information diffusion in social networks, in: Knowledge Discovery in Databases: PKDD, Springer, Berlin, Heidelgerg, 2006, pp. 259–271.
[10] R. Narayanam, Y. Narahari, A Shapley value-based approach to discover influential nodes in social networks, IEEE Trans. Autom. Sci. Eng. (2010) 1–18.
[11] A. Goyal, W. Lu, L.V.S. Lakshmanan, SIMPATH: an efficient algorithm for influence maximization under the linear threshold model, in: IEEE 11th International Conference on Data Mining, ICDM, 2011, pp. 211–220.