Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks*

Microsoft Research Technical Report
MSR-TR-2010-2
January 2010

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

Chi Wang
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
chiwang1@illinois.edu

Yajun Wang
Microsoft Research Asia
Beijing, China
yajunw@microsoft.com

Abstract

Influence maximization, defined by Kempe, Kleinberg, and Tardos (2003), is the problem of finding a small set of seed nodes in a social network that maximizes the spread of influence under certain influence cascade models. The scalability of influence maximization is a key factor for enabling prevalent viral marketing in large-scale online social networks. Prior solutions, such as the greedy algorithm of Kempe et al. (2003) and its improvements are slow and not scalable, while other heuristic algorithms do not provide consistently good performance on influence spreads. In this paper, we design a new heuristic algorithm that is easily scalable to millions of nodes and edges in our experiments. Our algorithm has a simple tunable parameter for users to control the balance between the running time and the influence spread of the algorithm. Our results from extensive simulations on several real-world and synthetic networks demonstrate that our algorithm is currently the best scalable solution to the influence maximization problem: (a) our algorithm scales beyond million-sized graphs where the greedy algorithm becomes infeasible, and (b) in all size ranges, our algorithm performs consistently well in influence spread — it is always among the best algorithms, and in most cases it significantly outperforms all other scalable heuristics to as much as 100%–260% increase in influence spread.

Keywords: influence maximization, social networks, viral marketing

*This is the second revision of the paper, done in Feb. 2010. The main change in this revision is to focus on the scalability of our new algorithm. We conduct new tests with real-world data up to millions of nodes and edges to show the strong scalability of our algorithm. Presentations are changed in various places to reflect this focus and to improve the overall readability.

1 Introduction

Word-of-mouth or viral marketing differentiates itself from other marketing strategies because it is based on trust among individuals' close social circle of families, friends, and coworkers. Research shows that people trust the information obtained from their close social circle far more than the information obtained from general advertisement channels such as TV, newspaper and online advertisements [15]. Thus many people believe that word-of-mouth marketing is the most effective marketing strategy (e.g. [14]).

The increasing popularity of many online social network sites, such as Facebook, Myspace, and Twitter, presents new opportunities for enabling large-scale and prevalent viral marketing online. Consider the following hypothetical scenario as a motivating example. A small company develops a cool online application and wants to market it through an online social network. It has a limited budget such that it can only select a small number of initial users in the network to use it (by giving them gifts or payments). The company wishes that these initial users would love the application and start influencing their friends on the social network to use it, and their friends would influence their friends' friends and so on, and thus through the word-of-mouth effect a large population in the social network would adopt the application. The problem is whom to select as the initial users so that they eventually influence the largest number of people in the network.

The above problem, called *influence maximization*, is first formulated as a discrete optimization problem by Kempe, Kleinberg, and Tardos as follows [9]: A social network is modeled as a graph with nodes representing individuals and edges representing connections or relationship between two individ-

uals. Influence are propagated in the network according to a stochastic cascade model, such as the following independent cascade (IC) model[1]: Each edge (u, v) in the graph is associated with a *propagation probability* $pp(u, v)$, which is the probability that node u independently activates (a.k.a. influences) node v at step $t + 1$ if u is activated at step t. Given a social network graph, the IC model, and a small number k, the influence maximization problem is to find k nodes in the graph (referred to as *seeds*) such that under the influence cascade model, the expected number of nodes activated by the k seeds (referred to as the *influence spread*) is the largest possible. Kempe et al. prove that the optimization problem is NP-hard, and present a greedy approximation algorithm guaranteeing that the influence spread is within $(1 - 1/e - \epsilon)$ of the optimal influence spread, where e is the base of natural logrithm, and ϵ depends on the accuracy of their Monte-Carlo estimate of the influence spread given a seed set.

However, their algorithm has a serious drawback — it is not scalable to large networks. A key element of their greedy algorithm is to compute the influence spread given a seed set, which turns out to be a difficult task (in fact, as we point out in Section 2 the computation is #P-hard). Instead of finding an exact algorithm, Monte-Carlo simulations of the influence cascade model are run for a large number of times in order to obtain an accurate estimate of the influence spread. Consequently, even with the recent optimizations [13, 3] that could achieves hundreds of times speedup, it still takes hours on a modern server to select 50 seeds in a moderate sized graph (15K nodes and 31K edges) while it becomes completely infeasible for larger graphs (e.g. more than 500K edges). Given that online social networks are typically of large-scale, we believe that the scalability issue of the greedy algorithm will be a fatal obstacle preventing it from supporting prevalent viral marketing activities in large-scale online social networks.

1.1 Our contribution

In this paper, we first show that computing influence spread in the independent cascade model is #P-hard, which closes an open question posed by Kempe et al. in [9]. It indicates that the greedy algorithm of [9] may have intrinsic difficulties to be made scalable for large graphs.

We then address the scalability issue by proposing a new heuristic algorithm that is several orders of magnitude faster than all existing greedy algorithms while matching the influence spread of the greedy algorithms. Our heuristic gains efficiency by restricting computations on the *local influence regions* of nodes. Moreover, by tuning the size of local influence regions, our heuristic is able to achieve tunable tradeoff between efficiency (in terms of running time) and effectiveness (in term of influence spread). Our heuristic can easily scale up to handle networks with millions of nodes and edges, and at this scale it beats all other existing heuristics of similar scalability in terms of the influence spread.

The main idea of our heuristic scheme is to use local arborescence[2] structures of each node to approximate the influence propagation. We first compute *maximum influence paths* (MIP) between every pair of nodes in the network via a Dijkstra shortest-path algorithm, and ignore MIPs with probability smaller than an influence threshold θ, effectively restricting influence to a local region. We then union the MIPs starting or ending at each node into the arborescence structures, which represent the local influence regions of each node. We only consider influence propagated through these local arborescences, and we refer to this model as the *maximum influence arborescence* (MIA) model.

We show that the influence spread in the MIA model is submodular (i.e. having a diminishing marginal return property), and thus the simple greedy algorithm that selects one node in each round with the maximum marginal influence spread can guarantee an influence spread within $(1 - 1/e)$ of the optimal solution in the MIA model, while any higher ratio approximation is NP-hard. The greedy algorithm on the MIA model is very efficient because (a) computation of the marginal influence spread on the arborescence structures can be done by efficient recursion; and (b) after selecting one seed with the largest influence spread, we only need to update local arborescence structures related to this seed for the selection of the next seed, and we further design a batch update scheme to speed up the update process.

We conduct extensive experiments on several real-world and synthetic networks of different scale and features, and under different types of the IC model. We compare our heuristic with both the greedy algorithm [9, 13, 3] and several existing heuristics including the degree discount heuristics of [3], the shortest-path based heuristics of [10], and the popular PageRank algorithm [2] for ranking web pages. Our simulation results show that: (a) the greedy algorithm of [9, 13, 3] and the shortest-path based heuristic [10] have poor scalability: they take hours or days to select 50 seeds when the graph size reaches a few hundred thousands and become infeasible for larger sized graphs, while in the same range MIA heuristic can finish in seconds (more than three orders of magnitude speedup), and it continues to scale up beyonds graphs with millions of edges, (b) comparing with the greedy algorithm and the shortest-path based heuristic in real graphs in which they are feasible to run, MIA heuristic has influence spread matches or is very close to those of the two other algorithms, (c) comparing with the rest heuristics, MIA algorithm is always among the best in influence spread, and in most cases it significantly outperforms the rest heuristics, with a margin as much as 100%–260% increase in influence spread. Moreover, we show that by tuning the threshold θ, we can adjust the tradeoff between efficiency and effec-

[1] Other models are also introduced in [9], but in this paper we focus on the independent cascade model.

[2] An arborescence is a tree in a directed graph where all edges are either pointing toward the root (in-arborescence) or pointing away form the root (out-arborescence).

tiveness at difference balance points on a spectrum.

To summarize, our main contribution is the design and evaluation of a scalable and tunable heuristic that handles the influence maximization problem for large-scale social networks. We demonstrate that our heuristic is currently the best one that could handle large-scale networks with more than a million edges, while even for moderate sized networks it is a very competitive alternative to much slower algorithms. The balanced efficiency and effectiveness of our heuristic make it suitable as a generic solution to influence maximization for many large-scale online social networks encountered in practice.

1.2 Related work

Domingos and Richardson [5, 17] are the first to study influence maximization as an algorithmic problem. Their methods are probabilistic, however. Kempe, Kleinberg, and Tardos [9] are the first to formulate the problem as a discrete optimization problem. Besides what we mentioned above already, they also study a number of other topics such as generalizations of influence cascade models and mixed marketing strategies in influence maximization. As pointed out, the main drawback of their work is the scalability of their greedy algorithms.

Several recent studies aimed at addressing this issue. In [13], Leskovec et al. present a "lazy-forward" optimization in selecting new seeds, which greatly reduces the number of evaluations on the influence spread of nodes and results in as much as 700 times speedup demonstrated by their experimental results. However, even though the "lazy-forward" optimization is significant, it still takes hours to find 50 most influential nodes in a network with a few tens of thousands of nodes, as shown in [3].

In [10], Kimura and Saito propose shortest-path based influence cascade models and provide efficient algorithms to compute influence spread under these models. The key differences between their work and ours are (a) instead of using maximum influence paths, they use simple shortest paths on the graph, which are not related to propagation probabilities, and (b) they do not utilize local structures such as our arborescences and thus in every round they need global computations to select the next seed. Therefore, their algorithms are not as efficient as ours.

This paper is the continuation of [3] in the pursuit of efficient and scalable influence maximization algorithms. In [3], we explore two directions in improving the efficiency: one is to further improve the greedy algorithm of [9], and the other is to design new heuristic algorithms. The first direction shows improvement but is not significant enough, indicating that this direction could be difficult to continue. The second direction leads to new degree discount heuristics that are very efficient and generate reasonably good influence spread. The major issue is that the degree discount heuristics are derived from the *uniform* IC model where propagation probabilities on all edges are the same, which is rarely the case in reality. Our current work is a major step in overcoming this limitation — our new heuristic algorithm works for the general IC model while still maintain good balance between efficiency and effectiveness. We conduct much more experiments than in [3] on more and larger scale graphs, and our results show that the MIA heuristic performs consistently better than the degree discount heuristic in all graphs.

Paper organization. Section 2 provides preliminaries on the IC model and the greedy algorithm, and also points out that computing the exact influence spread given a seed set is #P-hard. Section 3 presents our MIA model and the algorithm for this model as well as its extension, the PMIA model. Section 4 shows our experimental results. We discuss future directions in Section 5. Additional experimental results are presented in the appendix.

2 IC model and greedy algorithm

We consider a directed graph $G = (V, E)$ with edge labels $pp : E \to [0, 1]$. For every edge $(u, v) \in E$, $pp(u, v)$ denotes the propagation probability of the edge, which is the probability that v is activated by u through the edge in the next step after u is activated.

Given a seed set $S \subseteq V$, the independent cascade (IC) model works as follows. Let $S_t \subseteq V$ be the set of nodes that are activated at step $t \geq 0$, with $S_0 = S$. At step $t + 1$, every node $u \in S_t$ may activate its out-neighbors $v \in V \setminus \cup_{0 \leq i \leq t} S_i$ with an independent probability of $pp(u, v)$. The process ends at a step t with $S_t = \emptyset$. Note that each activated node only has one chance to activate its out-neighbors at the step right after itself is activated, and each node stays as an activated node after it is activated. The *influence spread* of S, which is the expected number of activated nodes given seed set S, is denoted as $\sigma_I(S)$.

Given an input k, the influence maximization problem in the IC model is to find a subset $S^* \subseteq V$ such that $|S^*| = k$ and $\sigma_I(S^*) = \max\{\sigma_I(S) \mid |S| = k, S \subseteq V\}$. It is shown in [9] that this problem is NP-hard, but a constant-ratio approximation algorithm is available.

We say that a non-negative real valued function f on subsets of V is *submodular* if $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$, for all $v \in V$ and all pairs of subsets S and T with $S \subseteq T \subseteq V$. Intuitively, this means that f has diminishing marginal return. Moreover, we say that f is *monotone* if $f(S) \leq f(T)$ for all $S \subseteq T$. For any submodular and monotone function f with $f(\emptyset) = 0$, the problem of finding a set S of size k that maximizes $f(S)$ can be approximated by a simple greedy algorithm shown as Algorithm 1. The algorithm iteratively selects new seed u that maximizes the incremental change of f into the seed set S until k seeds are selected. It is shown in [16] that the algorithm guarantees the 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.

Algorithm 1 Greedy(k, f)

1: initialize $S = \emptyset$
2: **for** $i = 1$ to k **do**
3: select $u = \arg\max_{w \in V \setminus S}(f(S \cup \{w\}) - f(S))$
4: $S = S \cup \{u\}$
5: **end for**
6: output S

In [9], it is shown that function $\sigma_I(\cdot)$ is submodular and monotone with $\sigma_I(\emptyset) = 0$. Therefore, algorithm Greedy(k, σ_I) solves the influence maximization problem with an approximation ratio of $1 - 1/e$.

One important issue, however, is that there is no efficient way to compute $\sigma_I(S)$ given a set S. Although Kempe et al. claim that finding an efficient algorithm for computing $\sigma_I(S)$ is open [9], we point out that the computation is actually #P-hard, by showing a reduction from the counting problem of s-t connectness in a graph.

Theorem 1 *Computing the influence spread $\sigma_I(S)$ given a seed set S is #P-hard.*

Proof. We prove the theorem by a reduction from the counting problem of s-t connectness in a directed graph [20]. An instance of s-t connectness is a directed graph $G = (V, E)$ and two vertices s and t in the graph. The problem is to count the number of subgraphs of G in which s is connected to t. It is straightforward to see that this problem is equivalent to computing the probability that s is connected to t when each edge in G has an independent probability of $1/2$ to be connected, and another $1/2$ to be disconnected. We reduce this problem to the influence spread computation problem as follows. Let $\sigma_I(S, G)$ denote the influence spread in G given a seed set S. First, let $S = \{s\}$, and let $pp(e) = 1/2$ for all $e \in E$, and compute $I_1 = \sigma_I(S, G)$. Next, we add a new node t' and a directed edge from t to t' to G, obtaining a new graph G', and let $pp(t, t') = 1$. Then we compute influence spread $I_2 = \sigma_I(S, G')$. Let $p(S, v, G)$ denote the probability that v is influenced by seed set S in G. It is easy to see that $I_2 = \sigma_I(S, G) + p(S, t, G) \cdot pp(t, t')$. Therefore, $I_2 - I_1$ is the probability that s is connected to t, and thus we solve the s-t connectness counting problem. It is shown in [20] that s-t connectness is #P-complete, and thus the influence spread computation problem is #P-hard. □

The above theorem shows that computing exact influence spread is hard. Moreover, finding an efficient approximation algorithm for computing the probability of s-t connectivity is a long-standing open problem [21]. Together with the fact that several improvements ([13, 3]) of the original greedy algorithm of [9] are still not efficient, we believe that we need to look for alternative ways, such as heuristic algorithms, to tackle the efficiency problem in influence maximization.

3 MIA model and its algorithm

3.1 Basic MIA model and greedy algorithm

For a path $P = \langle u = p_1, p_2, \ldots, p_m = v \rangle$, we define the *propagation probability* of the path, $pp(P)$, as

$$pp(P) = \Pi_{i=1}^{m-1} pp(p_i, p_{i+1}).$$

Intuitively the probability that u activates v through path P is $pp(P)$, because it needs to activate all nodes along the path. To approximate the actual expected influence within the social network, we propose to use the *maximum influence path* (*MIP*) to estimate the influence from one node to another. Let $\mathcal{P}(G, u, v)$ denote the set of all paths from u to v in a graph G.

Definition 1 (Maximum Influence Path) *For a graph G, we define the* maximum influence path *$MIP_G(u, v)$ from u to v in G as*

$$MIP_G(u, v) = \arg\max_P \{pp(P) \mid P \in \mathcal{P}(G, u, v)\}.$$

Ties are broken in a predetermined and consistent way, such that $MIP_G(u, v)$ is always unique, and any subpath in $MIP_G(u, v)$ from x to y is also the $MIP_G(x, y)$. If $\mathcal{P}(G, u, v) = \emptyset$, we denote $MIP_G(u, v) = \emptyset$.

Note that for each edge (u, v) in the graph, if we translate the propagation probability $pp(u, v)$ to a distance weight $-\log pp(u, v)$ on the edge, then $MIP_G(u, v)$ is simply the shortest path from u to v in the weighted graph G. Therefore, the maximum influence paths and the later maximum influence arborescences directly correspond to shortest paths and shortest-path arborescences, and thus they permit efficient algorithms such as Dijkstra algorithm to compute them.

For a given node v in the graph, we propose to use the *maximum influence in-arborescence* (*MIIA*), which is the union of the maximum influence paths to v,[3] to estimate the influence to v from other nodes in the network. We use an *influence threshold θ* to eliminate MIPs that have too small propagation probabilities. Symmetrically, we also define *maximum influence out-arborescence* (*MIOA*) to estimate the influence of v to other nodes.

Definition 2 (MAXIMUM INFLUENCE IN(OUT)-ARBORESCENCE) *For an influence threshold θ, the* maximum influence in-arborescence *of a node $v \in V$, $MIIA(v, \theta)$, is*

$$MIIA(v, \theta) = \cup_{u \in V, pp(MIP_G(u,v)) \geq \theta} MIP_G(u, v).$$

The maximum influence out-arborescence *$MIOA(v, \theta)$ is:*

$$MIOA(v, \theta) = \cup_{u \in V, pp(MIP_G(v,u)) \geq \theta} MIP_G(v, u).$$

[3] Since we break ties in maximum influence paths consistently, the union of maximum influence paths to a node do not have undirected cycles, and thus it is indeed an arborescence.

Algorithm 2 $ap(u, S, MIIA(v, \theta))$

1: **if** $u \in S$ **then**
2: $\quad ap(u) = 1$
3: **else if** $N^{in}(u) = \emptyset$ **then**
4: $\quad ap(u) = 0$
5: **else**
6: $\quad ap(u) = 1 - \Pi_{w \in N^{in}(u)}(1 - ap(w) \cdot pp(w, u))$
7: **end if**

Intuitively, $MIIA(v, \theta)$ and $MIOA(v, \theta)$ give the local influence regions of v, and different values of θ controls the size of these local influence regions.

Given a set of seeds S in G and the in-arborescence $MIIA(v, \theta)$ for some $v \notin S$, we approximate the IC model by assuming that the influence from S to v is only propagated through edges in $MIIA(v, \theta)$. With this approximation, we can calculate the probability that v is activated given S exactly. Let the *activation probability* of any node u in $MIIA(v, \theta)$, denoted as $ap(u, S, MIIA(v, \theta))$, be the probability that u is activated when the seed set is S and influence is propagated in $MIIA(v, \theta)$. Let $N^{in}(u, MIIA(v, \theta))$ be the set of in-neighbors of u in $MIIA(v, \theta)$. In the above notations, $MIIA(v, \theta)$ and S may be dropped when it is clear from the context. Then $ap(u, S, MIIA(v, \theta))$ can be computed recursively as given in Algorithm 2.

Note that because $MIIA(v, \theta)$ is an in-arborescence, there are no multiple paths between any pair of nodes in $MIIA(v, \theta)$, and thus there is no dependency issue in the calculation of the activation probability and the calculation in Algorithm 2 exactly matches the IC model restricted onto $MIIA(v, \theta)$.

In our MIA model we assume that seeds in S influence every individual node v in G through its $MIIA(v, \theta)$. Let $\sigma_M(S)$ denote the influence spread of S in our MIA model, then we have

$$\sigma_M(S) = \sum_{v \in V} ap(v, S, MIIA(v, \theta)). \tag{3.1}$$

Even though activating multiple nodes from the same set of seeds in the MIA model are correlated events, Equation (3.1) is still correct due to the linearity of the expectation over the sum of random variables.

We are interested in finding a set of seeds S of size k such that $\sigma_M(S)$ is maximized. It is not surprising that this optimization problem is NP-hard. In fact, the same reduction from set cover problem in [9] together with Theorem 5.3 of [6] is sufficient to show the following.

Theorem 2 *It is NP-hard to compute a set of nodes S of size k such that $\sigma_M(S)$ is maximized. Furthermore, it is NP-hard to approximate within a factor of $1 - 1/e + \epsilon$ for any $\epsilon > 0$.*

It is straight forward to verify the following result, which means we have an approximation algorithm.

Theorem 3 *Function σ_M is submodular and monotone and $\sigma_M(\emptyset) = 0$. Therefore, $\mathsf{Greedy}(k, \sigma_M)$ of Algorithm 1 achieves $1 - 1/e$ approximation ratio for the influence maximization problem in the basic MIA model.*

Note that the recursive computation of $ap(u)$ in Algorithm 2 can be transformed into an iterative form such that all $ap(u)$'s with u in $MIIA(v, \theta)$ can be computed by one traverse of the arborescence $MIIA(v, \theta)$ from leaves to the root. Thus, computing $\sigma_M(S)$ using Equation (3.1) and Algorithm 2 is polynomial-time. Together with Algorithm 1, we already have a polynomial-time approximation algorithm. However, we could further improve the efficiency of the algorithm, as we shown in the next section.

3.2 More efficient greedy algorithm

The only important step in the greedy algorithm is to select the next seed that gives the largest incremental influence spread. Consider the maximum influence in-arborescence $MIIA(v, \theta)$ of size t and a given seed set S. To select the next seed u, we need to compute the activation probability $ap(v, S \cup \{w\}, MIIA(v, \theta))$ for every $w \in MIIA(v, \theta)$, which takes $O(t^2)$ time if we simply use Algorithm 2 to compute every $ap(v, S \cup \{w\}, MIIA(v, \theta))$. We now show a batch update scheme such that we could compute $ap(v, S \cup \{w\}, MIIA(v, \theta))$'s for all $w \in MIIA(v, \theta)$ in $O(t)$ time.

To do so, we utilize the linear relationship between $ap(u)$ and $ap(v)$ in $MIIA(v, \theta)$, as shown by the following lemma, which is not difficult to derive from line 6 of Algorithm 2.

Lemma 1 (Influence Linearity) *Consider $MIIA(v, \theta)$ and a node u in it. If we treat the activation probabilities $ap(u)$ and $ap(v)$ as variables and other $ap(w)$'s as constants, where w is any node in $MIIA(v, \theta)$ other than u and v, then $ap(v) = \alpha(v, u) \cdot ap(u) + \beta(v, u)$, where $\alpha(v, u), \beta(v, u)$ are constants independent of $ap(u)$.*

Based on the recursive computation of $ap(u, S, MIIA(v, \theta))$ as shown in line 6 of Algorithm 2, it is straightforward to derive a recursive computation of $\alpha(v, u)$, as shown in Algorithm 3. Note that Algorithm 3 can be transformed into an iterative form such that all $\alpha(v, u)$'s can be computed by one traverse of $MIIA(v, \theta)$ from the root to the leaves.

Computing the linear coefficients $\alpha(v, u)$ as defined in Lemma 1 is crucial in computing the incremental influence spread of a node u. Let us consider again the maximum influence in-arborescence $MIIA(v, \theta)$ of size t and a given seed set S. For any $w \in MIIA(v, \theta)$, if we select w as the next seed, its $ap(w)$ increases from the current value to 1. Since $ap(w)$ and $ap(v)$ has a linear relationship with the linear coefficient $\alpha(v, w)$, the incremental influence of w on v is given by $\alpha(v, w) \cdot (1 - ap(w))$. Therefore, we only need one pass of $MIIA(v, \theta)$ to compute $ap(w)$'s for all $w \in MIIA(v, \theta)$, and a second pass of $MIIA(v, \theta)$ to compute $\alpha(v, w)$'s and

Algorithm 3 Compute $\alpha(v,u)$ with $MIIA(v,\theta)$ and S, after $ap(u,S,MIIA(v,\theta))$ for all u in $MIIA(v,\theta)$ are known.

/* the following is computed recursively */
if $u = v$ **then**
  $\alpha(v,u) = 1$
else
  set w to be the out-neighbor of u
  if $w \in S$ **then**
    $\alpha(v,u) = 0$ /* u's influence to v is blocked by seed w */
  else
    $\alpha(v,u) = \alpha(v,w) \cdot pp(u,w) \cdot \Pi_{u' \in N^{in}(w) \setminus \{u\}}(1 - ap(u') \cdot pp(u',w))$
  end if
end if

$\alpha(v,w) \cdot (1 - ap(w))$'s for all $w \in MIIA(v,\theta)$. This reduces the running time of computing incremental influence spread of all nodes in $MIIA(v,\theta)$ from $O(t^2)$ to $O(t)$.

Our complete greedy algorithm for the basic MIA model is presented in Algorithm 4. Lines (2–11) evaluate the incremental influence spread $IncInf(u)$ for any node u when the current seed set is empty. The evaluation is exactly as we described above using the linear coefficients $\alpha(v,u)$.

Lines (15–30) update the incremental influences whenever a new seed is selected in line 14. Suppose u is selected as the new seed in an iteration. The influence of u in the MIA model only reaches nodes in $MIOA(u,\theta)$. Thus the incremental influence spread $IncInf(w)$ for some w needs to be updated if and only if w is in $MIIA(v,\theta)$ for some $v \in MIOA(u,\theta)$. This means that the update process is relatively local to u. The update is done by first subtracting $\alpha(v,w) \cdot (1 - ap(w,S,MIIA(v,\theta)))$ before adding u into the seed set (line 19), and then adding u into the seed set (line 22), recomputing the $ap(w,S,MIIA(v,\theta))$ and $\alpha(v,w)$ under the new seed set (lines 24–25), and adding $\alpha(v,w) \cdot (1 - ap(w,S,MIIA(v,\theta)))$ into $IncInf(w)$ (line 28).

Time and space complexity. Let $n_{i\theta} = \max_{v \in V}\{|MIIA(v,\theta)|\}$ and $n_{o\theta} = \max_{v \in V}\{|MIOA(v,\theta)|\}$. Computing $MIIA(v,\theta)$ can be done using efficient implementations of Dijkstra's shortest-path algorithm. Assume the maximum running time to compute $MIIA(v,\theta)$ for any $v \in V$ is $t_{i\theta}$. When $MIIA(v,\theta)$'s for all node $v \in V$ are available, $MIOA(v,\theta)$'s can be derived from $MIIA(v,\theta)$'s, therefore no extra running time for $MIOA(v,\theta)$'s is needed. Notice that $n_{i\theta} = O(t_{i\theta})$.

For every node $v \in V$, our algorithm stores $MIIA(v,\theta)$, $MIOA(v,\theta)$, and for every $u \in MIIA(v,\theta)$, $ap(u,S,MIIA(v,\theta))$ and $\alpha(v,u)$ are stored (note that $ap(u,S,MIIA(v,\theta))$ can reuse the same entry for different seed set S). We also use a max-heap to store and update $IncInf(v)$ for all $v \in V$. Therefore, the space complexity of the algorithm is $O(n(n_{i\theta} + n_{o\theta}))$.

During the initialization of Algorithm 4, it takes $O(nt_{i\theta})$ time to compute $MIIA(v,\theta)$ for all $v \in V$, $O(nn_{i\theta})$ time to compute all $\alpha(v,u)$'s and $IncInf(u)$'s, and $O(n)$ time to initialize the max-heap for storing $IncInf(u)$'s. Therefore, the total running time for initialization is $O(nt_{i\theta})$. During one iteration of the main loop, it takes constant time to select the new seed from the max-heap, $O(n_{o\theta}n_{i\theta}\log n)$ time to update $IncInf(w)$'s on the max-heap, and $O(n_{o\theta}n_{i\theta})$ time to compute $ap(w,S,MIIA(v,\theta,S))$'s and $\alpha(v,w)$'s after selecting the new seed. Thus, one iteration of the main loop takes $O(n_{o\theta}n_{i\theta}\log n)$ time. Together, the total running time of the algorithm is $O(nt_{i\theta} + kn_{o\theta}n_{i\theta}\log n))$. Note that without applying the improvement of utilizing the linear relationship, the time complexity would be $O(nt_{i\theta} + kn_{o\theta}n_{i\theta}(n_{i\theta} + \log n))$.

Therefore, the algorithm performs the best when $n_{i\theta}$, $n_{o\theta}$, and $t_{i\theta}$ are significantly smaller than n, that is, when the ar-

Algorithm 4 MIA(G,k,θ)

1: /* initialization */
2: set $S = \emptyset$
3: set $IncInf(v) = 0$ for each node $v \in V$
4: **for** each node $v \in V$ **do**
5: compute $MIIA(v,\theta)$ and $MIOA(v,\theta)$
6: set $ap(u,S,MIIA(v,\theta)) = 0, \forall u \in MIIA(v,\theta)$ /* since $S = \emptyset$ */
7: compute $\alpha(v,u), \forall u \in MIIA(v,\theta)$ (Algorithm 3)
8: **for** each node $u \in MIIA(v,\theta)$ **do**
9: $IncInf(u) += \alpha(v,u) \cdot (1 - ap(u,S,MIIA(v,\theta)))$
10: **end for**
11: **end for**
12: /* main loop */
13: **for** $i = 1$ to k **do**
14: pick $u = \arg\max_{v \in V \setminus S}\{IncInf(v)\}$
15: /* update incremental influence spreads*/
16: **for** $v \in MIOA(u,\theta) \setminus S$ **do**
17: /* subtract previous incremental influence */
18: **for** $w \in MIIA(v,\theta) \setminus S$ **do**
19: $IncInf(w) -= \alpha(v,w) \cdot (1 - ap(w,S,MIIA(v,\theta)))$
20: **end for**
21: **end for**
22: $S = S \cup \{u\}$
23: **for** $v \in MIOA(u,\theta) \setminus S$ **do**
24: compute $ap(w,S,MIIA(v,\theta)), \forall w \in MIIA(v,\theta)$ (Algo. 2)
25: compute $\alpha(v,w), \forall w \in MIIA(v,\theta)$ (Algo. 3)
26: /* add new incremental influence */
27: **for** $w \in MIIA(v,\theta) \setminus S$ **do**
28: $IncInf(w) += \alpha(v,w) \cdot (1 - ap(w,S,MIIA(v,\theta)))$
29: **end for**
30: **end for**
31: **end for**
32: **return** S

borescences are small. This typically occurs for a reasonable range of θ values, when the graph is sparse and the propagation probabilities on edges are usually small, which is the case for social networks. Our experiments in the Section 4 will demonstrate the efficiency of our algorithm.

3.3 Prefix excluding MIA model

In the basic MIA model, we only consider the maximum influence path from u to v for influence propagation. Consider the scenario of two seeds s_1 and s_2 such that $MIP_G(s_2, v) \subset MIP_G(s_1, v)$. The probability that v is activated in the basic MIA model is only determined by s_2 and is not affected by s_1, or we can say that the influence of s_1 to v is blocked by s_2 in the middle.

To achieve a better approximation to the IC model, we prefer a MIA model in which the influence of a seed is not blocked by other seeds. A natural way to extend the basic MIA model is considering maximum influence paths avoiding other seeds. Let $S = \{s_1, s_2, \ldots, s_m\}$ and $S^i = S \setminus \{s_i\}$. We define $G(S^i)$ be the subgraph of G induced by $V \setminus S^i$. Then, for each seed s_i and node $v \in V \setminus S$, we use the maximum influence path $MIP_{G(S^i)}(s_i, v)$ to estimate the influence from s_i to v. In other words, we consider maximum influence paths avoiding other seeds in calculating the influence spread.

The generic Algorithm 1 also works in this model. However, it is not clear how to implement it efficiently similar to the approach in Algorithm 4. In this section, we consider a variant of the above extension that allows an efficient greedy algorithm. We call this extension the *prefix excluding MIA* (PMIA) model.

Intuitively, in the PMIA model, the seeds have an order (as the order by which they are selected by the greedy algorithm). For any given seed s, its maximum influence paths to other nodes should avoid all seeds in the prefix before s. The major technical difference is the definition of the *maximum influence in(out)-arborescence* for the PMIA model, especially if we want to design an efficient greedy algorithm in the framework of Algorithm 4.

Let $S = \langle s_1, s_2, \ldots, s_m \rangle$ be a sequence of seeds. Define $S_i = \langle s_1, s_2, \ldots, s_{i-1} \rangle$ and $S_1 = \emptyset$. Let $G(S')$ be the subgraph of G induced by $V \setminus S'$ for any sequence S'. We first define *ineffective seeds* with respect to a node v, which are those seeds whose influence to v are blocked by some other subsequent seeds in sequence S.

Definition 3 (Ineffective seeds) *For a given node $v \in V \setminus S$, we define the set of* ineffective seeds *for v as:*

$$IS(v, S) = \{s_i \in S \mid \exists j > i, s.t., s_j \in MIP_{G(S_i)}(s_i, v)\}.$$

Now consider the maximum influence in-arborescence (MIIA) of a node v in the PMIA model. First, for the maximum influence path from a seed s_i to v, it should be defined as $MIP_{G(S_i)}(s_i, v)$ to avoid seeds in its prefix. Second, for the case where the MIP from seed s_i to v is blocked by a subsequent seed s_j, we need to give a special treatment in order to use the influence linearity of Lemma 1 for an efficient computation of incremental influence spread. Consider a node $u \notin S$ located on the MIP from s_i to s_j. If u is selected as a seed later, then its MIP to v should avoid all seeds in S, and thus to compute its incremental influence spread correctly using the linearity property, we need to compute the MIP from u to v in the graph $G(S)$. Moreover, we need to remove the ineffective seed s_i and its MIP to v because otherwise s_i would have two different paths to v, violating the arborescence definition.

For out-arborescence from $v \notin S$, we need to consider all MIPs from v that avoid all seeds in S. This is because we only need to compute the out-arborescence of a node v when v is just selected as the next seed. In this case, the paths in the above computed out-arborescence of v match the paths in the corresponding in-arborescences used to compute the incremental influence of v (since those paths avoid all seeds already in S). Therefore, we have the following formal definitions.

Definition 4 (MIIA(MIOA) for the PMIA Model) *The* maximum influence in-arborescence *of v in the PMIA model for $v \notin S$ is:*

$$\begin{aligned} &PMIIA(v, \theta, S) = \\ &\quad (\cup\{MIP_{G(S_i)}(s_i, v) \mid s_i \in S \setminus IS(v, S), \\ &\qquad pp(MIP_{G(S_i)}(s_i, v)) \geq \theta\}) \\ &\quad \cup(\cup\{MIP_{G(S)}(u, v) \mid u \in V \setminus S, \\ &\qquad pp(MIP_{G(S)}(u, v)) \geq \theta\}). \end{aligned}$$

The maximum influence out-arborescence *of v in the PMIA model for $v \notin S$ is:*

$$\begin{aligned} PMIOA(v, \theta, S) \quad = \quad & \cup\{MIP_{G(S)}(v, u) \mid u \in V \setminus S, \\ & pp(MIP_{G(S)}(v, u)) \geq \theta\}. \end{aligned}$$

Given the above definition, we can have activation probabilities $ap(u, S, PMIIA(v, \theta, S))$ computed by Algorithm 2. Then, similar to Equation (3.1), we can define $\sigma_P(S)$ as the influence spread given a seed sequence S, which is computed using the following equation:

$$\sigma_P(S) = \sum_{v \in V} ap(v, S, PMIIA(v, \theta, S)). \tag{3.2}$$

Notice that different sequences S of the same set of seeds may generate different values of $\sigma_P(S)$. Therefore, the submodularity defined on set functions previous does not apply to σ_P. Fortunately, we can define *sequence submodularity* in a similar way, which also leads to the greedy algorithm with an approximation ratio of $1 - 1/e$.

Sequence submodularity. We now define sequence submodularity, which is implicitly used by Streeter and Golovin in [18]. Let $\mathcal{S}$ be the set of all sequences of V, including the empty sequence $\emptyset$. Let $\oplus$ be the binary operator that concatenates two

sequences into one. We say that a non-negative function f defined on $\mathcal{S}$ is *sequence submodular* if $f(S_1 \oplus S_2 \oplus \{t\}) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \{t\}) - f(S_1)$ for all sequences $S_1, S_2 \in \mathcal{S}$. Moreover, f is *prefix monotone* if $f(S_1) \leq f(S_2 \oplus S_1)$ for all $S_1, S_2 \in \mathcal{S}$. An important result that matches the one for set submodular functions is that if f is sequence submodular and prefix monotone and $f(\emptyset) = 0$, then the greedy algorithm of Algorithm 1 (with set union $\cup$ replaced by sequence concatenation $\oplus$) finds a sequence S within $1 - 1/e$ of the optimal S^*. Since the original proof in [18] is presented in a different context, we rephrase the proof below.

Theorem 4 (Theorem 3 in [18]) *Let f be a sequence submodular, prefix monotone function with $f(\emptyset) = 0$. Define $S_0 = \emptyset$ and for $1 \leq i \leq k$, let $s_i = \arg\max_{s \in V}\{f(S_{i-1} \oplus \{s\}\}$ and $S_i = S_{i-1} \oplus \{s_i\}$. Let $S^* = \arg\max_{S'}\{f(S') \mid S' \in \mathcal{S}$ and $|S'| = k\}$. We have*

$$f(S_k) \geq (1 - 1/e) \cdot f(S^*).$$

Proof. Let $\Delta_i = f(S^*) - f(S_i)$. By prefix monotonicity, we have $f(S^*) \leq f(S_i \oplus S^*)$. Let $S^* = \langle s_1^*, \ldots, s_k^* \rangle$, and $S_i^* = \langle s_1^*, \ldots, s_i^* \rangle$. By submodularity, for $1 \leq i \leq k$, we have

$$\begin{aligned} f(S_i \oplus S^*) &= f(S_i \oplus S_{k-1}^* \oplus \langle s_k^* \rangle) \\ &\leq f(S_i \oplus S_{k-1}^*) + f(S_i \oplus \langle s_k^* \rangle) - f(S_i) \\ &\leq f(S_i \oplus S_{k-1}^*) + f(S_{i+1}) - f(S_i), \end{aligned}$$

where the last inequality is due to the definition of S_{i+1}. Repeating the above derivation for k times, we have

$$\begin{aligned} f(S^*) \leq f(S_i \oplus S^*) &\leq f(S_i) + k \cdot (f(S_{i+1}) - f(S_i)) \\ &= f(S_i) + k \cdot (\Delta_i - \Delta_{i+1}). \end{aligned}$$

Therefore, $\Delta_i \leq k \cdot (\Delta_i - \Delta_{i+1})$ and $\Delta_{i+1} \leq (1 - \frac{1}{k})\Delta_i$. Hence

$$f(S^*) - f(S_k) = \Delta_k \leq (1 - \frac{1}{k})^k \Delta_0 \leq f(S^*)/e.$$

□

It is not difficult to verify the following result on σ_P, which means that the greedy algorithm works as an approximation algorithm.

Theorem 5 *Function σ_P is sequence submodular and prefix monotone and $\sigma_P(\emptyset) = 0$. Therefore,* Greedy(k, σ_P) *of Algorithm 1 (with set union $\cup$ replaced by sequence concatenation $\oplus$) achieves $1 - 1/e$ approximation ratio for the influence maximization problem in the PMIA model.*

Algorithm in the PMIA model. We now present the necessary changes needed to adapt Algorithm 4 to the PMIA model. The major issue is the computation of $PMIIA(v, \theta, S)$ and $PMIOA(v, \theta, S)$. The computation of $PMIOA(v, \theta, S)$ is relatively simple, since we only need to remove S from the graph. Therefore, we can use the Dijkstra algorithm on graph $G(S)$ to compute $PMIOA(v, \theta, S)$.

To efficiently compute $PMIIA(v, \theta, S)$, we maintain the set of ineffective seeds $IS(v, S)$ for each node $v \in V \setminus S$. Given $IS(v, S)$, $PMIIA(v, \theta, S)$ can be calculated as follows. We start a Dijkstra algorithm from v traversing inward edges. Whenever the Dijkstra algorithm hits a seed node s, it stops this branch and does not go further on the in-neighbors of s. After the Dijkstra algorithm completes, we remove all nodes $IS(v, S)$ from the computed in-arborescence.

When a new seed u is selected, we have to update $IS(v, S)$ for all nodes v in $PMIOA(u, \theta, S)$. This can be done by checking the set of *effective seeds* (those in $S \setminus IS(v, S)$) that are blocked by u in $PMIIA(v, \theta, S)$. For completeness, we present Algorithm 5 for the efficient greedy algorithm in the PMIA model. Algorithm 5 essentially follows Algorithm 4, with all $MIIA$'s and $MIOA$'s being replaced by $PMIIA$'s and $PMIOA$'s, and these $PMIIA$'s and $PMIOA$'s being recomputed whenever the seed set changes (lines 16 and 26).

4 Experiment

We conduct experiments on our algorithm as well as a number of other algorithms on several real-world and synthetic networks. Our experiments aim at illustrating the performance of our algorithm from the following aspects: (a) its scalability comparing to other algorithms; (b) its influence spread comparing to other algorithms; and (c) the tuning of its control parameter θ.

4.1 Experiment setup

Datasets. We use four real-world networks and a synthetic dataset. The first one, denoted NetHEPT, is the same as used in [3]. It is 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. The second is a much larger collaboration network, the DBLP Computer Science Bibliography Database maintained by Michael Ley. The other two datasets are published network data by Jure Leskovec. One is a Who-trust-whom network of Epinions.com [12], 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). Another is the Amazon product co-purchasing network [11] 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).[4] We refer to these two datasets as Epinions and Amazon. We choose these networks since it covers a variety of networks with sizes

[4] Although the Amazon dataset is for products, we still include it in our experiments to test a variant of a network. Moreover, it also makes sense to find top seed products that lead to the most co-purchasing behaviors.

Algorithm 5 PMIA(G, k, θ)

1: /* initialization */
2: set $S = \emptyset$
3: set $IncInf(v) = 0$ for each node $v \in V$
4: **for** each node $v \in V$ **do**
5: compute $PMIIA(v, \theta, S)$
6: set $ap(u, S, PMIIA(v, \theta, S)) = 0, \forall u \in PMIIA(v, \theta, S)$ /* since $S = \emptyset$ */
7: compute $\alpha(v, u), \forall u \in PMIIA(v, \theta, S)$ (Algorithm 3)
8: **for** each node $u \in PMIIA(v, \theta, S)$ **do**
9: $IncInf(u) += \alpha(v, u) \cdot (1 - ap(u, S, PMIIA(v, \theta, S)))$
10: **end for**
11: **end for**
12: /* main loop */
13: **for** $i = 1$ to k **do**
14: pick $u = \arg\max_{v \in V \setminus S}\{IncInf(v)\}$
15: /* update incremental influence spreads*/
16: compute $PMIOA(u, \theta, S)$
17: **for** $v \in PMIOA(u, \theta, S)$ **do**
18: /* subtract previous incremental influence */
19: **for** $w \in PMIIA(v, \theta, S) \setminus S$ **do**
20: $IncInf(w) -= \alpha(v, w) \cdot (1 - ap(w, S, PMIIA(v, \theta, S)))$
21: **end for**
22: **end for**
23: $S = S \cup \{u\}$
24: /* the following $PMIOA(u, \theta, S \setminus \{u\})$ is the same as computed in line 16 */
25: **for** $v \in PMIOA(u, \theta, S \setminus \{u\}) \setminus \{u\}$ **do**
26: compute $PMIIA(v, \theta, S)$
27: compute $ap(w, S, PMIIA(v, \theta, S)), \forall w \in PMIIA(v, \theta, S)$ (Algo. 2)
28: compute $\alpha(v, w), \forall w \in PMIIA(v, \theta, S)$ (Algo. 3)
29: /* add new incremental influence */
30: **for** $w \in PMIIA(v, \theta, S) \setminus S$ **do**
31: $IncInf(w) += \alpha(v, w) \cdot (1 - ap(w, S, PMIIA(v, \theta, S)))$
32: **end for**
33: **end for**
34: **end for**
35: **return** S

ranging from $30K$ edges to $2M$ edges. Some basic statistics about these networks are given in Table 1 (Epinions and Amazon networks are treated as undirected graphs in the statistics). Finally, in the scalability test, we use the DIGG package available on the web [4] to randomly generate power-law graphs of difference sizes based on the model of [1].

Generating propagation probabilities. Since our algorithm is targeted at the general IC model with nonuniform propagation probabilities, we use the following two models to generate these nonuniform probabilities.

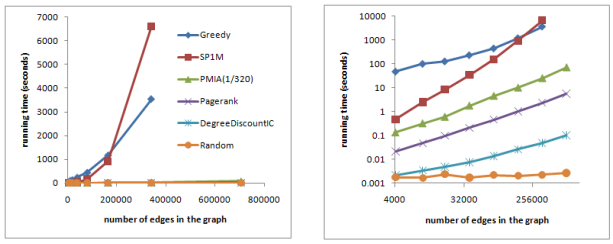
Table 1: Statistics of four tested real-world networks.

Dataset	NetHEPT	DBLP	Epinions	Amazon
#Node	15K	655K	76K	262K
#Edge	31K	2.0M	509K	1.2M
Average Degree	4.12	6.1	13.4	9.4
Maximal Degree	64	588	3079	425
#Connected Component	1781	73K	11	1
Largest Component Size	6794	517K	76K	262K
Average Component Size	8.6	9.0	6.9K	262K

- WC model: This is the *weighted cascade* model proposed in [9]. In this model, $pp(u, v)$ for an edge (u, v) is $1/d(v)$, where $d(v)$ is the in-degree of v. Thus even if the original graph is undirected, the model will generate asymmetric and nonuniform propagation probabilities.
- TRIVALENCY model: On every edge (u, v), we uniformly at random select a probability from the set $\{0.1, 0.01, 0.001\}$, which corresponds to high, midium, and low influences.

Algorithms. We compare our MIA heuristic with both the greedy algorithm and several heuristics that appear in the literature. The following is a list of algorithms we evaluate in our experiments.

- PMIA(θ): Our Algorithm 4 for the PMIA model with influence threshold θ. The value of θ for a particular dataset is selected using the heuristic discussed in the "tuning of parameter θ" part of Section 4.2.
- Greedy: The original greedy algorithm on the IC model [9] with the lazy-forward optimization of [13]. For each candidate seed set S, 20000 simulations is run to obtain an accurate estimate of $\sigma_I(S)$.
- DegreeDiscountIC: The degree discount heuristic of [3] developed for the uniform IC model with a propagation probability of $p = 0.01$, same as used in [3].
- SP1M: The shortest-path based heuristic algorithm of [10], also enhanced with the lazy-forward optimization of [13].
- PageRank: The popular algorithm used for ranking web pages [2]. Here the transition probability along edge (u, v) is $pp(v, u)/\rho_u$, where ρ_u is the sum of propagation probabilities on all incoming edges of u. Note that in the PageRank algorithm the transition probability of (u, v) indicates u's "vote" to v's ranking, and thus if $pp(v, u)$ is higher, v is more influential to u and thus u should vote v higher. 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.



(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.

- Random: As a baseline comparison, simply select k random vertices in the graph.

We ignore other centrality measures, such as distance centrality and betweenness centrality [7] as heuristics, since we have shown in [3] that distance centrality is very slow and has very poor influence spread, while betweenness centrality would be much slower than distance centrality.

To obtain the influence spread of the heuristic algorithms, for each seed set, we run the simulation on the networks 20000 times and take the average of the influence spread, which matches the accuracy of the greedy algorithms. The experiments are run on a server with 2.33GHz Quad-Core Intel Xeon E5410 and 32G memory.

We conduct further experiments using more datasets, more variants of the IC model, and more heuristic algorithms. The results are similar and are included in the appendix.

4.2 Experiment results

Scalability on the synthetic dataset. To test scalability, we generate a family of graphs of increasing sizes using the DIGG package [4], which applies the random power-law graph model of [1] to generate random graphs. We use graphs of doubling sizes — $2K$, $4K$, $8K$, ..., up to $256K$ in the number of nodes, and a power-law exponent of 2.16. The average degree of these graphs is between 2 and 3 for these graphs, which is lower than the real networks in Table 1. We use the WC model for the graphs, and run PMIA algorithm with a fixed $\theta = 1/320$, as well as other algorithms, to find 50 seeds in every graph. The result is shown in Figure 1, with normal scale shown in (a) and log-log scale of the same figure shown in (b) to differentiate different algorithms better.

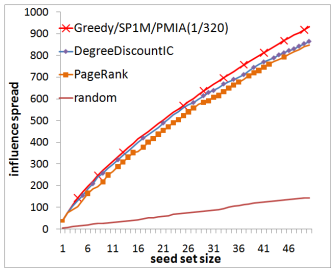
The result in Figure 1 (a) clearly separate all algorithms into two groups. Algorithms Greedy and SP1M are not scalable: their running times are in the hour range with around $400K$ edge graphs and it becomes infeasible to run them in larger graphs since we want to take average of 10 runs of every algorithm. Note that we already choose low average degree graphs so that they could run faster. Later reports on real graphs will show that they run even slower on those graphs. Our PMIA along with the rest heuristics can all scale up quite well. Figure 1 (b) differentiates the algorithms further. SP1M has the worst slope and is certainly not feasible for large-scale graphs. Greedy has the similar slope as other algorithms but its intercept is too large, because its Monte-Carlo simulation-based estimation of incremental influence spread for every node is too slow. Our PMIA has both good slope and intercept, making it easily scalable to large graphs with millions of edges.

Influence spread and running time for the real-world datasets We run tests on the four datasets and the two IC models to obtain influence spread results. The seed set size k ranges from 1 to 50. For ease of reading, in all influence spread figures (best viewed in color), the legend ranks the algorithms top-down in the same order as the influence spreads of the algorithms when $k = 50$. Moreover, if two curves are two close to each other, we group them together and show properly in the legend. All percentage difference reported below on influence spreads are the average of percentage differences from selecting one seed to selecting 50 seeds. Taking average is reasonable, since some algorithms may behave better when selecting the first few seeds while other algorithms behave better when selecting more seeds. The running time results are the time for selecting 50 seeds.

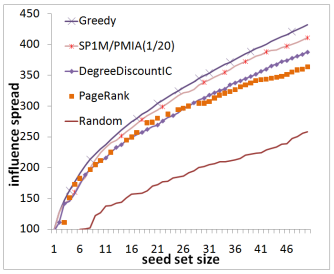
Figures 2–5 show the results on influence spreads for the four datasets on two IC models, while Figure 6 shows the running time results of the four datasets on the WC model (results on the TRIVALENCY model are similar and omitted).

For the moderate sized graph NetHEPT where Greedy is still feasible to run, the influence results in Figure 2 shows that Greedy produces the best influence spread, but PMIA is very close to Greedy: its influence spread essentially matches that of Greedy for the WC model and is only 3.8% less than Greedy for the TRIVALENCY model. Comparing with other heuristics, PMIA performs quite well: it matches the influence spread of SP1M while outforms the rest heuristics in both models — in the WC model, PMIA is 3.9% and 11.4% better, while in the TRIVALENCY model, PMIA is 6.5% and 15.4% better, comparing to DegreeDiscountIC and PageRank respectively. Random has a much worse influence spread, indicating that a careful seed selection is indeed important to effective viral marketing results. When looking at the running time in Figure 6 for NetHEPT on WC, we clearly see that Greedy is already quite slow (1.3 hours), while PMIA only takes 1 second, more than three orders of magnitude better. PMIA is also more than one order of magnitude faster than SP1M, and is comparable with PageRank. DegreeDiscountIC is the best in running time, because it is simple and specially tuned for the uniform IC model.

Figure 3 shows the result on the Epinions dataset, a large network with half a million edges. The graph is already too large for Greedy to run, so Greedy is out of the picture. For the WC model, PMIA still matches the influence spread of SPIM while it has a large winning margin over DegreeDiscountIC

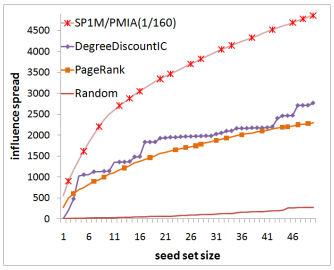


(a) WC model

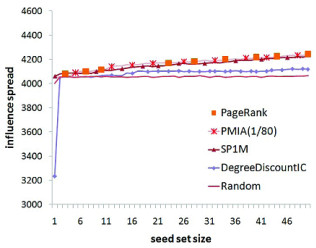


(b) TRIVALENCY model

Figure 2: Influence spread results on the NetHEPT dataset.

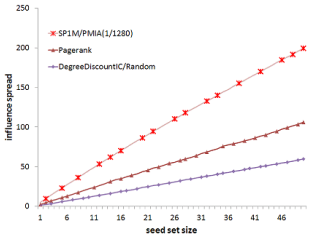


(a) WC model

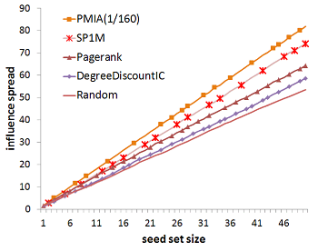


(b) TRIVALENCY model

Figure 3: Influence spread results on the Epinions dataset.

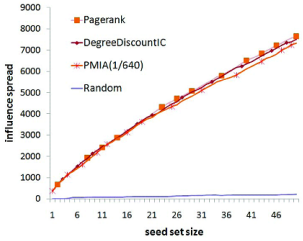


(a) WC model

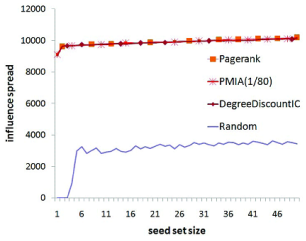


(b) TRIVALENCY model

Figure 4: Influence spread results on the Amazon dataset.



(a) WC model



(b) TRIVALENCY model

Figure 5: Influence spread results on the DBLP dataset.

and PageRank — PMIA is 96% and 115% better than DegreeDiscountIC and PageRank, respectively. This demonstrates that DegreeDiscountIC and PageRank are rather unstable heuristics while PMIA is very consistent in influence performance. For the TRIVALENCY model, we see that all heuristics, even Random reach a high level of influence spread after only a few seeds, while afterwards the increase in influence spread is slow. This behavior is quite different from the behavior of other test results we have seen so far, but it is very similar to a result presented in [9] for a graph when every edge has a propagation probability of 0.1. Therefore, we believe that the explanation is also similar: in this test, after deleting the edges based on their propagation probabilities and only keep the edges that will propagate influence, the resulting graph is likely to have a relatively large strongly connected component, and thus even random node selection would likely to hit this component after a few attempts, drastically increasing the influence spread. However, afterwards, additional seeds could only reach a small portion of still unaffected nodes, so further improvement in influence spread is small. But even in this case PMIA is still the best, outperforming the rest heuristics. For running time, we see that PMIA only takes 10 seconds but SP1M now takes 2.1 hours, more than 700 times slower than PMIA.

Next, for the one million-edge graph Amazon, Figure 4 shows that in the WC model PMIA again outperforms PageRank and DegreeDiscountIC with a large margin (99% and 266%, respectively), and in the TRIVALENCY model, it even outperforms SP1M significantly (14.1%, 23.9%, and 41.7% better than SP1M, PageRank, DegreeDiscountIC, respectively). Two unique features for this dataset are: (a) the influence spread is rather small, e.g. in TRIVALENCY, 50 seeds only generate a spread of around 80 nodes, and (b) the increase in influence spread is almost linear. The two features have the same reason — influence is very local and cannot propagate very far. It is probably because Amazon is a product co-purchasing network, not a social network. For running time, we now see that SP1M takes 30 hours, reaching its feasibility limit, while PMIA still only takes 10 seconds, showing its superb scalability over SP1M.

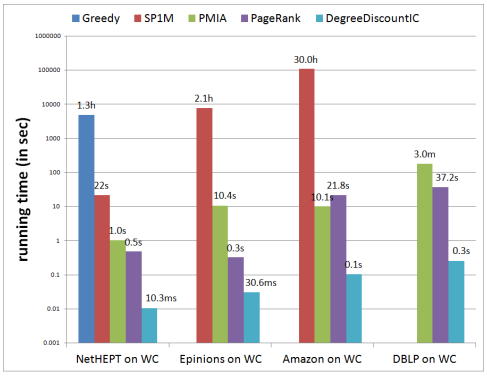


Figure 6: Running time of different algorithms in for datasets

Finally, for the two million edge DBLP dataset, Figure 5 shows that this time PageRank and DegreeDiscountIC matches PMIA and are slightly better than PMIA for the WC model. Looking at all test cases (including additional ones in the appendix), only a couple of cases where other scal-

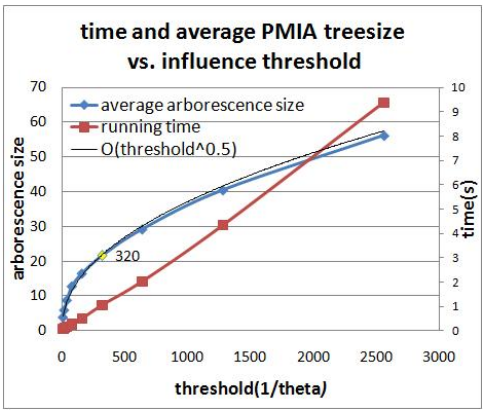


Figure 7: Running time and average arborescence size of PMIA vs. the threshold $1/\theta$ in the WC model, for NetHEPT dataset.

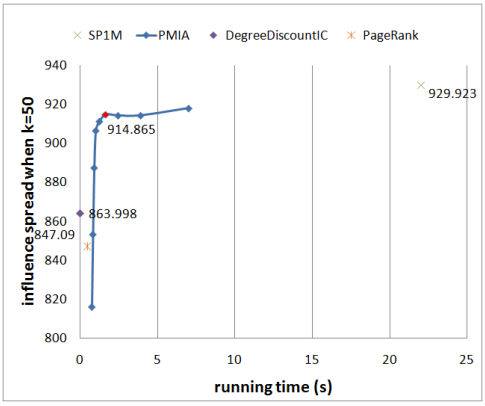


Figure 8: maximal influence spread by 50 seeds w.r.t. running time, for the NetHEPT dataset in the WC model.

able heuristics have matching influence spread as PMIA. This means that PMIA performs consistently well among the best scalable heuristics while others such as PageRank and DegreeDiscountIC are not stable — there exist a few cases that they perform well but in most other cases they performs not as well and sometimes they performs poorly comparing to PMIA. For running time, even at two million edge range, PMIA only takes 3 minutes to run. Therefore, PMIA has very good scalability and can handle million-sized or even larger graphs well.

Overall, we see that PMIA can scale beyond millions of edges, while Greedy and SP1M become too slow for half million edges or above. In all size ranges, PMIA consistently performs among the best algorithms (including Greedy and SP1M), while in most cases it significantly outperforms the rest scalable heuristics to as much as 100%–260% increase in influence spread.

Tuning of parameter θ. We investigate the effect of the tuning parameter θ on the running time and the influence spread of our algorithm. Figure 7 shows that the running time increases when the θ value decreases, as expected. More interestingly, the running time is almost linear to $1/\theta$. This can be roughly explained as follows. First, by the running time analysis of Section 3.2, we can see that when n and k are fixed and θ varies, the dominant term is a quadratic term $n_{o\theta}n_{i\theta}$, which means the running time is proportional to the square of the average arborescence size. Figure 7 further shows that the average arborescence size is about $O(\sqrt{1/\theta})$. Therefore together the running time is close to a linear relationship with $1/\theta$.

Figure 8 shows the change of influence spread with respect to the running time of our algorithm for the NetHEPT set in the WC model. Since the relationship between running time and $1/\theta$ is linear, it does not matter much if we use running time or $1/\theta$ as x-axis. The result indicates that as running time increases (θ decreases), the influence spread also increases, meaning that we obtain better quality results. Comparing other algorithms also shown in the figure, we see that on one side, we can tune $1/\theta$ to a larger value so that our influence spread can match the one provided by SP1M with at least 10 times speedup, while on the other side we can tune $1/\theta$ to a small value to get close to the running time of PageRank with matching influence spread. Therefore, we can use one algorithm to achieve different efficiency-effectiveness tradeoff needs by properly tuning the parameters.

One noticeable result is the knee in the curve of our algorithm. It means that the increase in influence spread is no longer significant after we lower θ to a certain level. This is because as shown in Figure 7, arborescence size increases in square root of $1/\theta$ (and thus in square root of running time), while influence spread may change much slower after the arborescence grows beyond a certain size. The knee point suggests a good tuning point for the algorithm. If we select θ such that the influence-time tradeoff is close to the knee point, we could obtain the best gain from both influence spread and running time. Correlating with Figure 7, we found that the corresponding knee point to be close to the point where the change of arborescence size slows down (the dot with $1/\theta = 320$). We observe similar situations in other dataset that we did not report here. Thus, this suggests the following way of tuning parameter θ. Given a new graph, randomly sample a small portion of nodes in the graph to compute the average arborescence sizes with varying $1/\theta$, and find a point where the change of arborescence size slows down, and use the θ value at that point for the *PMIA* algorithm. The θ values selected in our experiments are based on this method.

5 Future Work

One possible future research is to further explore the advantages of our MIA heuristic. For example, we believe that MIA heuristic fits into the parallel computation framework better than the greedy algorithm and shortest-path based SP1M heuristic. This is because our computation are restricted on local arborescences around nodes, and thus the graph can be eas-

ily partitioned for parallel computation, with sharing data only needed for arborescences at the boundary. On the contrary, the greedy algorithm and the SP1M heuristic need simulations and computations among the whole graph, so graph partition is difficult, and parallel computation is only possible for different computation tasks that require sharing of the entire graph. Another future direction is to look for hybrid approaches that combine the advantages of different algorithms to further improve the efficiency and effectiveness of influence maximization.

Beyond influence maximization, one interesting direction that requires further research is the data mining of social influence from real online social network data sets. A few studies have started to address this issue for blogspace [8] and academic collaboration network [19]. In fact, we used a dataset from [19] with propagation probabilities computed by their algorithm, but the graph size is small and thus we only include the result in Appendix A. We plan to study social influence mining in other social media and design appropriate algorithms for these social media. Social influence mining and influence maximization together will form the key components that enable prevalent viral marketing in online social networks.

References

[1] W. Aiello, F. R. K. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 171–180, 2000.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[3] 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.

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

[5] 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*, pages 57–66, 2001.

[6] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[7] L. Freeman. Centrality in social networks: conceptual clarification. *Social Networks*, 1:215–239, 1979.

[8] D. Gruhl, R. V. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th international conference on World Wide Web*, pages 491–501, 2004.

[9] 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*, pages 137–146, 2003.

[10] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 259–271, 2006.

[11] J. Leskovec. Amazon product co-purchasing network, march 02 2003. http://snap.stanford.edu/data/amazon0302.html.

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

[13] 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*, pages 420–429, 2007.

[14] I. R. Misner. *The World's best known marketing secret: Building your business with word-of-mouth marketing*. Bard Press, 2nd edition, 1999.

[15] J. Nail. The consumer advertising backlash, May 2004. Forrester Research and Intelliseek Market Research Report.

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

[17] 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*, pages 61–70, 2002.

[18] M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. Technical Report Technical Report CMU-CS-07-171, Carnegie Mellon University, 2007.

[19] 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.

[20] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[21] V. V. Vazirani. *Approximation Algorithms*. Springer, 2004.

Table 2: Statistics of NetPHY and DM.

Dataset	NetPHY	DM
#Node	37K	679
#Edge	174K	1687
Average Degree	12.5	4.97
Maximal Degree	286	63
#Connected Component	3883	1
Largest Component Size	19873	679
Average Component Size	9.57	679

Appendix

A Additional experiment results

In this section, we report additional results of our experiments on additional datasets, new propagation propability type for the IC model, and additional heuristic algorithms.

Additional datasets. Two additional datasets are tests. The first one from the full paper list of the "Physics" section of e-print arXive, doted as NetPHY, which contains $37,154$ nodes and $231,584$ edges, the same one used in [3]. The second dataset is obtained from the authors of [19], which is another collaboration network extracted from the data mining research area in the ArnetMiner archive (http://www.arnetminer.org) with 679 nodes and 1687 edges, and is denoted as DM. Some basic statistics about these networks are given in Table 2. Finally, in the scalability test, we use synthetic data to obtain networks of different sizes.

Generating propagation probabilities. We use one more model to generate propagation probabilities, as described below. We also use a different set of values for the TRIVALENCY model.

- TAP model: This is a model developed recently in [19], in which the authors develop a *topical affinity propagation* (TAP) algorithm to compute propagation probabilities of every edge based on structural and topical information available to the graph. The resulting propagation probabilities are also nonuniform. For the DM dataset, we use the propagation probabilities computed from the topical information available to the dataset. For the NetHEPT dataset, we use uniform topic distribution among nodes for TAP to compute propagation probabilities, since specific topical information is not available. The NetPHY dataset is too large for the TAP algorithm, so we do not use it for this data.
- TRIVALENCY model: use probability values 0.2, 0.04, 0.008 instead of 0.1, 0.01 and 0.001 in the main text.

Algorithms. We include the following additional algorithms for comparison.

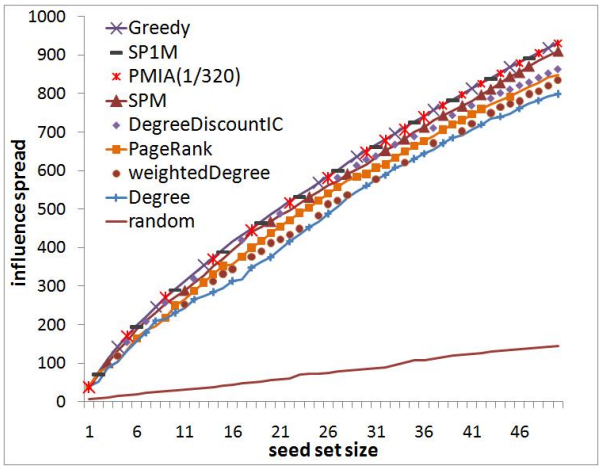


Figure 9: Influence spread for different algorithms in the WC model, for the NetHEPT dataset.

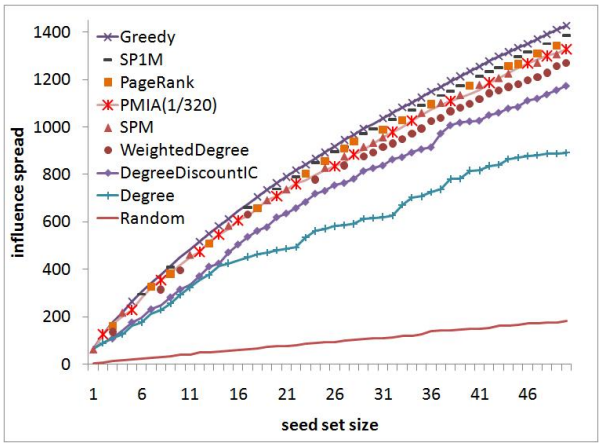


Figure 10: Influence spread for different algorithms in the WC model, for the NetPHY dataset.

- Degree: The simple heuristic that selects the k nodes with the largest out-degrees in the graph.
- WeightedDegree: The weighted degree of a node is the sum of propagation probabilities on all its outgoing edges. This heuristic selects the k nodes with the largest weighted degrees.
- SPM: The shortest-path based algorithm of [10], also enhanced with the lazy-forward optimization of [13]. In this version, only the shortest paths from S to a node v are counted for influence. Note that SP1M is an enhanced version of SPM, in which both the shortest paths and paths one hop longer than the shortest paths from S to a node v are counted for influence.

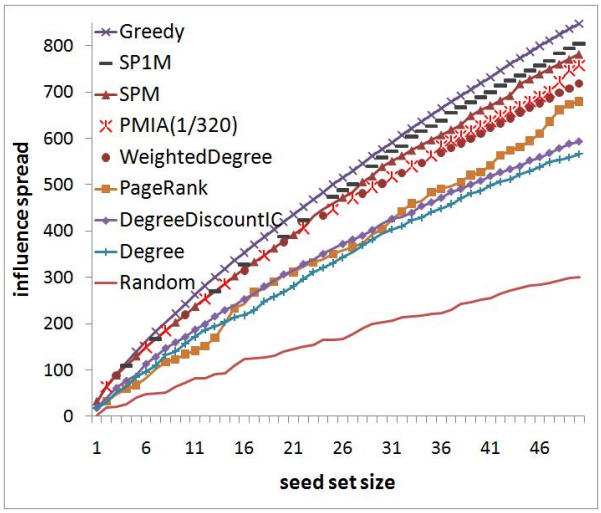


Figure 11: Influence spread for different algorithms in the TAP model, for the NetHEPT dataset.

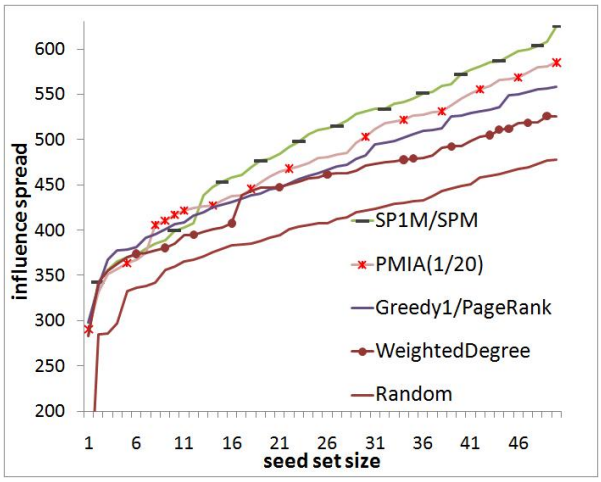


Figure 12: Influence spread for different algorithms in the TRIVANLENCY model with three probabilities 0.2, 0.04, 0.008, for the NetHEPT dataset.

Results on influence spread. Figures 9–13 shows the results on influence spreads, where we also include results for algorithms we tested in the main text. The results are mainly self-explanatory, and consistent with the finding we concluded in the main text. Overall PMIA performs consistently well over all datasets and all propagation models, matching or very close to the performance of Greedy and SPM/SP1M while outperform the rest heuristics, including the new ones we tested here. A special attention is on Figure 12, which shows that Greedy performs visibly worse than PMIA. The reason is Greedy is too slow and we have to reduce the number of simulations for influence spread estimation from 20000 to 200, causing it to lose accuracy on estimation (see the running time section for a reason why it is slow). This is also an indication that we can-

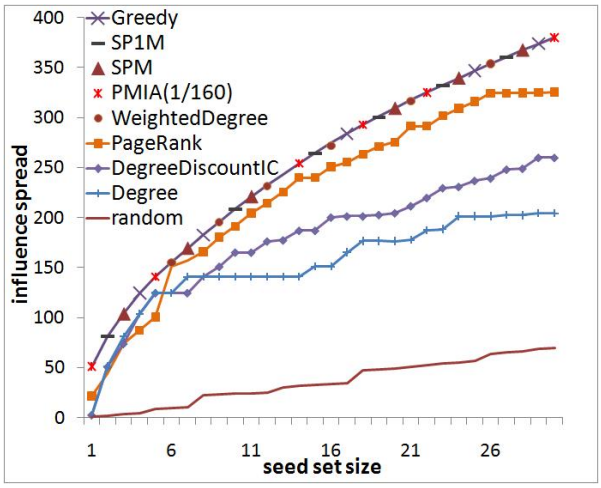


Figure 13: Influence spread for different algorithms in the TAP model, for the DM dataset.

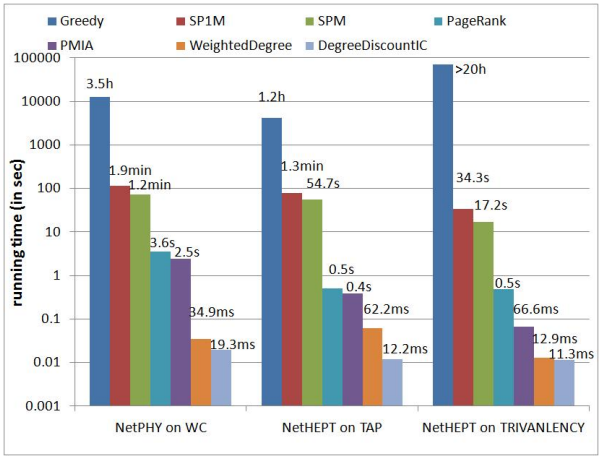


Figure 14: Running time of different algorithms in 3 datasets

not easily speed up Greedy by reducing the number of simulations. Another point worth explanation is that WeightedDegree performs quite well, closing to PMIA, in the two TAP model related tests (Figures 11 and 13). The reason is because WeightedDegree only considers influence propagated within one-step neighbors while the TAP model is likely to generate influence model in which most influences are indeed only propagate within one step. However, WeightedDegree performs not as well in other tests, showing that it is not consistent as PMIA.

Running time. Figure 14 shows the running time of different algorithms when selecting 50 seeds for 3 different tests: NetPHY using the WC model, NetHEPT using the TAP model, and NetHEPT using the TRIVANLENCY model (with probabilities 0.2, 0.04, and 0.008). The result is again consistent with what we have seen in the main text. Two specific points we would like to explain are as follows. First, Greedy is much slower in the TRIVALENCY model. This is because in this

model after selecting a seed, the marginal influence spread for the next seed candidate decreases dramatically, causing a lot of re-evaluations of marginal influence spread for selecting the next seed and making the lazy forward optimization of [13] much less effective than in other cases. Second, the running time of PMIA in the third test (NetHEPT on TRIVALENCY) is very fast ($67ms$). The reason that it is much faster than the other cases is because it uses a larger θ value of $1/20$, which generate smaller arborescences with depth at most 1. In this case, its running time is always close to that of the WeightedDegree, with the overhead only in the maintenance of the arboresence data structures and repeated updates due to seed selection. Thus we see that tuning θ could achieve much better running time. On the other hand, our PMIA is still better than WeightedDegree in influence spread (see Figure 12), because it considers overlapping influences among seeds while WeightedDegree does not.