Simulated Annealing Based Influence Maximization in Social Networks

Qingye Jiang[†], Guojie Song[†*], Gao Cong[‡], Yu Wang[†], Wenjun Si[†], Kunqing Xie[†]
[†]Key Laboratory of Machine Perception, Ministry of Education, Peking University, China
[‡]School of Computer Engineering, Nanyang Technological University, Singapore
{jzjmail@sina.com, gjsong@pku.edu.cn, gaocong@ntu.edu.sg}

Abstract

The problem of influence maximization, i.e., mining top-*k* influential nodes from a social network such that the spread of influence in the network is maximized, is NP-hard. Most of the existing algorithms for the problem are based on greedy algorithm. Although greedy algorithm can achieve a good approximation, it is computational expensive. In this paper, we propose a totally different approach based on Simulated Annealing(SA) for the influence maximization problem. This is the first SA based algorithm for the problem. Additionally, we propose two heuristic methods to accelerate the convergence process of SA, and a new method of computing influence to speed up the proposed algorithm. Experimental results on four real networks show that the proposed algorithms run faster than the state-of-the-art greedy algorithm by 2-3 orders of magnitude while being able to improve the accuracy of greedy algorithm.

Introduction

One important function of a social network is to carry the spread of information by the "word-of-mouth" communication (Ma et al. 2008). It is a fundamental issue to find a small subset of influential individuals in a social network such that they can influence the largest number of people in the network. Finding a subset of influential individuals has many applications. For example, consider a social network that performs as the platform for marketing (Kempe, Kleinberg, and Tardos 2005). A company plans to target a small number of "influential" individuals of the network by giving them free samples of a product, expecting that the selected users will recommend the product to their friends, their friends will influence their friends' friends and so on, thus many individuals will ultimately adopt the product through the powerful word-of-mouth effect (or called viral marketing).

Formally, the problem is called as *influence maximization*, which is, for a parameter *k*, to find a *k*-node set with the maximum influence, where influence is propagated in the network according to a stochastic cascade model (Kempe, Kleinberg, and Tardos 2005).

*Corresponding author. Email: gjsong@cis.pku.edu.cn

The existing methods of finding the top-*k* problem mainly focus on the greedy algorithm and its enhancements. Kempel et al.(Kempe, Kleinberg, and Tardos 2005) formulate the top-*k influence maximization* problem as an combination optimization problem and establishes that the problem is NP-hard. They propose to use the hill-climbing greedy algorithm with a $(1 - \frac{1}{e})$ approximation ratio. New enhancements based on the greedy algorithm are proposed, e.g. the CELF algorithm (Leskovec et al. 2007), the NewGreedy algorithm (Chen, Wang, and Yang 2009), and the community-based greedy algorithm(CGA) (Wang et al. 2010). Although these enhancements greatly improve the efficiency of the hill-climbing Greedy algorithm, using greedy algorithm for the *influence maximization* problem in a large network is still computationally challenging. (as to be shown in our experiments)

Diverging from previous proposals on mining top-*k* influential nodes that use greedy algorithm, in this paper, we propose a new algorithm based on the Simulated Annealing(SA) algorithm to find the top-*k* influential nodes. To the best of our knowledge, this is the first work utilizing the SA algorithm for the *influence maximization* problem. To further improve the efficiency of the basic algorithm, we propose to replace the diffusion simulations of a node set with its EDV(expected diffusion value) according to the properties of social networks. We also propose the single-node spreading heuristic(SH) to generate better solution sets and accelerate the algorithm's convergence process. Further, we combine the single-node spreading heuristic and EDV to integrate their merits in efficiency and accuracy, respectively.

Extensive experiments are conducted and we report a summary of results, which show that the proposed algorithms based on the SA are capable of outperforming the state-of-the-art greedy algorithm in terms of both efficiency and accuracy.

Related Work

The influence maximization problem is first proposed by Domingos and Richardson (Domingos and Richardson 2001). Kempe et al. (Kempe, Kleinberg, and Tardos 2003) investigate this problem on two representative diffusion models, *the independent cascade* (IC) model and *the linear threshold* (LT) model. In this work and their subsequent work, (Kempe, Kleinberg, and Tardos 2005), they

generalize the two models and prove the influence maximization problem is a *NP-hard* problem.

Kempe et al. (Kempe, Kleinberg, and Tardos 2003) propose to use the hill-climbing Greedy algorithm to solve the top-*k* problem for the first time. They empirically compare hill-climbing Greedy algorithm with the degree heuristic and centrality heuristic algorithm, and find that hill-climbing has a much better accuracy although it runs much slower than the two simple heuristic algorithms. However, the greedy algorithm is very expensive. Most of the subsequent proposals on mining top-*k* influential nodes are based on the greedy algorithm and improve the greedy algorithm to achieve better efficiency. Leskovec et al. (Leskovec et al. 2007) propose the CELF algorithm that ameliorates the greedy algorithm by utilizing the submodular property in influence diffusion. Chen et al. (Chen, Wang, and Yang 2009) advance a new Greedy algorithm called NewGreedy algorithm. The reported experimental results show that NewGreedy significantly outperforms CELF algorithm. Wang et al. (Wang et al. 2010) propose CGA algorithm that invokes greedy algorithm with respect to community. Although the greedy algorithm based proposals are able to achieve good accuracy, they are very slow on large social network.

In addition, Chen et al. (Chen, Wang, and Yang 2009) also presents a degree discount heuristic algorithm called DegreeDiscount, which assumes that the influence spread increases with the degree of nodes. The heuristic algorithm is very efficient. However, its accuracy can be much lower than that of greedy algorithm. Chen et al. (Chen, Wang, and Wang 2010) target at general IC model with nonuniform propagation probabilities, which is different from the IC models used in the other proposals.

A salient feature of the proposed SA based algorithms in this paper is that they are capable of outperforming the best greedy algorithm in terms of both efficiency and accuracy.

Proposed Method

Table 1: Notations

Notations	Descriptions
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A network with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$
$\overrightarrow{uv}$	the edge from node *u* to *v*
M	number of edges in $\mathcal{G}$
N	number of nodes in $\mathcal{G}$
k	size of nodes to be mined
p	propagation probability of IC model
$\sigma(A)$	the number of nodes that node set *A* can influence in a network

Problem Statement

Table 1 gives the important notations used in this paper. Influence maximization problem targets to find a set of *k* nodes $A = \{v_1, v_2, \ldots, v_k\}$ such that $\sigma(A)$ is maximized according to a diffusion model. We adopt Independent Cascade(IC) model widely used in previous work (Kempe, Kleinberg, and Tardos 2003). In the model, the state of a node in a social network is either *active* or *inactive*. Active nodes are able to influence their inactive neighbors. The state of a node can be switched from being inactive to being active, but not vice versa. The model has a parameter called *propagation probability p* , which models the tendency of individuals to be affected by its neighbors.

The diffusion mechanism of IC can be described as follows. The diffusion process begins with an initial set of active nodes A at round t=0. Let $S_0 = A$. At each round t, an active node v_i from the last round S_{t-1} will be given a single chance to influence each of its inactive neighbors v_j, with a propagation probability *p*. If v_j is influenced, it is activated and is added to set S_t. The process terminates when S_t is empty. The set of nodes influenced by A is the union of S_t generated at each round. We denote the number of nodes influenced by A as $\sigma(A)$.

Introduction to Simulated Annealing (SA)

Simulated Annealing is an Intelligent Algorithm proposed by Metropolis et al.(Metropolis et al. 1953). It simulates the process of metal annealing and optimizes the solutions of a number of NP-hard problems, e.g., Traveling Salesman Problem. SA algorithm works as follows:

1) It creates an initial solution *i* and an initial system temperature $T = T_0$, and calculates the fitness of the solution, denoted by $f(i)$, which stands for the initial energy of the system;

2) Next, it searches the neighbor solutions of the current solution and a new solution *j* is created. If $\Delta f = f(j) - f(i)$ is negative, then the new solution is a better one and will replace the current one *i*; otherwise, the new solution will replace the current one with a possibility $p_{i,j} = exp(\frac{-\Delta f}{T})$(Metropolis criterion), where T is the current system temperature. The replacement mechanism is to minimize the energy state of the system. The initial temperature T_0 must be set large enough. After a number of iterations of searching for the neighbor solutions, we will cut down the system temperature $T = T - \Delta T$ as the solution is mended. When the temperature reaches the termination temperature T_f, the algorithm stops.

SA based Algorithm for Influence Maximization

The SA algorithm is outlined in Algorithm 1. We define the fitness function of a solution set $A \subset \mathcal{V}$ as $\sigma(A)$, the number of nodes that A will influence using the IC model. The fitness function $\sigma(A)$ characterizes the diffusion quality of set A. The algorithm has two levels of iterations: the outer level is controlled by T_f and ΔT, and the inner level is controlled by q. We create an initial set $A = \{v_1, v_2, \ldots, v_k\}$ randomly (line 2). In each iteration, we get *A*'s **neighbor solution** *A'* by replacing one node in solution set *A* with a node in $\mathcal{V} - A$ (line 5). If $\Delta(f) = \sigma(A') - \sigma(A)$ is positive, i.e., the new solution is better, the new solution A' is accepted (line 9). Otherwise, the new solution is accepted if $min\{1, exp(\Delta(f)/t)\} > random[0, 1]$ (lines 11-13). When the number of inner iterations q is reached (line 15), the algorithm updates the outer loop parameter T_t.

It is critical to make sure that the SA-based algorithm for top-*k* mining problem converges.

LEMMA 1 *The proposed SA based algorithm for influence maximization problem will converge towards the optimum as the iteration number t becomes larger.*

Algorithm 1 SA: SA based Top-*k* mining algorithm

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, size *k*, initial temperature T_0, termination temperature T_f, the number of inner loop *q*, the amount to cut down the current temperature in the outer loop ΔT
Output: the set of top-*k* influential nodes *A*;

```
1: t ← 0,T_t ← T_0, count ← 0;
2: Select an initial seed set A ⊂ V, |A| = k, randomly ;
3: while T_t < T_f do
4:    calculate σ(A)
5:    A' ← F(A, G); {create a neighbor solution set }
6:    count ← count + 1;
7:    calculate the change of the fitness Δf ← σ(A') − σ(A);
8:    if Δf > 0 then
9:       A ← A';
10:   else
11:      create a random number ξ ∈ U(0, 1).
12:      if exp(Δf / T_t) > ξ then
13:         A ← A' ;
14:   if count > q then
15:      T_t ← T_t − ΔT, t ← t + 1, count ← 0
16: return A
```

PROOF. We first observe the Markov Chain corresponding to the top-*k* influence maximization problem: we define a state *i* as a result set A_i containing *k* nodes in network $\mathcal{G}$. The union of A_i's neighbor sets are represented by $N(i)$. For a state $j \in N(i)$, the probability of generating *j* is $g_{i,j} = \frac{1}{k*(N-k)}$, and the probability of accepting *j* is $a_{i,j} = min\{1, exp[-(\sigma(j) - \sigma(i))/T(t)]\}$. Thus, the transition probability from state i to state j is

$$\forall i, j, p_{i,j}(t) = \begin{cases} g_{i,j}a_{i,j}(t), & j \in N_i \text{ and } j \neq i \\ 0, & j \notin N_i \text{ and } j \neq i \\ 1 - \sum_{k \in N_i} p_{i,k}(t) , & j = i \end{cases}$$

The finite state Markov Chain corresponding to SA top-k algorithm fulfills the following conditions:
(1) $\forall i, j \in \Omega$(states union),$g_{i,j}(t)$ is not related to t, and $g_{i,j} = g_{j,i}$, while $\exists n \geq 1, s_0, s_1, ...s_n \in \Omega, s_0 = i, s_n = j$, to make $g_{s_k, s_{k+1}}(t) > 0, k = 0, 1, ..., n-1$;
(2) $\forall i, j, k \in \Omega$, if $\sigma(i) \leq \sigma(j) \leq \sigma(k)$, then $a_{i,k}(t) = a_{i,j}(t)a_{j,k}(t)$;
(3) $\forall i, j \in \Omega, t > 0$, if $\sigma(i) \geq \sigma(j)$, then $a_{i,j}(t) = 1$; if $\sigma(i) < \sigma(j)$, then $0 < a_{i,j}(t) < 1$.

Thus, following the work (Mitra, Romeo, and Vincentelli 1985), for the stationary distribution of the Markov Chain $v = \{v_1, v_2, ..., v_N\}$

$$\lim_{t \to 0} v_i(t) = \begin{cases} \frac{1}{|\Omega_{opt}|}, & i \in \Omega_{opt} \\ 0, & i \notin \Omega_{opt} \end{cases}$$

Ω_{opt} is the union of optimal solutions. This means our algorithm will tend to converge to optimum□.

Remark: The proposed SA algorithm can escape the local optimum and is able to learn to improve the influence spread of solution set automatically. It allows to replace a solution with a worse one, which is different from methods in greedy algorithm. Compared with greedy algorithm, SA can effectively cut down the number of simulations.

Optimization

The computation cost of the SA algorithm is $O(TRM)$, where T is the number of iterations, R is the number of simulations to compute $\sigma(A)$, and M is the number of edges. We will optimize the algorithm's efficiency in two aspects: First we propose SAEDV algorithm to improve the simulation cost $O(RM)$; Second we propose SASH algorithm to cut down the iteration T. We combine SAEDV and SASH to get another algorithm MSA.

Simulation Cost Reduction (SAEDV)

To compute $\sigma(A)$ for a solution set $\mathcal{A}$, the existing influence maximization algorithms need *R* times simulations to compute the average influence of a solution set. This is computationally expensive.

We propose Expected Diffusion Value(EDV) to replace the diffusion simulations when computing $\sigma(A)$ for a solution set $\mathcal{A}$ to cut down the computation cost. We estimate the influence spread of a target set *A* containing *k* nodes in the Simulated Annealing, instead of using spreading simulations for *R* times.

Let $NB(A) = \{w | w \in A\} \cup \{v | \exists w \in A, \overrightarrow{wv} \in \mathcal{E}\}$, $NB(A)$ represents the one-hop area of the node set A. Let $r(v) = |\{w | w \in A, \overrightarrow{wv} \in \mathcal{E}\}|$. We extract $NB(A)$ and the edges between the nodes in $NB(A)$ to make a subgraph. Then we have the following lemma.

LEMMA 2 *Given a small propagation probability* p *in the IC model, the expected number of nodes influenced by node set* A *is estimated by*

$$k + \sum_{v \in NB(A) - A} (1 - (1 - p)^{r(v)})$$

PROOF. For each node $v \in NB(A) - A$, the probability that *v* will not be affected by the target set is $(1-p)^{r(v)}$, so we can see that *v* will finally be affected directly by *A* with a probability $1 - (1-p)^{r(v)}$. The k nodes in A are sure to be activated. So the final expected activated nodes' number in the one-hop area is $k + \sum_{v \in NB(A)-A} (1 - (1-p)^{r(v)})$.□

In the Simulated Annealing algorithm, we can adopt $EDV(A) = \sum_{v \in N(A)-A} (1 - (1-p)^{r(v)})$ as the fitness function $\sigma(A)$ instead of the average influence spread of *A* that is computed by *R* times of simulations. The algorithm for calculating EDV is outlined in Algorithm 2. We first detect $NB(A)$. $\forall v \in \mathcal{V}$, we initialize $r(v)$(This only needs to be done once when it is combined with SA). When we replace node w_1 with node w_2 to create a new solution set, we search all the edges starting from w_1, and for each end node v we cut down $r(v)$ by 1 (line 10). For each edge starting form w_2, we find its end node v having an edge starting from w_2 and increase $r(v)$ by 1 (line 15). During the process, we will also change $NB(A)$(lines 9 and 14). The new EDV is finally calculated (line 16).

The SA algorithm using the function in Algorithm 2 is called SAEDV. The complexity of this algorithm is $O(Tk\overline{d})$, where $\overline{d}$ is the average degree of the network.

Algorithm 2 SAEDV: function $\sigma(A', A, w_1, w_2)$ to compute the Expected Diffusion Value (EDV) of a node set A'

Input: network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node set A, a neighbor solution set A' of A, the discarded node w_1 (i.e.$A \setminus A'$) in A to generate A' and the new node w_2 (i.e.$A' \setminus A$);
Output: EDV of a neighbor solution set A' derived from A;

1: $NB(A) = \{w|w \in A\} \cup \{v|\exists w \in A, \overrightarrow{wv} \in \mathcal{E}\}$;
2: **for each** $v \in \mathcal{V}$ **do**
3: **if** $v \in NB(A) - A$ **then**
4: $r(v) = |\{w|w \in A, \overrightarrow{wv} \in \mathcal{E}\}|$;
5: **else**
6: $r(v) = 0$
7: **for each** $\overrightarrow{w_1 v} \in \mathcal{E}$ **do**
8: **if** $r(v) = 1$ **then**
9: $NB(A) \leftarrow NB(A) - v$
10: $r(v) \leftarrow r(v) - 1$;
11: $r(w_2) \leftarrow 0$
12: **for each** $\overrightarrow{w_2 v} \in \mathcal{E}$ **do**
13: **if** $r(v) = 0$ **then**
14: $NB(A) \leftarrow NB(A) \cup v$
15: $r(v) \leftarrow r(v) + 1$;
16: $\sigma(A') \leftarrow k + \Sigma_{v \in NB(A) - A'}(1 - (1-p)^{r(v)})$;
17: return $\sigma(A')$;

Speeding up Convergence (SASH)

SAEDV improves the efficiency of SA by using EDV to estimate the real spreading ability of a node set. However it may reduce the accuracy of SA. Algorithm 1 randomly replaces an element in current solution set to create a new solution set. This motivates us to develop heuristic methods to speed up the convergence of SA algorithm. We proceed to present Algorithm SASH using two heuristic methods.

Distant Node Heuristic We observe that in the top-*k* nodes returned by SA algorithm, the probability that two nodes have very short distance is very small. Hence, when choosing a new element to construct a new solution set A', we disregard the nodes whose shortest path from nodes in the current solution set A are smaller than a threshold d (we set d as 2 in our experiments). Let $dis(w, v)$ be the length of the shortest path from w to v. Let $M(A, d) = \{w|w \in \mathcal{N} - A, \forall v \in A, dis(w, v) \leq d\}$. We do not consider the nodes in $M(A, d)$ when we create a new solution set.

Single Node Spreading Heuristic For each node v_i, we compute its influence spread $\sigma(\{v_i\})$. When choosing a new node to construct neighbor solution set A', we take into account the influence spread of single node so that the algorithm will find a better solution more quickly.

The function of constructing a neighbor set A' of A using the two heuristics is outlined in Algorithm 3. Given a node set A, Algorithm 3 first calculates the individual influence spread of all the nodes in $\mathcal{G}$ (which is needed the first time when the algorithm is invoked). Then we randomly select a node in A and replace it with another node in $\mathcal{V} - A$ based on the roulette of influence spread of single node(lines 6-9). The SA algorithm using the function in Algorithm 3 is called SASH.

Algorithm 3 SASH: function $A' = F(A, \mathcal{G})$ to create a neighbor solution

Input: network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the node set A;
Output: a neighbor solution set A' derived from A;

1: calculate $\sigma(w)$ for all node $w \in \mathbb{V}$;
2: $sum \leftarrow \sum_{w \in \mathbb{V}}(\sigma(w))$, $p(0) \leftarrow 0$, $flag \leftarrow true$;
3: **for** i from 1 to n **do**
4: $p(i) \leftarrow p(i-1) + \sigma(v_i)/sum$;
5: **while** flag **do**
6: create a random number $r \in [1, n]$, and a random probability $p \in [0, 1]$,
7: **for** i from 1 to n **do**
8: **if** $p(i-1) < p < p(i)$, $v_i \notin M(A, 2)$ and v_i is not the r^{th} element of A **then**
9: select v_i to replace r^{th} element of A to generate A', $flag \leftarrow false$;
10: **return** A';

Combination (MSA)

SAEDV can effectively cut down the time cost of computing the influence spread of a solution set. However, it just counts the influence spread in the neighborhood area of current solution set, so it may damage the accuracy of the algorithm. SASH can also speed up SA since it can effectively reduce the number of iterations by finding better solution quickly, although the improvement in terms of efficiency is not as obvious as SAEDV. Additionally, SASH can enhance the accuracy of SA. The two optimizations, SAEDV and SASH, optimize two different aspects of the SA algorithm; the two optimizations are orthogonal and can be combined.

The combined algorithm is called Mixed-SA(MSA). MSA replaces the fitness function $\sigma(A)$ with the fitness function given in Algorithm 2 (SAEDV), and replaces the creation function $F()$ with the function given in Algorithm 3 (SASH). As to be shown in Experiment, MSA is able to achieve more accurate solution than SAEDV while keeping its advantage of efficiency.

Experiments

The purposes of the experimental study are twofold:

- Evaluate the efficiency and effectiveness of the proposed Simulated Annealing(SA) algorithm by comparing with the state-of-the-art greedy algorithm—NewGreedy algorithm (Chen, Wang, and Yang 2009) and CGA (Wang et al. 2010) algorithm, as well as two heuristic algorithms—Degree-Discount (Chen, Wang, and Yang 2009) and Random(randomly select *k* nodes from the graph as the solution set) (Kempe, Kleinberg, and Tardos 2003).
- Evaluate the efficiency and effectiveness of the proposed optimized algorithms—SAEDV, SASH, and MSA.

Network Dataset

We use four real-life large-scale networks, namely Mobile social network, Who-trust-whom network of Epinions.com, Amazon Product Co-purchasing Network, and Web Network, which exhibit different features of real social networks. Some of them are used in recent influence maxi-

mization research (Wang et al. 2010; Chen, Wang, and Yang 2009) Table 2 gives the properties of the networks. The mobile network is extracted from the mobile call logs, where each mobile user corresponds to a node and the communication between two users corresponds to an edge. In Epinion network, nodes are members of Epinion and an edge between two nodes means one trust the other. In Amazon network, nodes are products and an edge between two products means the two products are often purchased together. In Web network, the nodes correspond to web pages and the edges correspond to links between pages.

Table 2: Statistics of four real-life networks

Dataset	Mobile	Epinions	Web	Amazon
Node	95.7K	75.9K	345K	262K
Edge	1.33M	508.8K	421.6K	1.23M
Average Degree	13.9	6.7	1.22	4.711

Experimental Setting

To compare the accuracy of different algorithms, we use the spread simulation method (Kempe, Kleinberg, and Tardos 2003) to compute the influence spread of the final solution set (top-k result) of each algorithm. Given a solution set, the spread simulation method does the spread simulations for R times and compute the average number of activated nodes as the estimation of $\sigma(A)$. Following the previous work, we set *R* as 10,000. The proposed algorithms, SA and SASH, also use the simulation method as the fitness function. For SAEDV and MSA, we count $\sigma(A)$ as the EDV of *A* (Algorithm 3). For SA and SA derived algorithms, we set $T_0 = 10,000,000$, $q = 1,000$, $\Delta T = 2,000$, and $T_f = 1,000,000$.

All experiments are performed on an Intel Xeon E5504 2G*2 (4 cores for every CPU), 36G memory. All the codes are written in C++.

Experimental Result

Accuracy when varying k. This experiment is to compare the accuracy of algorithms by varying *k* from 10 to 60 when the propagation probability is set at 0.05. Table 3 shows the influence spreading ability of the top-*k* nodes returned by the 8 different algorithms, including random algorithm (RDM), Degree-Discount (D-D), CGA, Newgreedy(N-G), SA, SAEDV, SASH, and MSA.

We observe 1) Random is much worse than the other methods in terms of accuracy on all data. This is consistent with the results reported in previous work. 2) NewGreedy (N-G) and CGA outperform Degree-Discount (D-D) and the performance disparity is more significant on data Web and Amazon. 3) SA consistently outperforms Newgreedy and CGA by about 33% on Amazon, and about $2\%-9\%$ on other data. 4) The three optimization algorithms, SAEDV, MSA, and SASH, have better influence spread than greedy algorithms. SASH is better than SA in term of influence spread. However, as expected, the adoption of EDV in SAEDV and MSA reduce the accuracy of SA by $1.64\% - 4.48\%$ and $3.23\% - 6.87\%$, respectively, on data Mobile.

To have a better understanding of SA and SASH, we study the number of iterations on the Amazon data set using SA and SASH. Table 4 shows the number of iterations that SA and SASH need to converge to their best solution when we vary k. SASH gets a better final solution set while its convergence needs fewer iterations than does SA. The result shows that SASH can obviously cut down the number of iterations. The results on the other three data sets are qualitatively similar.

Table 3: Influence spread of different algorithms as *k* is varied

Data	k	Algorithms							
		RDM	D-D	CGA	N-G	SAEDV	MSA	SA	SASH
Mobile	10	107	634	675	677	685	697	721	749
	20	186	866	886	899	921	926	962	982
	30	249	1023	1063	1081	1095	1112	1141	1184
	40	296	1174	1194	1205	1221	1245	1281	1335
	50	320	1279	1311	1323	1344	1362	1402	1453
	60	353	1400	1420	1453	1470	1491	1519	1587
Epinion	10	59	502	504	508	511	515	522	532
	20	91	511	577	580	593	597	605	621
	30	108	586	616	622	631	634	637	677
	40	122	621	654	662	673	678	682	720
	50	135	663	681	692	702	707	715	758
	60	147	692	714	732	745	753	769	796
Web	10	197	443	452	459	463	468	474	486
	20	340	732	754	761	766	769	773	789
	30	450	946	963	978	982	988	999	1023
	40	550	1049	1088	1098	1132	1143	1159	1180
	50	634	1167	1206	1245	1267	1288	1308	1355
	60	711	1286	1335	1378	1411	1432	1454	1490
Amazon	10	208	1288	1366	1646	1998	2032	2183	2434
	20	367	2261	2310	2890	3012	3189	3349	3655
	30	519	3125	3334	3723	4023	4078	4114	4645
	40	660	4121	4222	4576	4857	4923	5071	5496
	50	785	4572	4925	5231	5411	5499	5571	6099
	60	912	5308	5635	5822	6020	6152	6206	6814

Table 4: The number of required iterations as k is varied

k	10	20	30	40	50	60
SA	88K	376K	217K	498K	980K	777K
SASH	8.3K	20K	41K	69K	114K	168K

Accuracy when varying p. This experiment is to study the performance of the proposed SA algorithms when varying the propagation probability p. We set the size of solution set (k) as 30 and vary the propagation probability from 0.01 to 0.09 by following previous work. Table 5 shows the results on two representative networks: Web Network and Epinion network. The results on the other two data are qualitatively similar, and are ignored due to space limitation.

We observe: 1) SA and SA's derived algorithms always outperform D-D, N-G and CGA. For example, when the propagation probability is 0.05 on Epinion, the top-30 solution set of SA influences 636 nodes, while Newgreedy influences 603 nodes, CGA 593 and Degree-Discount 584. 2) SASH consistently improves the accuracy of SA. For instance, it outperforms SA by $1\% - 3.5\%$ on Epinion and Web data. As expected, the accuracy of SAEDV and MSA is lower than that of SA; however, they still perform better than greedy algorithms.

Table 5: Influence spread of different algorithms as p is varied

Data	p	Algorithms							
		RDM	D-D	CGA	N-G	SAEDV	MSA	SA	SASH
	0.01	89	218	321	324	348	348	351	351
	0.02	95	373	379	383	409	411	418	422
	0.03	99	441	448	453	486	489	492	494
	0.04	103	513	521	525	554	558	562	571
Epinion	0.05	107	584	593	603	631	633	636	656
	0.06	113	681	687	689	721	724	728	736
	0.07	117	763	769	793	807	810	814	831
	0.08	124	847	851	857	873	879	885	921
	0.09	130	969	976	980	1003	1008	1017	1028
	0.01	415	661	667	673	684	684	688	693
	0.02	423	746	750	760	776	782	789	824
	0.03	434	827	831	839	860	862	871	901
	0.04	446	895	902	911	934	937	944	967
Web	0.05	450	955	963	978	982	987	999	1023
	0.06	462	1027	1034	1045	1066	1072	1077	1092
	0.07	491	1115	1125	1134	1156	1163	1178	1184
	0.08	489	1190	1198	1204	1212	1221	1234	1254
	0.09	499	1278	1287	1294	1319	1326	1340	1352

Table 6: Runtime(seconds) of Different Algorithms to solve top-30 and top-50 problems on different networks

Data	Algorithms						
	D-D	N-G	CGA	SAEDV	MSA	SA	SASH
Mobile(k = 30)	13	16.7h	2.96h	8	8	270	80
Web	8	7.5h	5.1h	2	2	70	20
Epinion	15	7.7h	2.2h	2	2	630	28
Amazon	35	7.7h	0.7h	25	23	990	130
Mobile(k = 50)	17	23.3h	4.2h	14	14	421	142
Web	10	8.7h	6.7h	4	5	303	62
Epinion	18	9.4h	3.1h	4	4	970	71
Amazon	40	10.7h	1.1h	31	33	2903	231

Efficiency. We compare the runtime of D-D, NewGreedy, CGA, SA, SAEDV, MSA, and SASH when k = 30 and $k = 50$, the propagation probability $p = 0.05$ on four networks. Table 6 show the results.

We observe when $k = 30$ (the result at $k = 50$ is similar): 1) SASH is at least 70% faster than SA. The reason would be that SASH can locate the real influential nodes more efficiently, and the heuristic selection of new nodes in creating neighbor solution set helps to cut down the runtime. 2) MSA and SAEDV outperform all other algorithms in terms of runtime. MSA and SAEDV use only 8 seconds on Mobile, which is three order of magnitude faster than CGA, while their accuracy is better than CGA. 3) NewGreedy and CGA are slow, and need 16.7 hours and 2.96 hours, respectively, on Mobile. SA outperforms greedy algorithms by two orders of magnitude. 4) Degree-Discount is fast. However, its accuracy is worse than the 2 greedy algorithms and 4 SA algorithms. Also it is slower than SAEDV and MSA. Note that random needs less than 1 Second, but its accuracy is much worse than other methods, and is ignored here.

Summary. The proposed SA algorithm and its optimized versions, SAEDV, SASH, and MSA, outperform the state-of-the-art greedy algorithms up to 3 orders of magnitude in terms of runtime for influence maximization problem. They usually outperform the greedy algorithms by around 5% in terms of accuracy. Among the four proposed algorithms, MSA and SAEDV has the best performance in term of efficiency while MSA performs better than SAEDV in term of accuracy. SASH performs the best in terms of accuracy.

Conclusions

Unlike previous proposals on influential maximization that are mostly based on greedy algorithm, we take a totally different approach by using SA to mine top-k influential nodes. To our best knowledge, this is the first work to use SA to mine top-k influential nodes. Our experimental study shows that SA outperforms the state-of-the-art greedy algorithms, NewGreedy and CGA, in terms of both accuracy and efficiency, and the proposed optimizations are able to further improve SA.

This work opens up several promising directions for future work. First, it will be interesting to investigate the Parallel Computing advantage of SA for finding top-k influential nodes in very large social networks. Second, it will be interesting to investigate other optimizations on the SA algorithm to further improve performance.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (60703066, 60874082), and Beijing municipal natural science foundation (4102026).

References

Chen, W.; Wang, C.; and Wang, Y. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, 1029–1038.

Chen, W.; Wang, Y.; and Yang, S. 2009. Efficient influence maximization in social network. In *KDD*, 199–208.

Domingos, P., and Richardson, M. 2001. Mining the network value of customers. In *KDD*, 57–66.

Kempe, D.; Kleinberg, J.; and Tardos, E. 2003. Maximizing the spread of inffluence through a social network. In *ACM SIGKDD*, 137–146.

Kempe, D.; Kleinberg, J.; and Tardos, E. 2005. Influential nodes in a diffusion model for social networks. In *International colloquium on automata, languages and programming No32*, 1127–1138.

Leskovec, J.; Krause, A.; Guestrin, C.; Faloutsos, C.; VanBriesen, J.; and Glance, N. S. 2007. Cost-effective outbreak detection in networks. In *KDD*, 420–429.

Ma, H.; Yang, H.; Lyu, M. R.; and King, I. 2008. Mining social networks using heat diffusion processes for marketing candidates selection. In *CIKM*, 233–242.

Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; and Teller, E. 1953. Equation of state calculations by fast computing machines. In *Journal of Chemical Physics*, 1087–1092.

Mitra, D.; Romeo, F.; and Vincentelli, A. S. 1985. Convergence and finite-time behavior of simulated annealing. In *Proc. of 24th Conference on Decision and Control*, 761 – 767.

Wang, Y.; Cong, G.; Song, G.; and Xie, K. 2010. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*, 1039–1048.