Influence Spread Maximization in Social Network

Xinfei Shi, Hongzhi Wang, Jianzhong Li, and Hong Gao

***Abstract*—It's a challenging task to find a subset of node of size k in a social network such that targeting them initially as the seeds will maximize the influence spread. This problem is proved to be a NP-hard problem. We solve this problem in two aspects: 1) we improve the basic greedy algorithm, limiting the influence spread in a neighbor space to reduce the running time. We use the DAG and the recursion method to calculate the influence spread of each node. Also we transform this problem to a reachable probability query problem in an uncertain graph; 2) we present a more accurate degree discount heuristic algorithm which considers the relationship between the node and its neighbors. Intensive experiments on a large real-world social network show that: our improved greedy algorithm and degree discount heuristic algorithm are more efficient than the basic greedy algorithm and other heuristic methods.**

***Index Terms*—Classify-tree, DAG, degree heuristic, greedy, influence spread maximization, sampling.**

I. Introduction

There are many social networks such as Facebook, Twitter, paper reference network, Skype communication network, blog network and so on. They are very popular and successful. They become a platform for people to spread influence and expand relationship. Social network also brings some new problems. In this paper, we attempts to solve one of them, finding an initial seeds set of size k to maximize the influence spread, which is called *influence spread maximization* problem.

We use a scenario to illustrate its application. A company developed a new product, and wants to expand this product to society through social network. With limited budget, this product could only be sent as some free sample for limited number of people. This company wishes that these people like their product, and spread the benefit of the product to the public through their friends. In this way, the company will achieve its aim. The problem arises how to choose the initial people to maximize the number of people who will receive the information of product in the social network. This problem is called influence spread maximization. This problem brings technical challenge. The major challenge is that the size of social network is often very large with complex structure. The solution of this problem should be efficient and scalable.

A. Related work

Domingos and Richardson [1] first present a probabilistic solution. However, Kempe etc. [2] are the first to formulate this problem to a discrete optimization problem; they proved this problem is NP-hard. Leskovec etc. [3] present an optimal greedy algorithm which is called cost-effective lazy forward (CELF) framework using the sub-modular properties of influence spread to reduce the calculation for the spread estimate of nodes. When a node update its influence spread, its value will not bigger than the its current value. So we can use this idea to update the influence spread of nodes lazily. In [4], Chen etc. present NewGreedyIC algorithm which is based on the greedy algorithm. When it selects a new seed, it just random the graph one time, and it uses the equivalent model as the IC model. And they present a heuristic algorithm which is called DegreeDiscount. Its main idea is that we choose the max-degree node as the seed every time. And we reduce the degree of its neighbor nodes. This heuristic algorithm is very fast and efficient, but it requires very small propagation probability. There is another direction. In [5], this paper presents a heuristic algorithm to limit the size of node space be affected. With the varying of the limit size of the local space, this algorithm can get a trade-off between the efficiency and precision.

B. Our Contribution

In summary, current methods have two limitations: 1) they require repeated compute of the spread function on the entire graph; 2) the scan times are the quadratic to the number of nodes. In this paper, we attempt to overcome the shortcomings. At the same time, we get a trade-off between efficiency and the precision. Thus we can plug our method into the basic greedy algorithm.

We solve the problem in two directions.

In one direction, we design new schemes to further improve the basic greedy algorithm under linear threshold model. Our paper generates the DAG of induced graph $G_1(v)$ for every node v in the network to represent the vertices that information from v can reach. We set the weight of node and edge appropriately. To compute the influence spread, we present a recursion back patch method for every node in $DAG(v)$ under linear threshold model. However, the dag loses many information of the original graph. To address this problem, we use the graph theory technique in $G_1(v)$. We transform this problem to a reachable probability query problem in uncertain graph. We introduce the possible graph, define the classify tree and carry out the sampling in this tree. We get an approximate $\delta(v)$ under linear threshold model.

For independent cascade model, we propose a new degree discount heuristic method, DDH. When selecting a max degree node as a seed, we reduce the degree of its one hop neighbors which results in the reduction of the influence spread of its neighbors. This heuristic method could get more accurate result than existing methods.

We conduct extensive experiments on real-life social

Manuscript received May 8, 2013, revised July 19, 2013.

The authors are with the Massive Data Computing Research Lab, Computer Science and Technology, Harbin Institute of Technology (e-mail: xinfeishi@gmail.com, wangzh@hit.edu.cn, lijzh@hit.edu.cn).

network. Experimental results show that our method outperforms existing methods and has good scalability.

C. Paper Organization

Section II proposes the greedy algorithm and our improved method under the linear threshold model. Section III introduces the degree discount heuristic DDH under independent cascade model. Section IV shows our experimental result and analysis. Section V concludes our paper. For the convenience of discussion, we summary the symbols used in this paper in Table I.

TABLE I: SYMBOL TABLE

Symbols	Descriptions
n	Number of vertexes in G
m	Number of edges in G
η	Threshold of shortest path
$G_1(v)$	A induces graph for v
$DAG(v)$	A DAG graph of $G_1(v)$
$G_2(v)$	A possible graph of $G_1(v)$
T	#number of total samples
d_v	Degree of vertex v
dd_v	Discount degree of v

II. GREEDY INFLUENCE SPREAD MAXIMIZATION ALGORITHM

In this section, we introduce the definition of Influence Spread Maximization and the basic greedy algorithm which is the framework of our work. Then we introduce two flow models to spread influence (Section A). Then we present two methods to improve the greedy algorithm (Section B).

A. Problem Definition and the Greedy Algorithm

A social network is defined as a graph $G = (V, E)$, where V is the set of individuals in social network and E is the set of relationships between the individuals. For example, in a paper reference network, each node represents a paper. An edge (u, v) represents paper u cites v. It is a problem how the influence spreads through the network and what the influence cascade model is. For these two problems, we introduce the independent cascade model(IC) [2] and the linear threshold model (LT) [2] below to describe the influence spreading.

In a graph, each node has two states, active or inactive. One node can be transmitted to active state from inactive state, but cannot be transmitted in reversed direction. At time $t = 0$, a set A of size k will be chosen as an initial seeds set. IC or LT model describe the influence spread pattern.

IC model [2]

For every node v and its neighbor node w in V, there is an edge $e(v, w)$, and $p(v, w)$ is the weight of this edge. It means the probability of information spread from v to w. If at time t, v is active, and w is inactive, then v tries to activate w by probability $p(v, w)$. If this process succeeds, w is active at time $t + 1$. If all w's seed neighbors fail, then w is still inactive.

LT model [2]

For a node v, it has a threshold θ_v. v has many neighbors. For every neighbor w, there is a probability for the edge $e(v, w)$. And these probabilities satisfy a condition $\sum_{w \in N(v)} p(v, w) \leq 1$, $N(v)$ is the neighbor node set of v. If at time t, v is inactive and some neighbors of v are active. If these neighbors satisfy $\sum_{w \epsilon N(v) \text{ and } w \text{ is active}} p(w, v) \geq \theta_v$, v will be active in time $t + 1$.

For example, Fig. 1 is an example of social network G. u activates w with probability 0.3, and v is 0.4. Suppose at time t = 0, u and v are active. In IC model, at time t = 1, u and v activate w indecently. If both of u and v fail, w is inactive. So w is active with the probability $1 - (1 - 0.3) \times (1 - 0.4) = 0.58$. In LT model, it is supposed that w has a threshold $\theta_w = 0.5$. Since $\sum_{x \epsilon N(w)} p(x, w) = 0.3 + 0.4 \geq \theta_w = 0.5$, w is active at next following time.

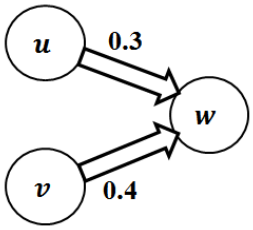

Fig. 1. An example of social network G.

Problem definition: Influence Spread Maximization

Supposing A is the initial seed set, we define $\delta(A)$ as the expected number of node which is active in the final state using the IC or LT model. $\delta(A)$ is the influence spread of seeds set A. Given a graph $G = (V, E)$ and a number k, we should find an initial seeds set A of size k which satisfies the condition that $\delta(A) \geq \delta(B)$, where B is any other initial seeds set of size k. This problem has been proved to be NP-hard [2]. To solve this problem, a greedy climb algorithm has been proposed in [2], as in Algorithm 1.

Algorithm 1: GA(G, k)
Input: G is the graph;
k is the number of seeds.
Output: A is target seeds set of size k.
1 Set initial seeds set $A = \emptyset$;
2 **while** ($|A| < k$)
3 {
4 For every node v in set $V \backslash A$
5 Calculate $\Delta_v = \delta(A \cup v) - \delta(A)$
6 Choose the node $v = argmax_v \Delta_v$ as the next seed: $A = A \cup v$
7 }
8 **return** A

We use this algorithm as the framework. The following two optimizers can be plug into Line 5.

To overcome the shortcomings of the greedy algorithm, we attempt to speed up the estimation of $\delta(A)$ calculation in Line 5 [6]. Consider the case that the probability of spread influence from a node to its neighbor through many hops (for example, one hundred hops) is very small. In practice, we can ignore very small probability event. To get a trade-off between the efficiency and precision, we use a threshold η to limit the size of this region. We can ignore the paths whose probability is smaller than η.

We propose our algorithms with basic greedy algorithm as the framework. In the section B, we limit the influence in a local neighbor region. Then we present two methods to calculate the influence spread. These two methods are both based on the graph theory. The former one generates a directed acyclic graph and utilizes the top-down recursion

process, which will be proposed in section 1) of B. The latter one uses the query reachable probability technology under uncertain graph, which will be introduced in section 2) of B.

B. Speed up the Basic Greedy Algorithm under the LT Model

To overcome the shortcomings of the greedy algorithm, we suppose the influence spreads in neighbor region, ignoring the paths whose probability is smaller than η.

For a special node, maybe there are many paths which can reach its neighbor. We just use the shortest path (which means the max spread probability). If the probability of that path is greater than the threshold, we can keep this node. Otherwise we will ignore it. Let $p(v, u) = (p_1 = v, p_2, p_3, \dots \dots p_n = u)$, v is a special seed node, u is any other node in V, and p is a simple no-cycle path from v to u. Let $P(p)$ be the probability of the path p. $P(p) = \prod w(p_i, p_{i+1})$. $w(p_i, p_{i+1})$ is the spread probability of the edge $e(p_i, p_{i+1})$.

If we set the weight of every edge $e(s, t)$ as $-ln\, w\,(s, t)$, we can see that the max probability path is the shortest path from v to u. We define the shortest path as:

$$sp(v,u) = argmax_p\{P(p)|\ p \epsilon\ p(v,u)\} \qquad (1)$$

We keep the node u if $P\big(sp(v,u)\big) \geq \eta$. These nodes form a node set $V_1(v)$. And $G_1(v)$ is the induced graph of G with node set $V_1(v)$. We will continue our calculation in $G_1(v)$.

In Line 5 of the greedy algorithm, the most costly step is the computation of $\delta(A)$. Hence we focus on how to calculate $\delta(A)$. From [7], we have $\delta(A) = \sum_{v\epsilon A} \delta^{V-A+v}(v)$. The spread of a set A is the sum of the spread of each node $v\epsilon A$ on sub-graph induced by $V - A + v$. Hence we can calculate the spread influence of each seed node, and sum the result to get the final $\delta(A)$.

In the following, we use two methods to calculate $\delta(v)$. And these two optimizers can be plugged into the basic greedy algorithm in Line 5. The first one generates a DAG for $G_1(v)$ to simplify the calculation. However, it loses some information of original graph $G_1(v)$, which possibly causes some inaccuracy error. Hence we present the second method to calculate the $\delta(v)$. This method transforms the original problem to a reachable probability query problem. We will discuss these two algorithms in section 1) and section 2), respectively.

1) Using the DAG and recursion to calculate $\delta(v)$

With many cycles in $G_1(v)$, the evaluation of $\delta(v)$ is very difficult. As a comparison, in a no-cycle graph, we can adopt the top-down recursion method to compute $\delta(v)$ efficiently. Hence we attempt to generate the spanning dag of $G_1(v)$ in section a) and then use the recursion method in section b) to construct optimizer for basic greedy algorithm.

- Generate DAG and set weight

The task of this step is to generate the spanning dag of $G_1(v)$. The pseudo algorithm is shown in Algorithm 2. In this algorithm, we first compute the strongly connected components (SCC) of graph $G_1(v)$ (Line 1). We collapse each strongly connected component into one vertex with the size of SCC. And we set the edge and the weight of edge appropriately in line 3. Finally we get a directed acyclic graph $DAG(v)$.

Algorithm 2: DAG($G_1(v), v$)
Input: v is a seed node in graph G.
$G_1(v)$ is the induced graph of G with node set $V_1(v)$
Output: $DAG(v)$ is the dag of $G_1(v)$
1 Calculate SCC for $G_1(v)$, collapse them.
2 $w(S) = |S|$ for every SCC S in $DAG(v)$.
3 $w(S_i, S_j) = \sum w(x,y)\quad x \in S_i, y \in S_j, e(x,y) \in E$.
4 **return** $DAG(v)$

We set the weight of S to $|S|$, which means the number of nodes S contains. If S is active, we know there are $|S|$ nodes are active. S_i and S_j are two SCCs. If S_i contains a node x, S_j contains a node y, and x and y has link in graph $G_1(v)$, we sum the weight of $e(x, y)$ to the get weight of $e(S_i, S_j)$ in $DAG(v)$.

- $\delta(v)$ Computation

In this step, we attempt to compute $\delta(v)$, using the dag graph $DAG(v)$ of $G_1(\mathrm{v})$ which is generated in the last step. Note that u may have more than one precursor and successor. Array $p[u]$ contains all the precursors of u. We use the top-down recursion method to solve this problem.

The pseudo code of this algorithm is shown in Algorithm 3. It calculates the $\delta(v)$. We use a table array T to record the $\delta(u)$ for every node u.

Algorithm 3: IS(node u)
Input: u is a node in G_1(v).
Output: $\delta(u)$ is the influence spread of u
1 Initial the element of array T to invalid.
2 **if** ($T[u]$ is valid) return $T[u]$;
3 **if** ($u == v$) return 1;
4 **else return** $T[u] = \sum_{x \epsilon p[u]} IS(x) \times w(x,u)$

In this algorithm, Line 1 initializes every element of array T to invalid. If $T[u]$ is valid and so $\delta(u)$ is calculated and saved in $T[u]$, we just return it to avoid the repeated computation in Line 2. Otherwise, we calculate the influence spread of all u's precursors and sum the spread probability from them to v. During this process, we record the value in $T[u]$ (Line 4).

2) Query reachable probability in uncertain graph under LT

The above algorithm DAGIS is based on DAG. It loses some information of original graph $G_1(v)$. We present another method based on the graph theory and sampling technology to solve this problem in original graph $G_1(v)$. This algorithm converts the graph into an uncertain graph and the problem of querying reachable probability. In the uncertain graph, the seed node v is the source node and every other node in graph $G_1(\mathrm{v})$ is considered as a sink node. In this graph, the influence is flowing from the source node to sink nodes in uncertain graph $G_1(v)$. Thus the problem becomes query reachable probability from single source to multi-sinks in the uncertain graph. To avoid too long spread path, we set a hop distance limit d. We will discuss the solution of this problem in this section.

We will introduce the definition of uncertain graph and the possible graph in section a); we define d-reachable paths from source to sinks to construct all the possible graphs in

formula (3). In section b) we define the classify-tree according to formula (3) to reduce the cost of searching; finally we use the sampling technology to get the approximate result of $\delta(v)$ to accelerate the processing.

- The uncertain graph and possible graph

In $G_1(v)$, the event of influence spreading from one node to another node is uncertain. Hence $G_1(v)$ can be modeled as an uncertain graph. In an uncertain graph $G = (V, E, P)$, P is the probability function of each edge, mapping $E \to (0, 1]$.

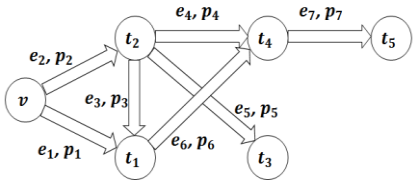

Fig. 2. Uncertain graph $G_1(v)$.

In an uncertain graph, with the source node *v*, and the sink node set *Y*, the problem is to calculate the reachable probability from *v* to any nodes in *Y*. We denote the reachable probability as $R(v, Y|d)$. d is the distance limit of path length. Then we have

$$R(v, Y|d) = \sum R(v, t|d) \qquad t \in Y \setminus v \qquad (2)$$

Then we focus on the computation of $R(v, t|d)$.

As the basic model, we define possible graph and use it to define the *d*-reachable probability.

If a graph $G_2 = (V, E')$ satisfies $E' \subseteq E$, we say G_2 is a possible graph of uncertain graph G. The probability of G_2 is $pr(G_2) = \prod_{e \in E'} p(e) \prod_{e \in E \setminus E'} (1 - p(e))$. We set the distance of two nodes *v* and *t* to the min-length of all the simple path links these two nodes noted as $Dis(v, t)$. If we have a distance threshold d satisfying $Dis(v, t) \leq d$ in G_2, then G_2 is *d*-reachable. The *d*-reachable probability from source v to sink t is the sum of probability of all the possible graphs which are *d*-reachable. That is $R(v, t \mid d) = \sum_{G_2 \text{ is } d-reachable} pr(G_2)$.

Suppose $p_1 \ldots\ldots p_r$ are all the *d*-reachable paths from *v* to t in G. p_i and p_j may share vertices. If path p_1 exists in G_2, then *v* and *t* are *d*-reachable no matter whether $p_{i,i\neq 1}$ exists. The probability of all this kind of possible graphs is the probability of p_1 (all the other paths and edges are not related). Hence p_1 generates a kind of possible graphs. When p_1 do not exist but p_2 exists, *v* and *t* are *d*-reachable no matter whether other paths exist. And the probability of all this kind of possible graphs is the probability of p_1 does not exist and p_2 exist. Thus we can get the union of all the possible graphs:

$$p_1 \cup \overline{p_1} p_2 \cup \ldots\ldots \cup \overline{p_1 p_2 \ldots\ldots p_{r-1}} p_r \qquad (3)$$

p_1 represents all the possible graphs which contain path p_1. $\overline{p_1} p_2$ represents all the possible graphs which contain path p_1 and don't contain p_2 and so on. This formula generates the entire possible graphs. And it is the basic theory of classify-tree which is defined in section b). We will show how to construct the entire *d*-reachable paths and hence to construct the entire possible graph.

To compute *d*-reachable probability, a straightforward method is to enumerate all possible graphs generally. There are $2^{|E|}$ possible graphs. Clearly, this cost is large. To accelerate the process, we design sampling technology to compute the approximate result of *d*-reachable probability. For effective sampling, we construct classify tree according to formula (3) and sampling through it in section b).

- Sampling in classify tree

Generally, we need to random generate the entire uncertain graph and use the number of *d*-reachable graphs to divide the total amount of graphs to get the reachable probability. This naive random method has weakness because it does not consider the relationship between edges in the process of composing path. Hence we randomize branches through the classify-tree which is defined as below.

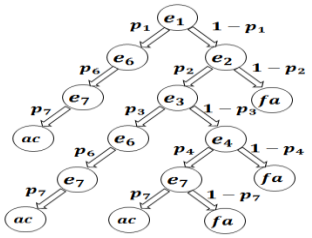

Fig. 3. Classify-tree for graph G (ac: accepted; fa: false; some false branches are omitted).

Definition 1 A classify-tree is a tree with the structure of each node as follows [8], [9].

$$node = \text{Node}(vertex, edge, currentCost, leftChild, rightChild, state).$$

Where $vertex$ represents the current considered node; $edge$ represents the current considered edge. $currentcost$ means the cost of current search branch. $leftChild$ represents the current considered edge $node.edge$ is selected into the simple search path. $rightchild$ represents $node.edge$ is not selected. Member $state$ including two states: accepted and false. The $state$ is false if we find that *v* to a sink t is not *d*-reachable through the edges we select. Otherwise the $state$ is accepted which represents a successful sample is got (the sampling process will be introduced below). Every internal node's state is accepted.

This classify-tree represents all the *d*-reachable paths. We consider how to expand branch on such classify-tree. With the current considered node denoted by $currentNode$, all the edges linked to $currentNode.vertex$ in the graph G are candidates for the next step. When we consider an edge e in this candidate set, if e satisfies a condition $currentNode.currentCost + 1 \leq d$, e is a valid branch. Otherwise e is an illegal branch, and we abandon this branch. The search process ends when we meet the leaf Node.

During the search, we are not necessary to generate the classify-tree explicitly, but construct it during searching. When a no-leaf node is visited, current considered edge is randomized by its probability. If this edge exists, we traverse the $leftChild$ branch. Otherwise we go to $rightChild$ branch which means this edge does not exist.

We use DFS to expand branch. If we come across a node, we get the node' state which means a sample is obtained. Then we use the amount of accepted samples and total samples to estimate the *d*-reachable probability and also get $\delta(v)$. We have *T* samples in total. With the amount of accepted samples of *t* denoted by *x*, the probability of *d*-reachable is approximately calculated as:

$$\mathcal{R}(v,t|d) = x/T \quad (4)$$

Thus, according to formula (2), influence spread $\delta(v)$ in original graph $G_1(v)$ can be calculated as:

$$\delta(v) = \mathcal{R}(v,Y|d) = \sum \mathcal{R}(v,t|d) \quad t \in V_1(v) \setminus v \quad (5)$$

Algorithm 4: Sampling(Node *v*)
Input: *v* is root node
Output: influence spread of *v*: $\delta(v)$
1 **int**[] acceptedSample = {0};
2 **int**[] totalSample = {0};
3 Generate root from *v*; path = ∅;
4 **while** (!(totalSample[*u*] >= *T* for all node *u*))
5 {
6 **DFSSearching**(root, path);
7 }
8 double *result* = 0;
9 **for** each *t* in $V_1(v) \setminus v$ *result* += acceptedSample[*t*]/*T*;
10 **return** *result*;

Algorithm 5: DFSSearching($node, path$)
Input: *node*: current consider node
path: current searching branch path
1 **initialize**:
2 *x* = *node.vertex*;
3 *ex* = *node.edge*;
4 *cost* = *node.currentCost*;
5 *w* = *ex.endVertex*; /* means *ex* = *e(x, w)*, *w* is the end vertex of edge *e* */
6 **if** (*totalSample*[*x*] < *T*)
7 {
8 *totalSample*[*x*]++;
9 **if**(*node.state* == accepted) *acceptedSample*[*x*]++;
10}
11 Random generate a number ***R*** in interval [0, 1];
12 **if**(*cost* + 1 <= *d* and ***R*** <= *p(x, w)*)
13 /* *e* is valid and *e* is random selected */
14 {
15 New *leftNode* = node(*w*, *w.firstCandidate*, *cost* + 1, null, null, accepted);
16 /* there are many candidates edge from *w*, we select the first candidate, and check the other edges in the after time. */
17 *path* += *ex;*
18 **DFSSearching**(*leftNode*, *path*);
19 *path* -= *ex*;
20 }
21 **else** //e is not valid or e is not random selected
22 {
23 New *rightNode* = node (*v*, *nextCandidateEdge*, *cost*, null, null, null);
24 **DFSSearching**(*rightNode*, *path*);
25 }

Consider the example in Fig. 3. We construct the classify-tree for graph *G* of Fig. 2. At the root node *v*, we have two choices, e_1 and e_2. When we choose e_1 with probability p_1, we go to node t_1. At this time, we can choose e_6 with probability p_6. Then we can select e_7 with probability p_7 to reach accepted (ac for short) state which means an accepted sample. When we do not choose e_1 with probability $(1-p_1)$, we can choose e_2 with probability p_2 to continue our search process instead. If we do not choose e_2, we will not reach any node. Then we reach a false (fa for short) state. The construction of other branches is similar.

The pseudo algorithm for Sampling is shown in Algorithm 4, where the DFS starting at root node is shown in Algorithm 5.

In Algorithm 4 Sampling, if we have not obtained all the samples, we call the function DFSSearching to continue our searching process in classify-tree (Lines 4-7). Line 9 calculates $\delta(v)$, and finally returns it (Line 10).

In Algorithm 5 DFSSearching, we check whether the current considered vertex *x* has visited all its samples to decide whether we should utilize current sample (Lines 6-9).Then in Lines 11-25, we randomize the branch using DFS to construct the classify-tree implicitly.

III. Degree Discount Heuristic under the IC Model

In this section, we introduce the degree heuristic to speed up the calculation of influence spread. First, we introduce max degree method and its weakness. Then we drive a more accurate degree heuristic method called DDH to overcome the weakness. DDH can also be treated as an optimizer and plugged into the basic greedy algorithm in Line 5 of Algorithm 1.

It costs too much time even if we use the improved greedy and graph algorithm when the size of graph is very large. One possible improvement is to use heuristic method. Degree heuristic strategy is usually used to estimate the influence spread of a node in social network. Such heuristic strategy [10] is effective in some cases, but the result of influence spread is still not as large as the basic greedy algorithm. They select a max-degree node each time. But they ignore some problems. A pair neighbor node can also have large degree, but their neighbor set may have many overlapping. Therefore, their influence is not as big as the optimal result. This motivates us to update the degree of *v*'s neighbors to update their influence spread when we select *v* as a seed.

With this consideration, we drive a more accurate degree discount heuristic under IC model which is called DDH. In this algorithm, we just consider one hop neighbors, and the relationship between these neighbors to update their degree. This makes our heuristic more accurate. And this idea forms the guideline of our work.

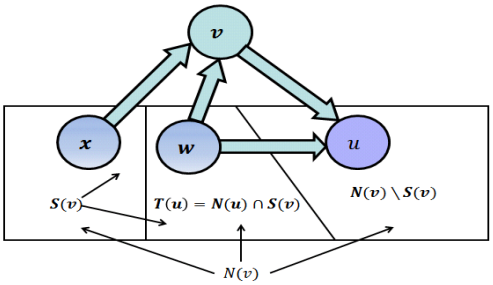

Fig. 4. The relationship between *v* and its one hop neighbors.

To illustrate the algorithm, we use a Fig. 4 to show the relationship between v and its one hop neighbors. For a node v, we define $N(v) = \{s \mid e(s, v) \in E\}$, in which the

set of one hop neighbors of v is. t_v neighbors of v has been selected as seeds, and they make up a set $S(v) = \{x \mid x \in N(v), x\ is\ seed\}$.

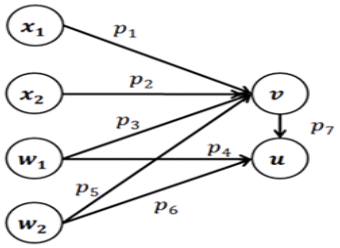

Fig. 5. An example of DDH.

So what's the influence spread of v when set $S(v)$ has been selected as seed? We consider how to calculate $\delta(v)$. v has two states: (1) v is activated by its seed neighbors with probability $I(v) = 1 - \prod(1 - p(x, v))\ \ x \in S(v)$. In this case, selecting v as the seed makes no contribution to $\delta(v)$. (2) If v is not influenced by its neighbors with the probability $1 - I(v)$, selecting v as a seed will activate other nodes, and this activation is regard as $\delta(v)$. This activation contains two aspects: (I) v is active with probability 1; (II) We consider a node u in $N(v)$ and u is not a seed node, which means $u \in N(v)\backslash S(v)$. u may be influenced by some seeds in $S(v)$. In this case, u cannot make contribution to $\delta(v)$. Let $T(u) = N(u) \cap S(v)$. u is not activated by $T(u)$ with the probability $B(u) = \prod(1 - p(w, u))\ \ w \in T(u)$. At the same time, u is activated by v with the probability $p(v, u)$. Then we can calculate the additional influence spread of v:

$$\delta(v) = \big(1 - I(v)\big)\Big(1 + \sum_{u \in N(v)\backslash S(v)} B(u) \times p(v, u)\Big) \quad (6)$$

When we select a node as a seed, we must update the degree of all its neighbors.

For example, in Fig 5, v has five neighbors including four seeds: x_1, x_2, w_1, w_2. v is not activated by its seed neighbors with the probability $1 - I(v) = (1 - p_1)(1 - p_2)(1 - p_3)(1 - p_5)$. In this case, selecting v as seed will make contribution to the influence spread. This contains two aspects: (1) v is active by probability 1. (2) u is not activated by w_1, w_2 and activated by v with the probability $B(u) \times p(v, u) = (1 - p_4)(1 - p_6)p_7$. Thus the influence spread of v is

$$\delta(v) = (1 - I(v))(1 + \sum B(u) \times p(v, u)) = (1 - p_1)(1 - p_2)(1 - p_3)(1 - p_5)(1 + (1 - p_4)(1 - p_6)p_7)$$

The pseudo code of DDH is shown in Algorithm 6.

Initially, v's discount degree which also means $\delta(v)$ is assigned to d_v (Lines 2-4). Then in each iteration, we select a node with the max influence spread (Line 7), add it to seeds set A (Line 8) and update the discount degree of all its neighbors as discussed above (Lines 9-15). Suppose the average number of node's neighbor is q. When we select a new seed y, we should update all y's neighbor v's degree. For the update of v, it should scan v's neighbor u and u's neighbor w. So we cost $O(q^3)$ time. With the Fibonacci heap, the running time of DDH is $O(kq^3 logn)$.

Algorithm 6: DDH(G, k)
Input: G is the graph; k is the number of seeds.
Output: A is target seeds set of size k.
1 Initialize $A = \emptyset$;
2 **for** each vertex v **do**
3 compute its degree d_v
4 $dd_v = d_v$; /* dd_v means v's discount degree and also its influence spread.*/
5 **end for**
6 **while** $(|A| < k)$
7 select $y = argmax_y\{dd_y \mid y \in V\backslash A\}$
8 $A = A \cup y$
9 **for** each neighbor v of y **do**
10 $I(v) = 1 - \prod(1 - p(x, v))\ \ x \in S(v)$
11 for each u ∈ N(v)\S(v) do
12 B(u) = ∏(1 − p(w, u)) w ∈ T(u) = N(u) ∩ S(v)
13 end for
14 $dd_v = (1 - I(v))(1 + \sum B(u) \times p(v, u))$ u ∈ N(v)\S(v)
15 end for
16 end while
17 return A

IV. Experiments

A. Experiments setup

All algorithms are written in C++ language. The running environment is a PC with Intel i5 3.1 GHZ CPU, 4G main memory and 500G disk. We use real social network to test our algorithms. Dataset Amazon, the Amazon product co-purchasing network [11], which is dated on March 2, 2003, where nodes are products and a directed edge from s to t means product s is often purchased with product t. It has 262k nodes and 1.2M edges.

We compare algorithms with other algorithms in two measures: influence spread and running time. For the LT model, we compare our two algorithms (DAGIS and Sampling) with the CELFGreedy [4] algorithm. For the IC model, we compare our algorithm DDH with CELFGreedy [4] and Degree [2] and Distance [2].

B. Experimental Results

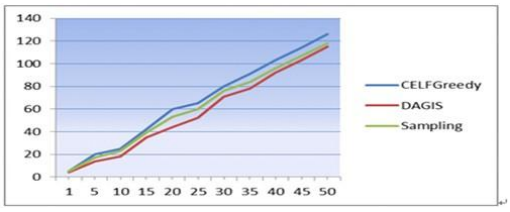

Fig. 6. Influence spread of these algorithms under LT model (p=0.01).

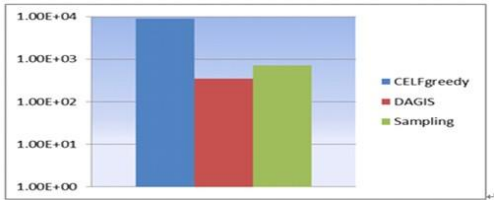

Fig. 7. Running time of different algorithms under LT model (k=50, use log function).

We provide the experimental results of different algorithms for two influence cascade models. In the experiments, without explicit explanation, we set k = 50, and the spread probability p is same for every edge and p = 0.01 and η=10^{-20}.

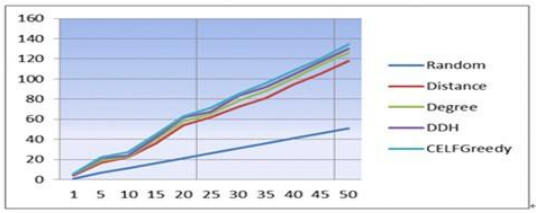

Fig. 8. Influence spread of these algorithms under IC model (p=0.01).

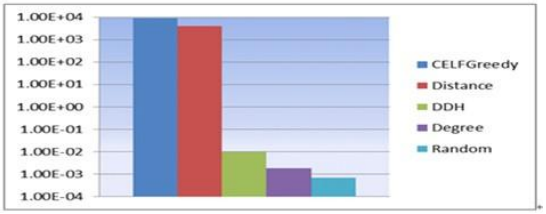

Fig. 9. Running time of different algorithms under IC model (k=50, use log function).

1) LT model

For LT model**,** we compare the methods in two aspects. For effectiveness, we vary k from 1 to 50. Different algorithms are run to select k nodes and calculate the influence spread of these k nodes. The experimental results are shown in Fig. 6. For efficiency, k = 50, and we run these three algorithms to compare their running time and the experimental results are shown in Fig. 7. From the result, it can be observed that even though the influence spread of DAGIS and Sampling is a little lower than the CELFGreedy algorithm (3.2% and 4.7% lower than the CELFGreedy), they are much (25 and 12 times) faster than the greedy algorithm because we limit the calculation in a smaller region.

2) IC model

For IC model**,** we compare the methods in two aspects. To test effectiveness, we vary k from 1 to 50. Different algorithms are run to select k nodes and calculate the influence spread of these k nodes. The experimental results are shown in Fig. 8. For efficiency, we set k = 50, and run these five algorithms to compare their running time and the experimental results are shown in Fig. 9. The random algorithm is the baseline, and its efficiency is very bad. From the results, it is observed that the Degree and Distance algorithm are much better than Random. But they are a little worse than our DDH. Degree is 4.6% lower than the CELFGreedy algorithm, and distance is 8.7%, and our DDH is 3.5%. We can see that our heuristic save much time, and it is 0.9×10^6 times faster than CELFGreedy. And its influence spread is almost same as the CELFGreedy.

V. Concluding and Future Work

Influence Spread maximization is a very important, useful and changeling task in social network. The size of social network is often very large with complex structure. Therefore, existing methods are not suitable for this problem and efficient algorithm is in demand. In this paper, to achieve high efficiency, we propose two improved greedy algorithms and a heuristic algorithm. The extensive experiments on the massive-size social network show that we can get the accurate guarantee of influence spread. And all our algorithms reduce the running time significantly. As future work, we can deeply study the heuristic method to adopt it to the linear threshold model. We can also consider not only one-hop neighbors but also nodes in multiple hops adaptively to get more accurate result.

References

[1] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *Proc. the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 61–70, 2002.
[2] D. Kempel, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," *ACM SIGKDD*, pp. 137-146, 2003.
[3] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," *KDD*, pp. 199-208, 2009.
[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *Proc. SIGKDD*, pp. 420-429, 2007.
[5] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," *ICDM 2010*.
[6] R.-H. Li, J. X. Yu, and Z.-C. Shang, *Estimating Node Influenceability in Social Networks*.
[7] A. Goyal, L. Wei, and L. V. S. Lakshmanan, "SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. in Data Mining (ICDM)," in R. P. Satorras and A. Vespigni, *Epidemics and immunization in scale-free networks, Graphs and Networks from the genome to the Internet 2003*.
[8] W. Zhang and Q. Y. Zhai, "Distance threshold based reacha- bility queries in uncertain graphs," *Journal of Chinese Computer Systems*, vol.33, no. 10, pp. 2164-2169, 2012.
[9] R. Jin *et al*., "Distance-constraint reachability computation in uncertain graphs," in *Proc. the VLDB Endowment*, 2011, vol. 4, no. 9, pp. 551-562.
[10] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*, Cambridge University Press, 1994.
[11] Amazon product co-purchasing network. (March 02 2003). [Online]. Available: http://snap.stanford.edu/data/amazon0302.html.

Xinfei Shi was born in Anhui province in 1990. From 2007 to 2011, he received his bachelor in computer science and technology in Harbin Institute of Technology, Harbin, Heilongjiang, China. From 2011 to 2013, he was a master candidate in computer science and technology of Harbin Institute of Technology. He is currently study in Harbin Institute of Technology. Current research interests: Massive Data Computing Research.

Hongzhi Wang was born in Liaoning province in 1978. In 2008, he got his Ph. D in computer science and technology in Harbin Institute of Technology, Harbin, Heilongjiang, China.

He is associate professor in Harbin Institute of Technology. His interests include data quality, xml data management.

Jianzhong Li was born in Heilongjiang province in 1950. From March 1972 to August 1975, he is with Heilongjiang University department of mathematics. From September 1975 to June 1978, he is with Tsinghua University department of electronic engineering. Professor Li is a Ph. D. and supervisor in Harbin Institute of Technology. His research interests include wireless sensor network, cyber-physical systems, database, massive data processing etc.

Hong Gao was born in Heilongjiang province in 1966. He is a Ph. D in computer science and technology of Harbin Institute of Technology.

She is a professor and supervisor in Harbin Institute of Technology. Her research interests include wireless sensor network, cyber-physical systems, massive data management and data mining.