Scalable Influence Maximization in Social Networks under the Linear Threshold Model

Microsoft Research Technical Report MSR-TR-2010-133 October 2010

Wei Chen
Microsoft Research Asia
Beijing, China
Email: weic@microsoft.com

Yifei Yuan
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania, U.S.A.
Email: yifeiy@cis.upenn.edu

Li Zhang
Microsoft Research Silicon Valley
Mountain View, California, U.S.A.
Email: lzha@microsoft.com

***Abstract*—Influence maximization is the problem of finding a small set of most influential nodes in a social network so that their aggregated influence in the network is maximized. In this paper, we study influence maximization in the linear threshold model, one of the important models formalizing the behavior of influence propagation in social networks. We first show that computing exact influence in general networks in the linear threshold model is #P-hard, which closes an open problem left in the seminal work on influence maximization by Kempe, Kleinberg, and Tardos, 2003. As a contrast, we show that computing influence in directed acyclic graphs (DAGs) can be done in time linear to the size of the graphs. Based on the fast computation in DAGs, we propose the first scalable influence maximization algorithm tailored for the linear threshold model. We conduct extensive simulations to show that our algorithm is scalable to networks with millions of nodes and edges, is orders of magnitude faster than the greedy approximation algorithm proposed by Kempe et al. and its optimized versions, and performs consistently among the best algorithms while other heuristic algorithms not design specifically for the linear threshold model have unstable performances on different real-world networks.**

***Keywords*-influence maximization; social networks; linear threshold model;**

I. Introduction

Influence maximization is the problem of finding a small set of most influential nodes in a social network so that their aggregated influence in the network is maximized. The seminal work by Kempe, Kleinberg and Tardos [1] provides the first systematic study of influence maximization as a discrete optimization problem. Influence maximization has the obvious application in viral marketing through social networks, where companies try to promote their products and services through the word-of-mouth propagations among friends in the social networks. With online social networking sites such as Facebook, LinkedIn, Myspace, etc. attracting hundreds of millions of people, online social networks are also viewed as important platforms for effective viral marketing practice. This further motivates the research community to conduct extensive studies on various aspects of the influence maximization problem (e.g. [1], [2], [3], [4], [5], [6], [7]).

In [1] Kempe et al. proposed two basic stochastic influence cascade models, the *independent cascade (IC) model* and the *linear threshold (LT) model*, which are extracted from earlier work on social network analysis, interactive particle systems, and marketing. In both models, a social network is modeled as a directed graph $G = (V, E)$, where the vertices of V represents individuals and edges in E represent relationships and the orientations of the edges indicate the direction of influence. Roughly speaking, in the IC model each edge has an activation probability and influence is propagated by activated nodes independently activating their inactive neighbors based on the edge activation probabilities. In the LT model, each edge has a weight, each vertex has a threshold chosen uniformly at random, and a vertex becomes activated if the weighted sum of its active neighbors exceeds its threshold (see Section II for model details).

The two models characterize two different aspects of social interaction. The IC model focuses on individual (and independent) interaction and influence among friends in a social network. The LT model focuses on the threshold behavior in influence propagation, which we can frequently relate to — when enough of our friends bought a MP3, played a new computer game, or used a new online social networks, we may be converted to follow the same action. The random threshold λ_v is to model the uncertainty of individuals' conversion thresholds. In [1], Kempe et al. show that both models can be generalized and their generalized versions become equivalent. However, the basic IC and LT models stay as two distinct models.

The *influence spread* in the LT or IC model given a seed set is the expected number of activated vertices after the activation process ends. The *influence maximization* problem with parameter k is to find a seed set S of size k such that the influence spread of S is the largest. Kempe et al. show that the maximization problem in both models are NP-hard, and then they provide a greedy approximation algorithm for both models that achieves an approximation ratio of 63%. However, the greedy algorithm relies on the computation of influence spread given a seed set, the exact solution of which is left as an open problem in [1] for both models. Instead, they use Monte-Carlo simulations on influence cascade to estimate the influence spread, which makes the algorithm rather slow and not scalable — the algorithm takes days to select 50 seeds in a moderate sized graph of $30K$ nodes.

To overcome the inefficiency of the original greedy algorithm, a series of studies have been done to either improve the original greedy algorithm or propose new heuristic algorithms ([3], [4], [6], [7]). The improvements on the greedy algorithm reported in [4], [6], although significant, is still not scalable — it reduces the computation in the above examples from days to hours, but cannot scale up to larger networks, as reported in [7]. The new heuristic algorithms proposed in [3], [6], [7] are orders of magnitude faster than the fastest implementation of the greedy algorithm while maintaining a competitive level influence spread, making it a very promising direction. However, all of these heuristic algorithms are designed using specific properties of the IC model. In constrast, for the equally important LT model, the above heuristics do not apply and there is no scalable heuristic designed by utilizing special features of the LT model. In this paper, we fill this gap in the research of scalable influence maximization algorithms.

First, we study the influence computation in the LT model (Section III). We show that computing the exact influence spread in the LT model is #P-hard, even if there is only one seed in the network. Our reduction uses the interpolation technique, and is more involved than the simple reduction used in [7] to show the #P-hardness in the IC model. This hardness result closes the open problem left in [1] and further indicates that the greedy algorithm may have intrinsic difficulty to be made more efficient. To constrast with this hardness result, we show that computing influence spread in directed acyclic graphs (DAGs) can be done in linear time, which relies on an important linear relationship in activation probabilities between a node and its in-neighbors in DAGs.

Next, based on the fast influence computation for DAGs we propose the first scalable heuristic algorithm tailored for influence maximization in the LT model, which we refer to as the LDAG algorithm (Section IV). Our idea is to construct a local DAG surrpounding every node v in the network, and restrict the influence to v to be within the local DAG structure. This makes influence computation tractable and fast on a small DAG. To select local DAGs that could cover a significant portion of influence porpagation, we propose a fast greedy algorithm of adding nodes into the local DAG of a node v one by one such that the individual influence of these nodes to v is larger than a threshold parameter θ. After constructing the local DAGs, we combine the greedy approach of selecting seeds that provide the maximum incremental influence spread with a fast scheme of updating incremental influence spread of every node. Our combined fast local DAG construction and fast incremental influence update make the LDAG algorithm very efficient.

We conduct extensive simulations on both real-world networks and synthetic networks to demonstrate the scalability and effectiveness of our algorithm (Section V). Our simulation results show that our LDAG algorithm scales to networks with millions of nodes and edges, while the optimized greedy algorithm already take days for networks in the size of $64K$. In term of influence spread, our LDAG algorithm is always very close to that of the greedy algorithm in all test cases, showing that the LDAG algorithm is able to achieve the same level of influence spread while running in orders of magnitude faster than the greedy algorithm. We also compare the LDAG algorithm with some other heuristic algorithms such as PageRank [8] and the degree discount heuristic [6]. The results show that the LDAG algorithm always performs among the best algorithms in term of the influence spread in all test cases, while other heuristics performs poorly in some test cases. We believe this is because our LDAG algorithm is designed specifically for the LT model while other heuristics do not take advantages of the special features of the LT model or are designed for a different model.

In summary, our contributions are mainly twofold. First, we study influence computation in the LT model, close an open problem in [1] by showing that computing influence spread in general graphs in the LT model is #P-hard, and show on the other hand that influence computation in a DAG can be done in linear time. Second, we design the first scalable heuristic algorithm LDAG tailored for the LT model based on fast influence computation in DAGs, and demonstrate through experiments that LDAG is scalable and performs consistently among the best algorithms in term of influence spread.

A. Related work

Domingos and Richardson [9], [10] are the first to study influence maximization as an algorithmic problem. Their methods are probabilistic, however. Kempe, Kleinberg, and Tardos [1] are the first to formulate the problem as a discrete optimization problem, but one of the issues of their work is the scalability of their greedy algorithm.

In [4], Leskovec et al. present a "lazy-forward" optimization in selecting new seeds to significantly reduce the number of influence spread evaluations, but it is still slow and not scalable to large graphs with hundreds of thousands of nodes and edges, as demonstrated in our experiments.

Several recent work proposes different heuristic algorithms specifically designed for the IC model. In [3], Kimura and Saito propose shortest-path based influence cascade models and provide efficient algorithms to compute influence spread under these models. In [6], Chen et al. propose degree discount heuristics for the uniform IC model in which all edge probabilities are the same. In [7], Chen et al. propose maximum influence arborescence (MIA) heuristic for the general IC model. since all of them are designed using specific features of the IC model, they do not apply directly to the LT model. In term of design principle, our LDAG algorithm is similar to the MIA algorithm. Both uses local structures to make the influence computation tractable and reduce computation cost. However, the local structure and the influence computation are different: MIA uses local tree structures because that is the only structure making the influence computation tractable in the IC model, while LDAG uses local DAG structures, and thus could include more influence paths in the local structure.

Narayanam and Narahari [5] propose a Shapley value based heuristic SPIN for the LT model. However, SPIN only relies on the evaluation of influence spreads of seed sets, and thus does not use specific features of the LT model. Moreover, SPIN is not scalable, with running time comparable (as shown in [5]) or slower (as shown in our tests) than the optimized greedy algorithm.

Several studies design machine learning algorithms to extract influence cascade model parameters from real datasets [11], [12], [13], [14], which can be used to generate influence graphs studied in this paper.

II. Linear threshold model and the greedy algorithm for influence maximization

Following the definition in [1], we define the linear threshold (LT) model as follows. *An (LT) influence graph* is a weighted graph $G = (V, E, w)$, where V is a set of n vertices (nodes) and $E \subseteq V \times V$ is a set of m directed edges, and $w : V \times V \rightarrow [0, 1]$ is a weight function such that $w(u, v) = 0$ if and only if $(u, v) \notin E$,[1] and $\sum_{u \in V} w(u, v) \leq 1$. In the LT model, when given a seed set $S \subseteq V$, influence cascades in graph G as follows. First, every vertex v independently selects a threshold λ_v uniformly at random in range $[0, 1]$, which reflects our lack of knowledge of users' true thresholds as pointed out in [1]. Next, influence cascades in discrete steps $i = 0, 1, 2, \ldots,$, and let S_i denote the set of vertices activated at step i, with $S_0 = S$. In each step $i \geq 1$, a vertex $v \in V \setminus \cup_{0 \leq j \leq i-1} S_j$ is activated (and thus is in S_i) if the weighted number of its activated in neighbors reaches its threshold, i.e.

$$\sum_{u \in \cup_{0 \leq j \leq i-1} S_j} w(u, v) \geq \lambda_v.$$

The process stops at a step t when $S_t = \emptyset$. For convenience, we consider $S_i = \emptyset$ for all $i > t$. Let $\sigma_L(S)$ denote the expected number of activated nodes given the seed set S (i.e. the expected value of $|\bigcup_{i \geq 0} S_i|$) , where the expectation is taken among all λ_v values from their uniform distributions. We call $\sigma_L(S)$ the *influence spread* of seed set S in influence graph G under the linear threshold model.

As shown in [1], the linear threshold model defined above is equivalent to the reachability in the following random graphs, called *live-edge graphs*: Given an influence graph $G = (V, E, w)$, for every $v \in V$, select at most one of its incoming edges at random, such that edge (u, v) is selected with probability $w(u, v)$, and no edge is selected with probability $1 - \sum_u w(u, v)$. The selected edges are called *live* and all other edges are called *blocked*. Let R_G denote the random graph generated from G, which includes all vertices in V and all live edges selected. Thus, we have

Proposition 1 (Claim 2.6 of [1]): Given an influence graph G and a seed set S, the distribution of the set of active nodes in G with seed set S under the linear threshold model is the same as the distribution of the set of nodes reachable from S in the random graph R_G.

The above equivalence is central in analyzing the linear threshold model, including that showing its influence spread is submodular ([1]), proving that computing its influence spread is #P-hard, and designing scalable algorithm for influence maximization, as shown later in the paper. We say that a set function f on subsets of V is *submodular* if for any $S \subseteq T \subseteq V$ and any $u \in V \setminus T$, $f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T)$. Intuitively, submodularity indicates that f has diminishing margin returns when adding more nodes into a set. We say that f is monotone if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq V$. The following proposition states an important property of σ_L.

Proposition 2 (Theorem 2.5 of [1]): The influence spread σ_L under the linear threshold model is monotone and submodular.

The *influence maximization* problem under the linear threshold model is, when given the influence graph G and an integer k, finding a seed set S of size k such that its influence spread $\sigma_L(S)$ is the maximum. It is shown in [1] that finding the exact optimal solution is NP-hard, but because σ_L is monotone and submodular, a greedy algorithm has a constant approximation ratio. Algorithm 1 shows a generic greedy algorithm for any set function f. It simply execute in k rounds, and in each round it selects a new entry that gives the largest marginal increase in f. It is shown in [15] that for any monotone and submodular set function f with $f(\emptyset) = 0$, the greedy algorithm has an approximation ratio $f(S)/f(S^*) \geq 1 - 1/e$, where S is the output of the greedy algorithm and S^* is the optimal solution.

However, the generic greedy algorithm requires the evaluation of $f(S)$. In the context of influence maximization, the exact computation of $\sigma_L(S)$ was left as an open problem

[1] With such a definition of weight function $w()$, edge set E is redundant, but it is convenient to keep E as intuitively we are refering G as a graph.

Algorithm 1 Greedy(k, f)

```
initialize $S = \emptyset$
for $i = 1$ to $k$ do
    select $u = \arg\max_{w \in V \setminus S}(f(S \cup \{w\}) - f(S))$
    $S = S \cup \{u\}$
end for
output $S$
```

in [1]. In the next section, we close this open problem by showing that the exact computation of $\sigma_L(S)$ is #P-hard. This hardness result further indicates the intrinsic limitation of the greedy algorithm approach and motivates us to pursue scalable heuristic algorithms for the influence maximization problem in the LT model.

III. Influence computation in the LT model

In this section, we first show that computing exact influence spread for general graphs is #P-hard, and then provide a linear-time algorithm computing exact influence spread in directed acyclic graphs (DAGs). The DAG computation algorithm is the basis for our influence maximization solution presented in the next section.

A. #P-hardness for general graphs

We show that the exact computation of $\sigma_L(S)$ is #P-hard by applying Proposition 1 and using a reduction from the simple path counting problem.

Theorem 1: Given an influence graph G and a seed set S as the input, it is #P-hard to compute $\sigma_L(S)$ in G.

Proof. We reduce this problem from the problem of counting simple paths in a directed graph. Given a directed graph $G = (V, E)$, counting the total number of simple paths in G is #P-hard [16]. Let $n = |V|$. From G, we construct a graph G' by adding a new node $s \notin V$ and adding edges from s to all nodes in V. Let d be the maximum in-degree of any node in G'. For every edge e in G', we assign its weight $w(e) = w \leq 1/d$, where w is a constant to be determined later. By the selection of d, we know that the summation of weights among all incoming edges of a node is at most 1, satisfying the weight requirement of the LT model. Now we have an instance of the influence computation problem: G' with edge weights w and seed set $S = \{s\}$. We show that if for any eligible w computing $\sigma_L(S)$ in G' is solvable, we can count the number of simple paths in G, and thus show that influence computation is #P-hard.

Let $\mathcal{P}$ denote the set of all simple paths starting from s in G'. By the equivalence given in Proposition 1, we have[2]

$$\sigma_L(S) = \sum_{\pi \in \mathcal{P}} \prod_{e \in \pi} w(e).$$

Let B_i be the set of simple paths of length i in $\mathcal{P}$, for $i = 0, 1, \ldots, n$, and let $\beta_i = |B_i|$. Then we have

$$\sigma_L(S) = \sum_{i=0}^{n} \sum_{\pi \in B_i} \prod_{e \in \pi} w(e) = \sum_{i=0}^{n} \sum_{\pi \in B_i} w^i = \sum_{i=0}^{n} w^i \beta_i$$

With the above equation, we can set w to $n+1$ different values $w_0, w_1, \ldots, w_n$. For each w_i, by assumption we can obtain $\sigma_L(S)$ corresponding to w_i. Then we obtain a set of $n+1$ linear equations with $\beta_0, \ldots, \beta_n$ as variables. The coefficient matrix M of these equations is a Vandermonde matrix ($M_{ij} = w_i^j, i, j = 0, \ldots, n$). Thus the equations have a unique solution for $\beta_0, \ldots, \beta_n$ and is easy to compute.

Finally, we notice that for each $i = 1, \ldots, n$, there is a one-to-one correspondence between paths in B_i and simple paths of length $i - 1$ in graph G. Therefore, $\sum_{i=1}^{n} \beta_i$ give the answer to the total number of simple paths in graph G. Thus we complete the reduction and the theorem holds. □

From the proof we see that #P-hardness holds even if seed set S contains only one vertex. It is also #P-hard if we want to compute the influence to one vertex, since the influence spread to a graph is the sum of influence to all vertices in the graph.

B. Linear-time algorithm for DAGs

Theorem 1 shows that computing influence spread in general graphs is #P-hard. We now show that in directed acyclic graphs (DAGs), the computation instead can be done in time linear to the size of the graph.

Let $D = (V, E, w)$ be a DAG, and let $S \subseteq V$ be a seed set in D. For any $u \in V$, let $ap(u)$ be the *activation probability* of u, that is, the probability that u is activated in D under the LT model given the seed set S. By definition, $ap(u) = 1$ if $u \in S$. The following lemma shows the important linear relationship of activation probability in a DAG, the proof of which uses Proposition 1.

Lemma 3: For any $v \in V \setminus S$, we have

$$ap(v) = \sum_{u \in V \setminus \{v\}} ap(u) \cdot w(u, v). \qquad \text{(III.1)}$$

Proof. by Proposition 1, we consider the random graph R_D generated by live edges. Let E_u denote the event that edge (u, v) is selected as a live edge in R_D. By the live-edge graph generation method, we have $\Pr(E_u) = w(u, v)$. Let R_u denote the event that u is reachable from seed set S. By Proposition 1, $\Pr(R_u) = ap(u)$. Then for all $v \in V \setminus S$, we have

$$ap(v) = \sum_{u \in V \setminus \{v\}} \Pr(E_u) \cdot \Pr(R_u \mid E_u).$$

When the graph is a DAG, for any $(u, v) \in E$, whether u is reachable from S is independent of (u, v) being selected as a live edge, because there is no path from S to u that passes through v. Therefore, $\Pr(R_u \mid E_u) = \Pr(R_u) = ap(u)$, and the lemma holds. □

[2] By convention, if π only contains node s with no edges, $\prod_{e \in \pi} w(e) = 1$.

Algorithm 2 compute $ap(u)$ for all u in DAG D, with seed set S

```
1: $\forall u$ in $D$, $ap(u) = 0$; $\forall u \in S, ap(u) = 1$;
2: topologically sort all nodes reachable from $S$ in $D$ into
   a sequence $\rho$, with in-degree zero nodes sorted first.
3: for each node $u \in \rho \setminus S$ according to order $\rho$ do
4:     $ap(u) = \sum_{x \in N^{in}(u) \cap \rho} ap(x) \cdot w(x, u)$
5: end for
```

Based on the above lemma, we can design a linear-time algorithm to compute $ap(v)$ for all $v \in V$, as shown in Algorithm 2. We use $N^{in}(u)$ and $N^{out}(u)$ to denote the in-neighbors and outneighbors of node u, respectively. Topological sort in line 2 is to total order nodes and respect the partial order given by the DAG D. Then in lines 3–5 we compute $ap(u)$'s using Equation (III.1) for all u's following the topological order, which guarantees that when we compute $ap(u)$, all $ap(x)$'s of u's in-neighbors x have been computed. The operation "$\cap\rho$" in line 4 is mainly for convenience of reusing this code in Algorithm 5, and could be omitted here without affecting the computation. It is clear that the algorithm computes all $ap(v)$'s in time linear to the size of the DAG D. Finally, we can add all $ap(v)$'s together to obtain the influence spread of the seed set S in graph D.

IV. LDAG ALGORITHM FOR INFLUENCE MAXIMIZATION

In the previous section, we show that computing influence in a DAG is easy. However, real social networks are not DAGs typically, so we cannot apply the algorithm for DAG directly. In [1] the authors use Monte-Carlo simulations to estimate the influence spread, making their greedy algorithm very inefficient. To allow efficient computation of influence in general social networks, we compute a local DAG for every node v in the graph and use influence to v propagated through its local DAG to approximate the influence in the original network. We refer to this as the *LDAG influence model*. The intuition is that (a) influence computation in DAGs are fast, and (b) influence cascade in the LT model is typically local because of the exponential dropoff in the probability of influence propagation. However, the local DAGs need to be carefully select in order to cover a significant portion of influence while allowing fast computations. In this section, we first introduce the general framework of the LDAG influence model, and then study local DAG constructions and influence maximization in the LDAG model.

A. LDAG influence model

In the LDAG influence model, each node v in the graph $G = (V, E, w)$ is associated with a local DAG $LDAG(v)$, a subgraph of G containing v. We also refer to $LDAG(v)$ as the *LDAG rooted at* v. Given a seed set S, we assume that the influence from S to v is only propagated within $LDAG(v)$ according to the LT model. Let $ap(v, S)$ be the activation probability of v when influence is propagated within $LDAG(v)$. Then the influence spread of S in the LDAG influence model, denoted as $\sigma_D(S)$,[3] is given by

$$\sigma_D(S) = \sum_{v \in V} ap(v, S). \qquad \text{(IV.2)}$$

Given G and LDAGs rooted at all nodes, maximizing $\sigma_D(S)$ is still NP-hard.

Theorem 2: The influence maximization problem in the LDAG influence model is NP-hard.

Proof. (Sketch). We reduce the problem from the Vertex Cover problem. For every node v, we use v together with v's in-neighbors and the edges between these in-neighbors and v to construct the LDAG rooted at v. The rest of the proof is similar to the one in [1]. □

Although finding the optimal seed set in the LDAG influence model is NP-hard, computing $\sigma_D(S)$ given S is in polynomial-time because all computations are on DAGs. It is also easy to see that $\sigma_D(S)$ is still monotone and submodular (because every $ap(v, S)$ is monotone and submodular on S). Therefore, we can use the greedy Algorithm 1 on $\sigma_D()$ to find S with an approximation ratio of $1 - 1/e$.

In the following, we address two tasks to make our LDAG algorithm effective and efficient. First, for each node v, we need to compute its local DAG $LDAG(v)$ that covers a significant portion of influence from other nodes to v. Second, we make the algorithm more efficient when combining the greedy Algorithm 1 and the DAG influence computation of Algorithm 2.

B. Local DAG construction

We first need to compute the local DAG surrounding v such that it covers a significant portion of influence from other nodes to v while ignoring nodes that has only very small influence to v. This is formally defined as the following maximization problem. Let $Inf_G(u, v)$ be the influence probability from u to v in graph G, that is, the probability that v is activated when u is the only seed.

Definition 1 (MAX-LDAG Problem): Given an influence graph $G = (V, E, w)$, a node $v \in V$, and a threshold $\theta \in [0, 1]$ as the input, the MAX-LDAG problem is to compute a DAG $D = (X, Y, w)$, such that (a) D is a subgraph of G, i.e. $X \subseteq V$, $Y \subseteq E$; (b) $v \in X$; (c) $Inf_D(u, v) \geq \theta$ for all $u \in X$; and (d) $\sum_{u \in X} Inf_D(u, v)$ is the maximum among all DAGs satisfying (a), (b), and (c).

Condition (c) above is for controlling the size of the LDAG using parameter θ, which represents a tradeoff between efficiency (smaller DAGs and thus faster computations) and accuracy (larger DAGs and more accurate influence result). Condition (d) aims at maximizing the sum of individual influence from any node u in the DAG to its root

[3] Symbol D in $\sigma_D()$ is a notation standing for DAG, in order to differentiate from $\sigma_L()$. It does not represent a specific DAG D as used elsewhere in the paper.

v. This is a heuristic for covering as much as possible the influence propagation in the selected DAG. Unfortunately the problem MAX-LDAG is NP-hard, as shown by the following theorem.

Theorem 3: Problem MAX-LDAG is NP-hard.

Proof. The reduction is from the Regular Graph Vertex Cover problem. An instance I_1 of the Regular Graph Vertex Cover problem includes a positive integer k and a graph $G = (V, E)$ with n vertices all having degree d. The decision problem is to determine if there exists a vertex set $S \subseteq V$ with $|S| = k$ such that every edge in E is incident to some vertex in S. The decision version of the MAX-LDAG problem is that, given as input an influence graph $G' = (V', E', w')$, a special vertex s in the graph, a threshold θ, and an influence target T, whether there exists a DAG D such that $D = (X, Y, w')$ is a subgraph of G', and for each $u \in X$, $Inf_D(u) \geq \theta$, and $\sum_{u\in X} Inf_D(u, s) \geq T$.

We construct an instance I_2 of the above problem from I_1 as follows. For each vertex $v \in V$, we construct two vertex v_1 and v_2 in G' and let $(v_1, v_2) \in E'$ with weight 1. For each edge $(v, u) \in E$, let $(v_2, u_1) \in E'$ with weight $1/d$ and $(u_2, v_1) \in E'$ with weight $1/d$. Then, we construct another vertex r in G' and for all $v \in V$,let $(v_1, r) \in E'$ with weight $1/n$. Vertex r is our special vertex s of the instance I_2. Finally, let $\theta = \frac{1}{nd}$ and $T = 4 - \frac{2k}{n}$.

First, if there exists a vertex cover S of I_1 with $|S| = k$, we construct a DAG $D = (X, Y, w')$ in G' as follows. For each $v \in S$, we delete (v_1, v_2) in G', and get a subgraph D. It is straightforward to check that D is a DAG, due to the fact that S is a vertex cover in G. For each $v \in S$, since v_1 has only one path to r in D, therefore, $Inf_D(v_1) = 1/n$. For each $u \notin S$, the path from u_1 to r has two forms: one is (u_1, r) and the other is (u_1, u_2, v_1, r) where v_1 is an adjacent vertex of u_2. Therefore, $Inf_D(u_1, r) = \frac{1}{n} + \sum_{u_2,(v_1,u_2)\in E'} \frac{1}{nd} = \frac{2}{n}$. For u_2's, $Inf_D(u_2, r) = 1/n$. Then for each $v \in S$, in D there are two types of paths from v_2 to r, one via some u_1 with $u \notin S$ and the other via some u_1 with $u_1 \in S$. Therefore, $Inf_D(v_2, r) = \sum_{u\in S,(u,v)\in E} \frac{1}{nd} + \sum_{u\notin S,(u,v)\in E} \frac{2}{nd}$. Obviously, condition (c) of Definition 1 is satisfied for all vertices in D. Next, we have

$$\begin{aligned}
&\sum_{u\in X} Inf_D(u,r)\\
&\quad = \sum_{v\in S}(\frac{1}{n} + \sum_{u\in S,(u,v)\in E} \frac{1}{nd} + \sum_{u\notin S,(u,v)\in E} \frac{2}{nd})\\
&\qquad + \sum_{v\notin S}(\frac{2}{n} + \frac{1}{n})\\
&\quad = 3 - \frac{2k}{n} + \frac{1}{nd} \sum_{v,u\in S,(u,v)\in E} 1 + \frac{2}{nd} \sum_{v\in S,u\notin S,(u,v)\in E} 1\\
&\quad = 3 - \frac{1}{nd} \sum_{v,u\in S,(u,v)\in E} 1 = 4 - \frac{2k}{n}
\end{aligned}$$

Algorithm 3 FIND-LDAG(G, v, θ), compute $LDAG(v, \theta)$

```
1: $X = \emptyset$; $Y = \emptyset$; $\forall u \in V, Inf(u, v) = 0$; $Inf(v, v) = 1$
2: while $\max_{u\in V\setminus X} Inf(u, v) \geq \theta$ do
3:     $x = \arg\max_{u\in V\setminus X} Inf(u, v)$
4:     $Y = Y \cup \{(x, u) \mid u \in X\}$ /* adding edges */
5:     $X = X \cup \{x\}$ /* adding the node */
6:     for each node $u \in N^{in}(x)$ do
7:         $Inf(u, v)$ += $w(u, x) \cdot Inf(x, v)$
8:     end for
9: end while
10: return $D = (X, Y, w)$ as the $LDAG(v, \theta)$
```

The third equality above is due to $\sum_{u\notin S,(u,v)\in E} 1 = d - \sum_{u\in S,(u,v)\in E} 1$ for all $v \in S$, while the last equality is derived from the fact $kd = \sum_{v\in S,u\in V,(u,v)\in E} 1 = |E| + \frac{1}{2}\sum_{v,u\in S,(u,v)\in E} 1$ because S is a vertex cover. Therefore, $\sum_{u\in X} Inf_D(u, r) \geq T$ is satisfied.

On the other hand, if we have a solution to I_2, the solution must break the cycle $(v_1, v_2, u_1, u_2, v_1)$,where $(v, u) \in E$. If the edge $(v_2, u_1), v \neq u$ was deleted, then we claim that deleting the edge (v_1, v_2) instead could increase the total influence. In fact, after changing the deletion, the set of paths from v_2 to r increases without any influence loss. Therefore, we can always get a solution for I_2 that only deletes edges of the form (v_1, v_2). Note that the set of deleted edges correspond to a vertex cover of G, because they remove all cycles of the form $(v_1, v_2, u_1, u_2, v_1)$, which correspond to edges in G. However, by the proof above, the total influence is greater than or equal to $4 - 2k/n$ if and only if we delete less than or equal to k such edges, which means that we can find a vertex cover of size at most k for G. □

To circumvent the NP-hardness result, we use an efficient greedy heuristic algorithm shown in Algorithm 3 to compute a local DAG $LDAG(v, \theta)$ for each node v given a threshold θ. Initially the DAG D to be computed is empty, and only the influence probability from v to itself is 1 and all others are 0. The algorithm add nodes into D one by one. Each time, the algorithm selects the node x that has the largest influence to v among all nodes not in D, and add x together with all its outgoing edges to nodes already in D into DAG D. After x is added into D, we need to update the influence probability for those in-neighbors u of x not yet in D, since they may have new paths through x to influence v (line 7). The process ends when the x selected has influence to v less than θ. It is easy to note that the algorithm has the similar structure as the classic Dijkstra shortest-path algorithm. Let n_v be the number of nodes in $LDAG(v, \theta)$, and ℓ_v be the volume of $LDAG(v, \theta)$, which is the sum of in-degrees of nodes in $LDAG(v, \theta)$. Then we know that our LDAG construction algorithm (Algorithm 3) has an efficient implementation in $O(\ell_v + n_v \log \ell_v)$ time (using a Fibonacci heap).

Algorithm 4 compute $\alpha_v(u)$ for all u in DAG D

```
1: $\forall u$ in $D$, $\alpha_v(u) = 0$; $\alpha_v(v) = 1$;
2: topologically sort all nodes that can reach $v$ in $D$ into a sequence $\rho$, with $v$ sorted first.
3: for each node $u \in \rho \setminus (S \cup \{v\})$ according to order $\rho$ do
4:     $\alpha_v(u) = \sum_{x \in N^{out}(u) \cap \rho} w(u,x) \cdot \alpha_v(x)$
5: end for
```

While our LDAG algorithm does not provide an approximation guarantee, it is very efficient and produces very good results in practice. In our experiment section, we will show that our FIND-LDAG algorithm leads to much better result in influence maximization than randomly selected local DAGs, and its performance is already very close to the gready Algorithm 1 for influence maximization, suggesting that the room for further improving the local DAG construction is small.

C. LDAG algorithm

After selecting the LDAGs rooted at all nodes, we may simply follow the greedy Algorithm 1 to select the k seeds, and use Algorithm 2 to compute influence spread. This is not very efficient, however, as we now explain. Let $InfSet(u) = \{v \in V \mid u \in LDAG(v, \theta)\}$, which represents the set of nodes that could be influenced by u in the LDAG influence model. After a new seed s is selected, for every $v \in InfSet(s)$, we need to recompute the incremental influence spread of every node u in $LDAG(v)$ because it may have changed. The naive computation would be for every such u, using Algorithm 2 to compute the influence to v in $LDAG(v)$ when u is added to the seed set. Let n_v and m_v denote the number of nodes and edges of $LDAG(v)$, respectively. Since computing influence in $LDAG(v)$ given a seed set takes $O(m_v)$ time using Algorithm 2, and we need to try all possible nodes in $LDAG(v)$ as the next seed, it takes $O(n_v m_v)$ time to compute the incremental influence spread of any node in $LDAG(v)$. In the following, we utilize the linear relationship shown in Lemma 3 to reduce the above time from $O(n_v m_v)$ to $O(m_v)$.

Informally, in a DAG D, the activation probabilities of nodes u and v have a linear relationship, so if we know the linear coefficient α and the activation probability $ap(u)$ of u, we can immediately know that when u is selected as a seed, the additional influence u imposes on v is $(1 - ap(u))\alpha$. More specifically, consider a DAG $D = (V, E, w)$ and a seed set $S \subseteq V$, and for all $u \in V$, let $ap(u)$ denote the activation probability of u as computed by Algorithm 2. For two nodes $u, v \in V \setminus S$, let the *incremental influence of u to v in D* be the amount of increase in $ap(v)$ when u is selected as an additional seed. We have

Lemma 4: For two nodes $u, v \in V \setminus S$, the incremental influence of u to v in D is $(1 - ap(u)) \cdot \alpha_v(u)$, where $\alpha_v(u)$ is calculated in Algorithm 4.

Algorithm 5 LDAG algorithm for seed selection

```
/* preparation phase */
set $S = \emptyset$
$\forall\, v \in V, IncInf(v) = 0$
for each node $v \in V$ do
    generate $LDAG(v, \theta)$ using Algorithm 3
    /* $InfSet(v)$'s are derived from $LDAG(v, \theta)$'s */
    $\forall u \in LDAG(v, \theta)$, set $ap_v(u) = 0$
    $\forall u \in LDAG(v, \theta)$, compute $\alpha_v(u)$ using Algorithm 4
    for each $u$ in $LDAG(v, \theta)$ do
        $IncInf(u) \mathrel{+}= \alpha_v(u)$
    end for
end for
/* main loop for selecting $k$ seeds */
for $i = 1$ to $k$ do
    $s = \arg\max_{v \in V \setminus S}\{IncInf(v)\}$
    for each $v \in InfSet(s) \setminus S$ do
        /* update $\alpha_v(u)$ for all $u$'s that can reach $s$ in $LDAG(v, \theta)$ */
        $\Delta\alpha_v(s) = -\alpha_v(s)$; $\forall u \in S, \Delta\alpha_v(u) = 0$
        topologically sort all nodes that can reach $s$ in $LDAG(v, \theta)$ into a sequence $\rho$, with $s$ sorted first.
        compute $\Delta\alpha_v(u)$ for all $u \in \rho$, using lines 3–5 of Algorithm 4, where $\rho \setminus (S \cup \{v\})$ is replaced by $\rho \setminus (S \cup \{s\})$ and $\alpha_v()$ is replaced by $\Delta\alpha_v()$.
        $\alpha_v(u) \mathrel{+}= \Delta\alpha_v(u)$, for all $u \in \rho$
        $IncInf(u) \mathrel{+}= \Delta\alpha_v(u) \cdot (1 - ap_v(u))$ for all $u \in \rho$
        /* update $ap_v(u)$ for all $u$'s reachable from $s$ in $LDAG(v, \theta)$ */
        $\Delta ap_v(s) = 1 - ap_v(s)$; $\forall u \in S, \Delta ap_v(u) = 0$
        topologically sort all nodes reachable from $s$ in $LDAG(v, \theta)$ into a sequence $\rho$, with $s$ sorted first.
        compute $\Delta ap_v(x)$ for all $u \in \rho$, using lines 3–5 of Algorithm 2, where $\rho \setminus S$ is replaced by $\rho \setminus (S \cup \{s\})$ and $ap()$ is replaced by $\Delta ap_v()$
        $ap_v(u) \mathrel{+}= \Delta ap_v(u)$, for all $u \in \rho$
        $IncInf(u) \mathrel{-}= \alpha_v(u) \cdot \Delta ap_v(u)$ for all $u \in \rho$
    end for
    $S = S \cup \{s\}$
end for
return $S$
```

Proof. (Sketch). The proof is done by repeatedly expanding $ap(v)$ using Equation (III.1). □

Algorithm 4 shows that we can compute all $\alpha_v(u)$'s for all $u \in D$ in time linear to the size of the DAG. Then by the above lemma we can immediately obtain the incremental influence of all nodes u to v. Therefore, by using the linear relationship, we reduce the incremental influence computation from $O(n_v m_v)$ to $O(m_v)$.

Algorithm 5 shows our full algorithm of selecting k seeds. For ease of reading, we add subscript v on variables such as $ap_v(u)$ to denote that the variables are for $LDAG(v,\theta)$ rooted at node v. Lines 1–12 are for the preparation phase, in which $LDAG(v,\theta)$'s are computed (and $\mathit{InfSet}(v)$'s are derived from them), $ap_v(u)$'s are initialized to 0 since there is no seed, and $\alpha_v(u)$'s are computed. We use a max-heap to maintain $\mathit{IncInf}(u)$ for every $u \in V$, which is the incremental influence of u to all nodes if u is selected as the seed. By Lemma 4, initially $\mathit{IncInf}(u) = \sum_{v \in \mathit{InfSet}(u)} \alpha_v(u)$.

The main loop in lines 14–31 uses the greedy approach to select k seeds one by one. After selecting a seed s with the highest incremental influence (line 15), we need to update related $ap(u)$'s and $\alpha(u)$'s in order to calculate the new incremental influence of nodes. The nodes needed to be updated are those sharing at least one $LDAG(v,\theta)$ with s, i.e. those u's in $LDAG(v,\theta)$ for all $v \in \mathit{InfSet}(s) \setminus S$. In each of such $LDAG(v,\theta)$'s, when s is selected as the next seed, by Algorithm 4 we can see that only those nodes u that can reach s in $LDAG(v,\theta)$ need to update their $\alpha_v(u)$'s. Due to the linear relationship, the update of those $\alpha_v(u)$'s can be done by simply computing their changes $\Delta\alpha_v(u)$'s through reusing the code in Algorithm 4, with the initial condition $\Delta\alpha_v(s) = -\alpha_v(s)$ because $\alpha_v(s)$ is changed to zero when s is selected as a seed (lines 18–21). These nodes with updated $\alpha_v(u)$'s do not change their $ap(u)$'s because they are not reachable from s. Thus their incremental influence $\mathit{IncInf}(u)$ needs to be updated by adding $\Delta\alpha_v(u)(1 - ap(u))$ (line 22). Similarly, for all nodes u reachable from s in $LDAG(v,\theta)$, they need to update $ap_v(u)$ but not $\alpha_v(u)$, and the update of $ap_v(u)$ follows Algorithm 2 by computing the changes $\Delta ap_v(u)$, with the initial condition $\Delta ap_v(s) = 1 - ap_v(s)$ and for all seeds $u \in S$ $\Delta ap_v(u) = 0$ (lines 24–27). Finally the incremental influence $\mathit{IncInf}(u)$ of each u reachable from s decreases by $\alpha_v(u)\Delta ap_v(u)$ (line 28).

Time and space complexity of LDAG algorithm. Let n be the total number of nodes in G. Let m_θ and $\bar{m}_\theta$ be the maximum size and average size (in terms of the number of edges) of $LDAG(v,\theta)$ among all $v \in V$, respectively. Let n_θ be the maximum size of $\mathit{InfSet}(v)$ among all $v \in V$. The $LDAG(v,\theta)$ is computed by a Dijkstra-like algorithm in Algorithm 3, and let $\bar{t}_\theta$ be the average time to compute one $LDAG(v,\theta)$ among all $v \in V$. Note that $\bar{m}_\theta = O(\bar{t}_\theta)$, and for sparse graphs (such as typical social networks) $\bar{t}_\theta = \bar{m}_\theta$.

For every node $v \in V$, the LDAG algorithm maintains $LDAG(v,\theta)$ and $\mathit{InfSet}(v)$, and for every u in $LDAG(v,\theta)$, the algorithm maintains $ap_v(u)$ and $\alpha_v(u)$. The algorithm also maintains a max-heap for $\mathit{IncInf}(v)$ for all $v \in V$. Thus, it is easy to see that space needed is proportional to the total size of all LDAGs, which is $O(n\bar{m}_\theta)$.

The preparation phase of our LDAG algorithm (lines 1–12) generates all $LDAG(v,\theta)$'s, and for each of them compute $\alpha_v(u)$'s. The LDAG construction takes $O(n\bar{t}_\theta)$ time, and $\alpha_v(u)$ computations take $O(n\bar{m}_\theta)$ time totally. The initialization of the max-heap for $\mathit{IncInf}()$ takes $O(n)$ time. Thus the entire preparation phase takes $O(n\bar{t}_\theta)$ time.

Table I
STATISTICS OF FOUR REAL-WORLD NETWORKS.

Dataset	NetHEPT	Epinions	Amazon	DBLP
number of nodes	15K	76K	262K	655K
number of edges	31K	509K	1.2M	2.0M
average degree	4.12	13.4	9.4	6.1
maximal degree	64	3079	425	588
number of connected components	1781	11	1	73K
largest component size	6794	76K	262K	517K
average component size	8.6	6.9K	262K	9.0

Note: Directed graphs are treated as undirected graphs in these statistics.

In each iteration of the main loop (lines 14–31), selecting a new seed from the max-heap takes constant time, and for each $v \in \mathit{InfSet}(v) \setminus S$, updating $\alpha_v(u)$'s and $ap_v(u)$'s for all $u \in LDAG(v,\theta)$ takes $O(m_v)$ time, where m_v is the size of $LDAG(v,\theta)$. Then updating each $\mathit{IncInf}(u)$ takes $O(\log n)$ time on the max-heap. Therefore, one iteration of the main loop takes $O(\sum_{v \in \mathit{InfSet}(s) \setminus S} m_v \cdot \log n) = O(n_\theta m_\theta \log n)$. Therefore, to select k seeds, the total time is $O(n\bar{t}_\theta + kn_\theta m_\theta \log n)$. Note that without using the linear relationship speedup, the time complexity would be $O(n\bar{t}_\theta + kn_\theta m_\theta (m_\theta + \log n))$.

V. EXPERIMENTS

We use experiments on real-world networks as well as synthetic networks to demonstrate the effectiveness and the efficiency of our LDAG algorithm.

A. *Setting up the experiments*

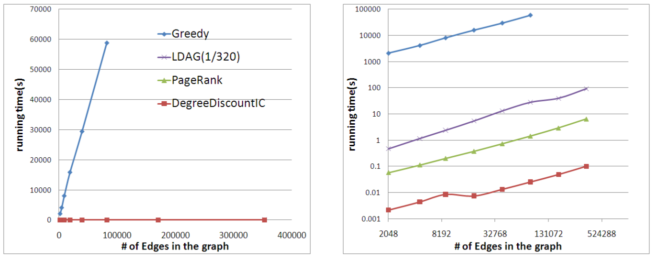
The four real-world networks we use and their basic statistics are summarized in Table I, which include: (a) NetHEPT, an academic collaboration network extracted from ”High Energy Physics - Theory” section of the e-print arXiv (http://www.arXiv.org), with nodes representing authors and edges representing coauthorship relations; (b) DBLP, the Computer Science Bibliography Database maintained by Michael Ley (http://www.informatik.uni-trier.de/~ley/db/), again with nodes representing authors and edges representing coauthorship relations; (c) Epinions, the Who-trust-whom network of Epinions.com [18], where nodes are members of the site and a directed edge from u to v means v trust u (and thus u has influence to v); and (d) Amazon, the Amazon product co-purchasing network [19] dated on March 2, 2003, where nodes are products and a directed edge from u to v means product v is often purchased with product u (and thus u has influence to v). These datasets vary in size and features and thus can test the performance of the algorithms in different cases. We also use synthetic power-law degree graphs generated by the DIGG package [20] to test the scalability of our algorithm with different sized graphs of the same feature.

For the graphs we tested, we use the following two methods to generate the influence weights on all edges: (a) the uniform method, in which for every node v in a graph with in-degree d_v, we set the weight of every incoming edge of v to be $1/d_v$; and (b) the random method, in which the weight of every edge is generated uniformly at random in the range $[0, 1]$, and then we normalize the weights of all incoming edges of a node v so that they sums to 1.

In the experiments, we compare our LDAG algorithm with several other algorithms for influence maximization. As explained in the Introduction, to the best of our knowledge we do not find any other algorithm that is designed using specific features of the LT model. Thus we compare our algorithm with a few other more or less generic algorithms and heuristics, as we list below.

- LDAG(θ): Our LDAG Algorithm 5 with threshold parameter θ. In all of our tests, we use $\theta = 1/320$.[4]
- Greedy: The greedy Algorithm 1 on the LT model with the lazy-forward optimization of [4]. For each candidate seed set S, 20000 simulations is run to obtain an accurate estimate of $\sigma_L(S)$.
- SPIN: A heuristic algorithm based on the Shapley values [5]. We use 20000 simulations to estimate $\sigma_L(S)$ to be consistent with Greedy and other algorithms. Also, in computing Shapley values, 10000 random permutations and 400 random nodes in each permutation are used, same as in [5] on a graph similar to NetHEPT.
- DegreeDiscountIC: The degree discount heuristic of [6] developed for the uniform IC model with a propagation probability of $p = 0.01$. Although this heuristic is designed specifically for the uniform IC model, we use it as a general heuristic in the class of degree centrality heuristics. It performs much better than the pure degree heuristic, as shown in [6] as well as in our own tests in the LT model.
- PageRank: The popular algorithm used for ranking web pages [8]. The transition probability along edge (u, v) is $w(v, u)$. Intuitively, $w(v, u)$ in the LT model indicates the strength of influence of v to u, and thus we use it in the reverse direction to be the "vote" of u on v's rank, as interpreted by the PageRank algorithm. We select k nodes with the highest PageRanks as the seeds. We use 0.15 as the restart probability for PageRank, and we use the power method to compute the PageRank values. The stopping criteria is when two consecutive iterations differ for at most 10^{-4} in L_1 norm.

We run 20000 simulations to accurately estimate $\sigma_L(S)$ for every seed set S obtained from any heuristic algorithms, which matches the accuracy of the greedy algorithm. All experiments are run on a server with 2.33GHz Quad-Core Intel Xeon E5410, 32G memory, and Microsoft Windows Server 2003.



(a) normal scale (b) log-log scale

Figure 1. Scalability of different algorithms in synthetic datasets. Each data point is an average of ten runs.

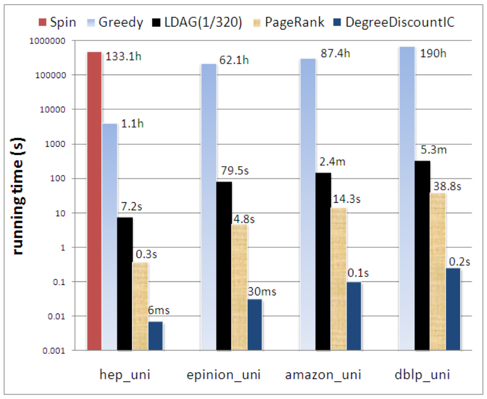


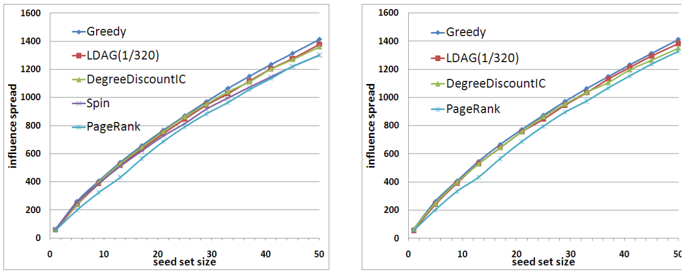
Figure 2. Running time of different algorithms in the four datasets

B. Results of the experiments

Scalability on the synthetic dataset. We first test the scalability of the algorithms, using a family of synthetic power-law graphs generated by the DIGG package [20]. We generate graphs with doubling number of nodes, from $2K$, $4K$, up to $256K$, using power-law exponent of 2.16. Each size has 10 different random graphs and our running time result is the average among the runs on these 10 graphs. Every undirected edge is treated as two directed edges, and we use uniform method to generate edge weights for the directed edges. We run different algorithms to select 50 seeds in each graph. Figure 1 shows our scalability test result, with normal scale in (a) and log-log scale in (b) for better viewing of different curves. The results clearly show that Greedy is not scalable — it almost runs 16 hours for the graphs with about $80K$ edges. LDAG scales quite well, only taking 1.5 minutes for graphs with $350K$ edges.

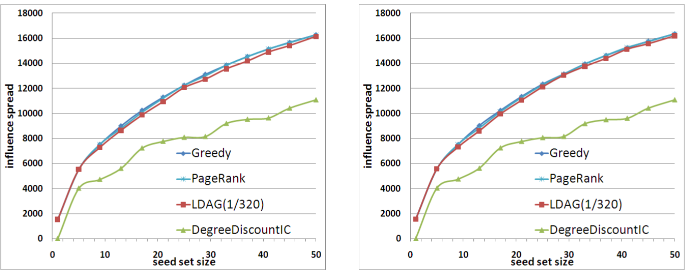
The scalability of our LDAG algorithm is further demonstrated in our tests on real networks, as shown in Figure 2. The figure shows the running time of different algorithms on the four real-world networks of different sizes, and weights are generated using the uniform method (the results on the random method are similar and thus omitted). We can see that even for the largest DBLP graph with $655K$ nodes and $2M$ edges, LDAG only takes 5 minutes to finish, while

[4] We found that a fairly loose range of θ from $1/80$ to $1/640$ produce similar results in term of influence spread in all of our test cases (one is within 3.6% while all the rest are within 2% of the case $\theta = 1/320$), and thus we choose $\theta = 1/320$ for all cases.



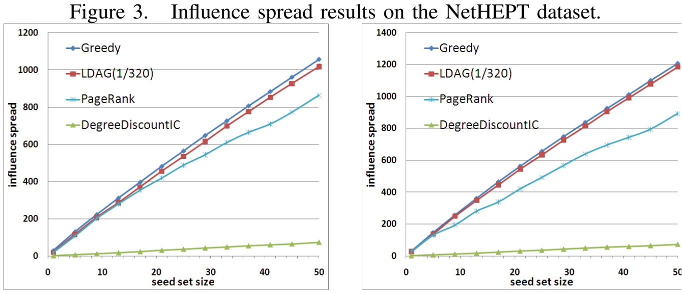
(a) uniform weights (b) random weights

Figure 3. Influence spread results on the NetHEPT dataset.



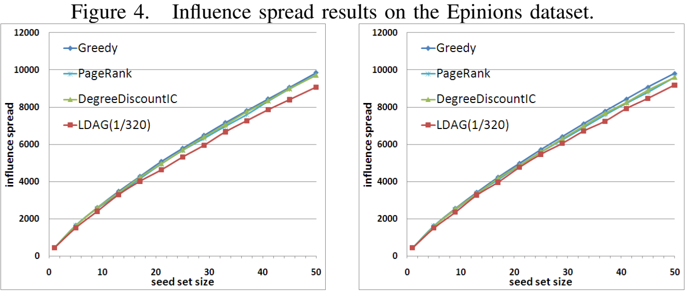
(a) uniform weights (b) random weights

Figure 4. Influence spread results on the Epinions dataset.



(a) uniform weights (b) random weights

Figure 5. Influence spread results on the Amazon dataset.



(a) uniform weights (b) random weights

Figure 6. Influence spread results on the DBLP dataset.

Table II
AVERAGE NUMBER OF NODES AND EDGES IN LDAGS IN THE NETWORKS WITH UNIFORM WEIGHTS AND RANDOM WEIGHTS, RESPECTIVELY.

Dataset	NetHEPT	Epinions	Amazon	DBLP
number of nodes (uniform)	37.2	51.1	48.7	47.1
number of edges (uniform)	285	162	82.1	159
number of nodes (random)	38.4	53.8	49.6	58.0
number of edges (random)	293	170	82.3	158

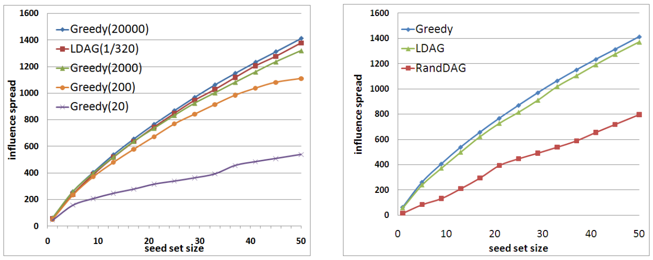
Greedy takes 190 hours. Therefore, our LDAG algorithm has much better scalability and is more than three orders of magnitude faster than the greedy algorithm. One of the key reasons for the efficiency of LDAG is the small LDAG size computed by the algorithm, as reported in Table II.

Among other heuristics algorithms, SPIN turns out to have very poor scalability, running very slow even for the smallest NetHEPT graph of $15K$ nodes. For this reason we do not include SPIN in our scalability test or in other larger datasets. We found that our result on SPIN has some inconsistency as reported in [5], which shows that SPIN runs a little faster than the optimized greedy algorithm. There could be some detailed implementation difference that causes the descrepancy. However, even adopting the result of [5], which are all on smaller networks, SPIN is only a few times faster than Greedy and thus we do not believe SPIN is a scalable algorithm. PageRank and DegreeDiscountIC have a better scalability than LDAG, but we will show next that their influence spread is not as good as LDAG.

Influence spread in the real-world datasets To compare influence spread generated by different algorithms, we run all algorithms on four real-world networks using both uniform weights and random weights, and we select from 1 seed to 50 seeds. Our tests generate eight different set of results, as reported in Figures 3–6. For ease of reading, the legend of each figure lists the algorithms in the same order as their corresponding influence spread with 50 seeds.

Several conclusions can be made from these test results. First, comparing with Greedy, LDAG consistently matches the performance of Greedy. The largest difference is in the DBLP dataset where LDAG is 6.7% and 5.5% lower than Greedy (for uniform and random weights respectively),[5] while in other datasets LDAG is very close to Greedy or essentially the same as Greedy. Second, comparing with other heuristics PageRank and DegreeDiscountIC, LDAG perofrms consistently well, while DegreeDiscountIC performs poorly in Amazon dataset (more than 90% lower than LDAG) and Epinions dataset (more than 34% lower than LDAG), and PageRank performs significantly worse than LDAG in Amazon dataset (9.4% and 19.9% lower in uniform weights and random weights, respectively). PageRank and DegreeDiscountIC do performs a few percentage better than LDAG in the DBLP dataset. However, the unstable performance of PageRank and DegreeDiscountIC makes them not suitable as a general heuristic for the LT model to be applied for different type of networks.

[5] All percentages reported are taken as the average of percentages from using one seed to using 50 seeds.



(a) (b)

Figure 7. Comparing influence spread in NetHEPT network with uniform weights, with the following different setup: (a) Greedy with 20, 200, 2000, and 20000 simulation runs; (b) using DAGs computed by our FIND-LDAG algorithm vs. randomly computed DAGs.

We believe that our test results provide a balanced view among these different algorithms. The main advantage of our LDAG algorithm is that it performs consistently among the best algorithms in all tests, and we attribute it to the design of LDAG tailored specifically to the LT influence model.

Greedy algorithm with smaller number of simulations. We run Greedy with a smaller number of simulations to estimate influence spread (20, 200 and 2000, instead of 20000) to see if we can significantly reduce Greedy's running time while obtaining similar influence spread. Our results in Figure 7(a) clearly shows that it is not the case: Greedy with 20, 200 and 2000 simulations are 56%, 13% and 4.3% worse than Greedy with 20000 simulations, and all are worse than our LDAG algorithm. In particular, Greedy(20) has running time comparable to LDAG, but has much worse influence spread. Therefore, the greedy algorithm cannot maintain high influence spread when reducing its number of simulations in estimating incremental influence spread.

Importance of finding a good LDAG. Finally, we verify the effectiveness of our LDAG construction Algorithm 3. To do so, we compare it against an algorithm that randomly generates LDAGs: at each step, the algorithm randomly selects one node from the in-neighbors of all nodes already in the LDAG and adds the node as well as its outgoing edges into the LDAG. We denote this algorithm as RandDAG. Since threshold θ does not apply to RandDAG, we directly control the size of LDAGs, and in our test RandDAG always select 30 nodes in each LDAG. To make a fair comparison, we also modify Algorithm 3 so that it also selects 30 nodes for each LDAG instead of using parameter θ. We run both algorithms in the NetHEPT network using uniform weights. Our results in Figure 7(b) show that our greedy approach of generating LDAGs perform much better than randomly generated LDAGs, indicating that indeed LDAGs should be carefully selected. On the other hand, our greedy LDAG construction already results in influence spread very close to that of the greedy seed selection algorithm, so the benefit of further improvement in LDAG construction is small.

To summarize, the experimental results demonstrate that our LDAG algorithm performs consistently among the best algorithms in term of the influence spread and can scale to graphs with millions of nodes and edges, while other algorithms either do not scale or do not have stable performance in influence spread. Therefore, we believe that our LDAG algorithm is suitable as the scalable solution to the influence maximization problem in the LT model.

VI. Future work

There are many future directions related to this work. First, we need to go beyond the basic IC and LT models in the framework of [1], and study influence maximization problems in other models, especially those that may not satisfy the submodularity property. Second, we need to analyze real social networks using real-world social interaction data to learn that what type of influence models are suitable for these networks, in order to effectively apply influence maximization algorithms. Third, influence maximization may be combined with other network analysis techiques such as clustering and community detection to further enhance the efficiency and effectiveness of influence maximization. Finally, one may further pursue the theoretical problems related to influence maximization, for example, finding efficient approximation algorithms for computing influence in the IC or LT model, constructing LDAGs with approximation ratio guarantees, etc.

References

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

[2] ——, "Influential nodes in a diffusion model for social networks," in *Proceedings of the 32nd International Conference on Automata, Languages, and Programming*, 2005, pp. 1127–1138.

[3] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks," in *European Conference on Machine Learning and Knowledge Discovery in Databases*, 2006, pp. 259–271.

[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2007, pp. 420–429.

[5] R. Narayanam and Y. Narahari, "A shapley value based approach to discover influential nodes in social networks," *IEEE Transactions on Automation Science and Engineering*, 2010, to appear. Online version available at http://clweb.csa.iisc.ernet.in/nrsuri/social-networks-nrsuri-2010.pdf.

[6] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

[7] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large scale social networks," in *Proceedings of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010, full version appears as Microsoft Research Technical Report MSR-TR-2010-2.

[8] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks*, vol. 30, no. 1-7, pp. 107–117, 1998.

[9] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the 7th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2001, pp. 57–66.

[10] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2002, pp. 61–70.

[11] A. Anagnostopoulos, R. Kumar, and M. Mahdian, "Influence and correlation in social networks," in *Proceedings of the 14th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2008.

[12] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

[13] K. Saito, M. Kimura, K. Ohara, and H. Motoda, "Selecting information diffusion models over social networks for behavioral analysis," in *European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010, pp. 180–195.

[14] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Learning influence probabilities in social networks," in *Proceedings of the Third International Conference on Web Search and Web Data Mining*, 2010, pp. 241–250.

[15] G. Nemhauser, L. Wolsey, and M. Fisher, "An analysis of the approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, pp. 265–294, 1978.

[16] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[17] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," Microsoft Research, Tech. Rep. MSR-TR-2010-133, Oct. 2010.

[18] J. Leskovec, "Epinions social network," http://snap.stanford.edu/data/soc-Epinions1.html.

[19] ——, "Amazon product co-purchasing network, march 02 2003," http://snap.stanford.edu/data/amazon0302.html.

[20] L. Cowen, A. Brady, and P. Schmid, "DIGG: DynamIc Graph Generator," http://digg.cs.tufts.edu.