Scalable Adversarial Attack Algorithms on Influence Maximization

Lichao Sun
Bethlehem, PA, USA
lis221@lehigh.edu

Xiaobin Rui∗
China University of Mining and Technology
Xuzhou, Jiangsu, China
ruixiaobin@cumt.edu.cn

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

ABSTRACT
In this paper, we study the adversarial attacks on influence maximization under dynamic influence propagation models in social networks. In particular, given a known seed set $S$, the problem is to minimize the influence spread from $S$ by deleting a limited number of nodes and edges. This problem reflects many application scenarios, such as blocking virus (e.g., COVID-19) propagation in social networks by quarantine and vaccination, blocking rumor spread by freezing fake accounts, or attacking competitor’s influence by incentivizing some users to ignore the information from the competitor. In this paper, under the linear threshold model, we adapt the reverse influence sampling approach and provide efficient algorithms of sampling valid reverse reachable paths to solve the problem. We present three different design choices on reverse sampling, which all guarantee $1/2 - \epsilon$ approximation (for any small $\epsilon > 0$) and an efficient running time.

CCS CONCEPTS
• Information systems → Social advertising; Social networks; • Theory of computation → Probabilistic computation; Submodular optimization and polymatroids.

KEYWORDS
influence maximization, triggering model, greedy algorithm

ACM Reference Format:
Lichao Sun, Xiaobin Rui, and Wei Chen. 2023. Scalable Adversarial Attack Algorithms on Influence Maximization. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (WSDM ’23), February 27-March 3, 2023, Singapore, Singapore. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3539597.3570416

1 INTRODUCTION
Influence maximization (IM) is the optimization problem of finding a small set of most influential nodes in a social network that generates the largest influence spread, which has many applications such as promoting products or brands through viral marketing in social networks [10, 16, 22]. However, in real life, there are so many competitions in various scenarios with different purposes, such as attacking competitor’s influence [18], controlling rumor [4, 14], or blocking the virus spread. The adversary will block the virus propagation by quarantine and vaccination, block rumor spread by freezing fake accounts, or attack competitors’ influence by incentivizing some users to ignore the information from the competitor. All these scenarios can be modeled as the adversary trying to remove certain nodes and edges to minimize the influence or impact from the competitor’s seed set in social networks, which we denoted as the adversarial attacks on influence maximization (AdvIM).

We model the AdvIM task more formally as follows. The social influence network is modeled as a weighted network $G = (V, E, w)$, where $V$ is a set of nodes representing individuals, $E$ is a set of directed edges representing influence relationships, and $w$ is influence weights on edges. In the beginning, we have a fixed seed set $S$, and the propagation from $S$ follows the classical linear threshold model [16]. For an attack set $A$ consisting of a mix of nodes (disjoint from $S$) and edges, we measure the effectiveness of $A$ by the influence reduction $\rho_S(A)$ it achieves, which is defined as the difference in influence spread with and without removing nodes and edges in the attack set $A$. Then, given a node budget $q_N$ and an edge budget $q_E$, the AdvIM task is to find an attack set $A$ with at most $q_N$ nodes (excluding any seed node) and at most $q_E$ edges, such that after removing the nodes and edges in $A$, the influence spread of $S$ is minimized, or the influence reduction $\rho_S(A)$ is maximized.

We show that under the LT model, the influence reduction function $\rho_S(A)$ is monotone and submodular, which enables a greedy approximation algorithm. However, the direct greedy algorithm is not efficient since it requires a large number of simulations of propagation from the seed set $S$. In this paper, we adapt the reverse influence sampling (RIS) approach to design efficient algorithms for the AdvIM task. Due to the nature of the problem, only successful propagation from the seed set can be potentially reduced by the attack set. This creates a new challenge for reverse sampling. In this paper, we present three different design choices for such reverse sampling and their theoretical analysis. They all provide a $1/2 - \epsilon$ approximation guarantee and have different trade-offs in efficiency. We then conduct experimental evaluations on several real-world networks and demonstrate that our algorithms achieve good influence reduction results while running much faster than existing greedy-based algorithms. To summarize, our contributions are: (a) proposing the study of adversarial attacks on influence maximization problems; (b) designing efficient algorithms by adapting the RIS approach and providing their theoretical guarantees; and (c) conducting experiments on real-world networks to demonstrate the effectiveness and efficiency of our proposed algorithms.
Related Work. Domingos and Richardson first studied Influence Maximization (IM) [10, 22], and then IM is mathematically formulated as a discrete optimization problem by Kempe et al. [16], who also formulate the independent cascade model, the linear threshold model, the triggering model, and provide a greedy approximation algorithm based on submodularity. After that, most works focus on improving the efficiency and scalability of influence maximization algorithms [3, 8, 9, 15, 26, 27, 29]. The most recent and the state of the art is the reverse influence sampling (RIS) approach [3, 21, 25–27], and the IMM algorithm of [26] is one of the representative algorithms for the RIS approach. Some other studies look into different problems, such as competitive and complementary influence maximization [4, 6, 12, 14, 19, 28], adoption maximization [2], robust influence maximization [7, 13], etc. The most similar topic to this work is competitive influence maximization that aims to maximize the influence while more than one party is in the network.

One similar work also wants to stop the influence spread in the social network [17]. However, this work has a different objective function that estimates the total influence by summing up the influence of the individual node in seed sets, which is different from the traditional influence maximization. They proposed the greedy approach through forward-tree simulation without time-complexity analysis. In this work, our objective function directly matches the original influence maximization objective function. Moreover, we propose RIS-based algorithms to overcome the efficiency issue in the forward simulation approach while also providing theoretical guarantee on both the approximation ratio and the running time.

The full version of the paper with complete proofs and other technical details is available at [23].

2 MODEL AND PROBLEM DEFINITION

Adversarial Attacks on Diffusion Model. In this paper, we focus on the well-studied linear threshold (LT) model [16] as the basic diffusion model. A social network under the LT model is modeled as a directed influence graph \( G = (V, E, w) \), where \( V \) is a finite set of vertices or nodes, \( E \subseteq V \times V \) is the set of directed edges connecting pairs of nodes, and \( w : E \to [0, 1] \) gives the influence weights on all edges. The diffusion of information or influence proceeds in discrete time steps \( t = 0, 1, 2, \ldots. \) At time \( t = 0 \), the seed set \( S_0 \) is selected to be active, and also each node \( v \) independently selects a threshold \( \theta_v \) uniformly at random in the range \([0, 1]\), corresponding to users’ true thresholds. At each time \( t \geq 1 \), an inactive node \( v \) becomes active if \( \sum_{u \in S_{t-1}} w(u, v) \geq \theta_v \) where \( S_t \) is the set of nodes activated by time \( t - 1 \). The diffusion process ends when there are no more nodes activated in a time step.

Given the weights of all nodes \( v \in V \), we can construct the live-edge graph \( L = (V, E(L)) \), where at most one of each \( v \)'s incoming edges is selected with probability \( w(u, v) \), and no edge is selected with probability \( 1 - \sum_{u \in S_{t-1}} w(u, v) \). Each edge \( (u, v) \in E \) is called a live edge. Kempe et al. [16] show that the propagation in the linear threshold model is equivalent to the deterministic propagation via bread-first traversal in a random live-edge graph \( L \). An important metric in a diffusion model is the influence spread, defined as the expected number of active nodes when the propagation from the given seed set \( S_0 \) ends, and is denoted as \( \sigma(S_0) \). Let \( \Gamma(G, S) \) denote the set of nodes in graph \( G \) that can be reached from the node set \( S \). Then, by the above equivalent live-edge graph model, we have \( \sigma(S_0) = \mathbb{E}_L[|\Gamma(L, S_0)|] = \sum_L Pr[L|G] \cdot |\Gamma(L, S_0)| \), where the expectation is taken over the distribution of live-edge graphs, and \( Pr[L|G] \) is the probability of sampling a particular live-edge graph \( L \) in graph \( G \). As defined above, we have:

\[
p(u, L, G) = \begin{cases} w(u, v), & \text{if } \exists u : (u, v) \in L \\
1 - \sum_{w(u, v) \in E} w(u, v), & \text{otherwise}
\end{cases}
\]

which is the probability of the configuration of incoming edges for node \( v \) in \( L \); then, the probability of a particular live-edge graph \( L \) is \( Pr[L|G] = \prod_{u \in V} p(u, L, G) \). When we need to specify the graph, we use \( \sigma(S_0, G) \) to represent the influence spread under graph \( G \).

A set function \( f : V \to \mathbb{R} \) is called submodular if for all \( S \subseteq T \subseteq V \) and \( u \in V \setminus T \), \( f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T) \). Intuitively, submodularity characterizes the diminishing return property often occurring in economics and operation research. Moreover, a set function \( f \) is called monotone if for all \( S \subseteq T \subseteq V, f(S) \leq f(T) \). It is shown in [16] that influence spread \( \sigma \) for the linear threshold model is a monotone submodular function. A non-negative monotone submodular function allows a greedy solution to its maximization problem subject to a cardinality constraint, with an approximation ratio \( 1 - 1/e \), where \( e \) is the base of the natural logarithm [20]. This is the technical foundation for most influence maximization tasks.

Adversarial Attacks on Influence Maximization. The classical influence maximization problem is to choose a seed set \( S \) of size at most \( k \) seeds to maximize the influence spread \( \sigma(S, G) \). For the Adversarial Attacks Influence Maximization (AdvIM) problem, the goal is to select at most \( q_N \) nodes and \( q_E \) edges to be removed, such that the influence spread on a given seed set \( S \) is minimized. Let \( A \) denote a joint attack set, which contains a subset of nodes \( \Delta_N \subseteq V \setminus S \) and a subset of edges \( \Delta_E \subseteq E, i.e. A = \Delta_N \cup \Delta_E \). Denote the new graph after removing the nodes and edges in \( A \) as \( G' = G \setminus A \). We first define the key concept of influence reduction under the attack set \( A \).

Definition 1 (Influence Reduction). Given a seed set \( S \), the influence reduction under the attack set \( A \), denoted as \( \rho_S(A) \), is the reduction in influence spread from the original graph \( G \) to the new graph \( G' = G \setminus A \). That is, \( \rho_S(A) = \sigma(S, G) - \sigma(S, G') \).

Note that \( S \cap A = \emptyset \), which means we cannot attack any seed node. We can now define the main optimization task in this paper.

Definition 2. The Adversarial Attacks on Influence Maximization (AdvIM) under the linear threshold model is the optimization task where the input includes the directed influence graph \( G = (V, E, w) \), seed set \( S \), attack node budget \( q_N \), and attack edge budget \( q_E \). The goal is to find an attack set \( A \) to remove, which contains at most \( q_N \) nodes (excluding the seed set \( S \)) and \( q_E \) edges, such that the total influence reduction is maximized: \( A^* = \text{argmax}_{A} \left| \left| \Delta_N \right| \leq q_N, \left| \Delta_E \right| \leq q_E \right| \rho_S(A) \).

Before the algorithm design, we first establish the important fact that \( \rho_S(A) \) as a set function is monotone and submodular.

Lemma 1. Influence reduction \( \rho_S(A) \) for the LT model satisfies monotonicity and submodularity.

Proof. The proof is based on the live-edge graph representation of the LT model. The proof for the monotonicity property is
we aim to speed up the greedy approach by adapting the approach according to the definition of the influence reduction, we have:

\[ \rho_S(A_1 \cup \{x\}) - \rho_S(A_1) \geq \rho_S(A_2 \cup \{x\}) - \rho_S(A_2) \]

Then it is sufficient to show that for any fixed live-edge graph \( L \), for every node \( v \in V \setminus S \), if \( v \not\in \Gamma(L, S) \), \( v \not\in \Gamma(L \setminus A_1, S) \) but \( v \not\in \Gamma(L \setminus S, \Gamma(L \setminus A_1, S)) \), then \( v \not\in \Gamma(L, S) \) \( v \not\in \Gamma(L \setminus A_1, S) \), so \( v \not\in \Gamma(L, S) \) \( v \not\in \Gamma(L \setminus S, \Gamma(L \setminus A_1, S)) \) are on the path, and thus attacking any node or edge on the path would reduce the influence of \( v \); but if \( P_L \) is invalid, attacking a node or edge on \( P_L \) will not reduce the influence of \( v \) since any \( P_L \) is not influenced by \( S \). For convenience, sometimes we also use the notation of \( S \)-conditioned RR path \( P_L^S \), which is \( P_L \) when \( P_L \) is valid and if \( P_L \) is invalid. Let \( P^S \) be the probability subspace of VRR paths. Let \( n^\sigma(S) = \sigma(S) = |S| \). The following lemma connects the influence reduction of an attack set \( A \) with the VRR paths.

**Lemma 2.** For any given seed set \( S \) and attack set \( A \),
\[
\rho_S(A) = \sigma(S) \cdot E_L[|A \setminus V E(P_L^S) \cup \emptyset|] 
= \sigma(S) \cdot E_L[|A \setminus V E(P_L) \cup \emptyset|] \mid S \setminus V E(P_L) \cup \emptyset] 
\]

Proof. By definition, we have
\[
\rho_S(A) = E_L[|\{v \in V \setminus S \mid v \notin \Gamma(L, S) \setminus \Gamma(L \setminus A_1, S)\}|] 
= E_L[\big[\sigma(S) \cdot E_{U(\Gamma(V \setminus S))}[|\{v \in \Gamma(L, S) \setminus \Gamma(L \setminus A_1, S)\}|] \big] 
\]

The above property implies that we can sample enough RR path from the original space or VRR path from subspace \( P^S \) to accurately estimate the influence reduction of \( A \). More importantly, by Lemma 2 the optimal attack set can be found by seeking the optimal set of nodes and edges that intersect with (a.k.a. cover) the most number of VRR paths, which is a max-cover problem. Therefore, following the RIS approach, we turn the influence reduction maximization problem into a max-cover problem. We use the IMM algorithm 26 as the template, but other RIS algorithms follow the similar structure. Our algorithm AAIMM contains two phases,
estimating the number of VRR paths needed and greedy selection via max-cover, as shown in Algorithm 1. The two main parameters $\lambda'$ and $\lambda^\ast(t)$ used in the algorithm are given below:

$$\lambda' \leftarrow (2 + \frac{2}{3}e')\left(\ln\left(\left(\frac{n^2}{qN}\right)\left(\frac{m}{q_E}\right)\right) + t \ln n^+ + \ln \log_2 n^-\right) \cdot n^-$$

$$\lambda^\ast(t) \leq 2n \cdot (1/2 \cdot \alpha + \beta)^2 \cdot e^{-2}$$

In Phase 1, we generate $\theta$ valid VRR paths $\mathcal{R}$, where $\theta$ is computed to guarantee the approximation with high probability. In Phase 2, we use the greedy algorithm to find the $q_N$ nodes and $q_E$ edges that cover the most number of VRR paths. This greedy algorithm is similar to prior algorithms, and thus we omit it here. Phase 1 follows the IMM structure to estimate a lower bound of the optimal value, which is used to determine how many VRR paths needed. The main difference is that we need to sample valid RR paths, not the simple RR sets as before. This part will be discussed separately in the next section. Moreover, our solution space now is $\binom{n^+}{q_N} \binom{m}{q_E}$, and thus we replace the $\binom{n}{q}$ in the original IMM algorithm. Let $A^\ast$ be the optimal solution of the AdvIM problem, and $\text{OPT} = \rho_S(A^\ast)$.

**Lemma 3.** For every $\epsilon > 0$ and $\delta > 0$, to guarantee the approximation ratio with probability at least $1 - \frac{1}{n^2}$, the number of VRR paths needed by AAIMM is $O\left(\frac{(qN \log n^+ + qE \log m + t \log n^-) \cdot \sigma^\ast(S)}{\text{OPT} \cdot e^2}\right)$.

\begin{algorithm}
\caption{AAIMM: Adversarial Attacks IMM}
\textbf{Input:} Graph $G = (V, E, w)$, seed set $S$, budgets $q_N, q_E$.
\begin{algorithmic}
\State // Phase 1: Estimate $\theta$, the number of VRR paths needed, and generate these VRR paths
\State $R \leftarrow \emptyset$; $LB \leftarrow 1$; $\epsilon' \leftarrow \sqrt{2}/n$; using binary search to find a $\gamma$ such that $\frac{\lambda'(\gamma)}{\gamma^{n+\epsilon'}} \leq 1/m^n$. $t \leftarrow t + \gamma + \ln 2/\ln n^-$;
\For {$i = 1$ to $\log_2(n - 1)$}
\State $x_i \leftarrow n^- / 2^i$;
\State $\theta_i \leftarrow \lambda' \cdot e^{-\epsilon} / x_i$; // $\lambda'$ is defined in Eq. (3)
\EndFor
\While {$|R| \leq \theta_i$} do
\State Sample a VRR path $P$ from subspace $P^S$, and insert it into $R$;
\State $A_i \leftarrow \text{NodeEdgeSelection}(R, q_N, q_E)$;
\If {$n^- \cdot F^*_R(A_i) \geq 1 + (1 + \epsilon') \cdot x_i$} then
\State $LB \leftarrow n^- \cdot F^*_R(A_i) / (1 + \epsilon')$;
\State break;
\EndIf
\EndWhile
\State $0 \leftarrow \lambda^\ast(t) / LB$; // $\lambda^\ast(t)$ is defined in Eq. (4)
\While {$|R| \leq \theta$} do
\State Sample a VRR path $P$ from subspace $P^S$, and insert it into $R$;
\EndWhile
\State // Phase 2: select attack nodes and edges from the generated VRR paths
\State $A \leftarrow \text{NodeEdgeSelection}(R, q_N, q_E)$;
\State return an attack set $A$
\end{algorithmic}
\end{algorithm}

Proof. When working on the subspace $P^S$ in AAIMM, we are working on the objective function $n^+ \cdot E_{p \sim q_S(\mathcal{E}(A \cap V \cap E = \emptyset))}$ (e.g. see lines 3, 8, and 9). Thus by Lemma 2, the real objective function $\rho_S(A)$ is only a fraction $\frac{\sigma^\ast(S)}{n^+}$ of the new objective function. Let $OPT'$ be the optimal value of the new objective function. Applying the analysis of IMM [26], we know that the number of VRR path samples that we need in the subspace $P^S$ is

$$O\left(\frac{(qN \log n^+ + qE \log m + t \log n^-) \cdot n^-}{OPT' \cdot e^2}\right).$$

Because $OPT \geq \frac{\sigma^\ast(S)}{n^+} \cdot OPT'$, the above formula is changed to

$$O\left(\frac{(qN \log n^+ + qE \log m + t \log n^-) \cdot \sigma^\ast(S)}{OPT' \cdot e^2}\right).$$

The special case of adversarial attacks on influence maximization is attacking nodes only or edge only, i.e., $q_E = 0$ or $q_N = 0$. Then, the greedy algorithms of the special cases can achieve at least $1 - 1/(e - \epsilon)$ of the optimal performance. However, our greedy algorithms for the AdvIM attacks both nodes and edges together. The idea of our greedy approach is that at every greedy step, it searches all nodes and edges in the candidate space and picks the one having the maximum marginal influence deduction. If the budget for node or edge exhausts, then the remaining nodes or edges are removed from $C$. Note that as $C$ contains nodes and edges assigned to different partitions, AAIMM selects nodes or edges crossing partitions. This falls into a greedy algorithm subject to a partition matroid constraint, which is defined below.

Given a set $U$ partitioned into disjoint sets $U_1, \ldots, U_n$ and $I = \{X \subseteq U : |X \cap U_i| \leq k_i, \forall i \in [n]\}$, $(U, I)$ is called a partition matroid. Thus, the node and edge space $\mathcal{A}$ with the constraint of AdvIM, namely $(\mathcal{A}, (A : |A_N| \leq q_N, |A_E| \leq q_E))$, is a partition matroid. This indicates that AdvIM is an instance of submodular maximization under partition matroid, which can be solved by a greedy algorithm with $1/2$-approximation guarantee [11]. Using this result, we can obtain the result for our AAIMM algorithm.

**Theorem 1.** For every $\epsilon > 0$ and $\delta > 0$, with probability at least $1 - \frac{1}{n^2}$, the output $A^\ast$ of the AAIMM algorithm framework satisfies $\rho_S(A^\ast) \geq \left(\frac{1}{2} - \epsilon\right) \rho_S(A^\ast)$. In this case, the expected running time for AAIMM is $O\left(\frac{(qN \log n^+ + qE \log m + t \log n^-) \cdot \sigma^\ast(S)}{OPT' \cdot e^2}\right)$.
3.2 VRR Path Sampling

VRR path sampling is the key new component to fully realize our AAIMM algorithm. Here, we discuss three methods and their guarantees, and empirically evaluate these methods in our experiments.

**Naive VRR Path Sampling.** The first implementation is naively generate a RR path \( P \) starting from a random root \( v \in V \setminus S \), and if \( P \) is valid then return it; otherwise regenerate a new path (see Algorithm 2). It is easy to see that to generate one RR path, we need to generate a number of RR paths. Let ERP be the expected running time of generating one RR path. By a simple argument based on live-edge graphs, we can get that on average we need to generate \( n^{-\gamma}/\sigma^-(S) \) RR paths to obtain one VRR path that reaches the seed set \( S \). This means ERPV = ERP \cdot n^{-\gamma}/\sigma^-(S).

### Algorithm 2: Naive-VRR-Path: Naive VRR Path Sampling

**Input:** Graph \( G \), seed set \( S \)

1. **repeat**
   1. Randomly select a root \( v \in V \setminus S \), and generate the reverse-reachable path \( P \) rooted at \( v \);
2. until \( P \cap S \neq \emptyset \);
3. return a VRR path \( P \).

**Theorem 2.** Naive VRR path sampling (Algorithm 2) correctly samples a VRR path. The expected running time of AAIMM with naive VRR path sampling (Algorithm 2) is

\[
O \left( \frac{(q_n \log n^{-\gamma} + q_G \log m + \ell \log n^{-\gamma}) \cdot n^{-\gamma}}{\text{OPT} \cdot \varepsilon^2} \cdot \text{ERP} \right).
\]  

**Proof.** It is obvious that the naive VRR path sampling will return a VRR path according to distribution \( P^S \). We just need to prove that ERPV = ERP \cdot n^{-\gamma}/\sigma^-(S), and the rest follows Theorem 1. For each live-edge graph \( L \), if we randomly select a root \( v \in V \setminus S \), then with probability \((|\Gamma(L, S)| - |S|)/n^{-\gamma} \), \( v \) is reachable from \( S \) in \( L \), which means the RR path from \( v \) will intersect with \( S \) on this live-edge graph \( L \). Taking expectation over \( L \), we know that the probability of an RR path is valid is \( E_L[(|\Gamma(L, S)| - |S|)/n^{-\gamma}] = \sigma^-(S)/n^{-\gamma} \). Therefore, on average we need to generate \( n^{-\gamma}/\sigma^-(S) \) RR paths to get one VRR path, which means ERPV = ERP \cdot n^{-\gamma}/\sigma^-(S).

**Forward-Backward VRR Path Sampling.** To avoid wasting RR path samplings as in the naive method, we can first do a forward simulation from \( S \) to generate a forward forest, recording the nodes and edges that a forward simulation from \( S \) will pass. Then, when randomly selecting a root \( v \), we restrict the selection to be among the nodes touched by the forward simulation. Finally, the VRR path is the path from \( S \) to \( v \) recorded in the forward forest. This is the forward-backward sampling method given in Algorithm 3. Let EFF(S) be the meantime of generating a forward forest from seed set \( S \). Then we have the following result on this method.

### Algorithm 3: FB-VRR-Path: Forward-Backward VRR Path Sampling

**Input:** Graph \( G \), seed set \( S \)

1. Initialize an empty forest \( F \);
2. Forward propagating a new forest \( F \) by using LT model with \( S \) and \( G \);
3. Randomly select a node \( v \in F \setminus S \), and set path \( P \) to be the one from \( S \) to \( v \) in the forest \( F \);
4. \( R^v \leftarrow R^v \cup P; \) return a VRR path \( P \).

**Theorem 3.** Forward-backward VRR path sampling (Algorithm 3) correctly samples a VRR path. The expected running time of AAIMM with forward-backward VRR path sampling (Algorithm 3) is

\[
O \left( \frac{(q_n \log n^{-\gamma} + q_G \log m + \ell \log n^{-\gamma}) \cdot \sigma^-(S)}{\text{OPT} \cdot \varepsilon^2} \cdot \text{EFF}(S) \right). \tag{8}
\]

**Proof.** For any fixed live-edge graph \( L \), conditioned on sampling a VRR path \( P \), random sampling a root \( v \) is equivalent to sampling from \( \Gamma(L, S) \setminus S \) uniformly at random, which is exactly from the forward forest generated by forward simulation from \( S \). Thus, the forward-backward sampling method is correct.

Compared with naive sampling, the forward-backward sampling saves those sampling of invalid RR paths. However, it needs to generate a complete forward forest first, which is more expensive than generating one reverse path. Therefore, there is a tradeoff between the forward-backward method and the naive method, and the tradeoff is exactly quantified by their running time results: If EFF(S)/ERP < n^{-\gamma}/\sigma^-(S), then the forward-backward method is faster; otherwise, the naive method is faster. Therefore, which one is better will depend on the actual graph instance.

Since the forward-backward method generates a forward forest first, one may be tempted to sample more VRR paths from this forest, but it will generate correlations among these VRR paths. Another attempt of generating a number of forward forests to record the frequencies of node appearances, and then use these frequencies to guide the RR path sampling would also deviate from the subspace probability distribution \( P^S \), and thus these methods would not provide theoretical guarantee of the overall correctness of AAIMM.

Reverse-Reachable Simulation with DAG. The naive method above wastes many invalid RR path samplings, while the forward-backward method wastes many branches in the forward forest. Thus, we desire a method that could do a simple VRR path sampling without such waste. For directed-acyclic graphs (DAGs), we do discover such a VRR path sampling method by re-weighting edges.

Suppose \( G \) is a directed acyclic graph. As shown in [9], in a DAG, influence spread of a seed set can be computed in linear time. Let \( ap_\sigma(S) \) be the probability of \( \sigma \) being activated when \( S \) is the seed set. Let \( N^-(\sigma) \) be the set of \( \sigma \)'s in-neighbors. Then in a DAG \( G \), we have \( a_{\sigma}(S) = \sum_{\sigma \in N^-(\sigma)} a_{\sigma}(S) \cdot \omega(\sigma, \sigma') \). The above computation can be carried out with one traversal of \( D \) in linear time from the seed set \( S \) following any topological sort order. For a DAG \( G \), we propose the following sampling of a VRR path in \( P^S \) as DAG-VRR-Path in Algorithm 4.
Algorithm 4: DAG-VRR-Path: DAG VRR Path Sampling

**Input:** Graph $G$, seed set $S$

1. Randomly sample root $v \in V \setminus S$ with probability proportional to $a_{(u)}(S)$, i.e. with probability 
   \[ \frac{a_{(u)}(S)}{\sum_{u \in V \setminus S} a_{(u)}(S)} = \frac{a_{(u)}(S)}{\sigma^-(S)}; \]
2. $u_0 \leftarrow v; P \leftarrow \{u_0\}; i \leftarrow 0$;
3. repeat
   4. Sample $u_{i+1} \in N^-(u_i)$ with probability proportional to 
      \[ \frac{a_{u_{i+1}}(S) \cdot w(u_{i+1}, u_i)}{\sum_{u \in N^-(u_i)} a_{u}(S) \cdot w(u, u_i)} = \frac{a_{u_{i+1}}(S) \cdot w(u_{i+1}, u_i)}{a_{u}(S)}; \]
   5. Add node $u_{i+1}$ and edge $(u_{i+1}, u_i)$ into path $P$;
   6. $i \leftarrow i + 1$;
4. until $u_i \in S$
5. return a VRR path $P$.

**Lemma 4.** If the graph $G$ is a DAG, then any path sampled by Algorithm DAG-VRR-Path follows the subspace distribution $p^S$.

**Proof.** First, notice that Algorithm DAG-VRR-Path re-weights the incoming edges $(u, v)$ of every node $v$ according to the activation probability $a_{u}(S)$. Therefore any node that cannot be activated by $S$ will cause $(u, v)$ to have zero weight, and thus will not be sampled in the reverse sampling process. Therefore, the reverse sampling process will always sample toward the seed set, and since the graph has no cycle, it will always end at a seed node in $S$. This means that the output of DAG-VRR-Path is always a VRR path. We just need to show that it follows the subspace distribution $p^S$.

To show that its distribution is the same as $p^S$, all we need to show is that if we have two paths $P_1$ and $P_2$ that both start from a seed node in $S$, then the ratio of the probabilities of generating these two paths by the above procedure is the same as the ratio in the original RR path space. Let $P_1 = (u_0, u_1, \ldots, u_0)$ and $P_2 = (v_0, v_1, \ldots, v_0)$, with $u_0, v_0 \in V \setminus S$ and $u_i, v_i \in S$. Let $\pi(P)$ be the probability of sampling the RR path $P$ in the original space. Then we have $\pi(P_1) = \frac{\pi^-(S)}{\pi^-(S)} \prod_{i=1}^{n} w(u_i, u_{i-1})$, and $\pi(P_2) = \frac{\pi^-(S)}{\pi^-(S)} \prod_{j=1}^{n} w(z_j, z_{j-1})$. So the ratio is

\[ \frac{\pi(P_1)}{\pi(P_2)} = \frac{\prod_{i=1}^{n} w(u_i, u_{i-1})}{\prod_{j=1}^{n} w(z_j, z_{j-1})}. \]  

Now let $\pi'(P)$ be the probability of generating path $P$ by DAG-VRR-Path. Then we have

\[ \pi'(P_1) = \frac{a_{u_{i+1}}(S)}{\sigma^-(S)} \prod_{i=1}^{n} \frac{a_{z_j}(S) \cdot w(z_j, z_{j-1})}{a_{z_{i+1}}(S)} = \frac{\prod_{i=1}^{n} w(u_i, u_{i-1})}{\sigma^-(S)}, \]  

Where the above derivation also uses the fact that $u_0 \in S$ and thus $a_{u_{i+1}}(S) = 1$. Similarly, we have

\[ \pi'(P_2) = \frac{a_{z_j}(S)}{\sigma^-(S)} \prod_{j=1}^{n} \frac{a_{z_j}(S) \cdot w(z_j, z_{j-1})}{a_{z_{j+1}}(S)} = \frac{\prod_{j=1}^{n} w(z_j, z_{j-1})}{\sigma^-(S)}. \]  

Therefore, clearly $\pi'(P_1)/\pi'(P_2) = \pi(P_1)/\pi(P_2)$, and the above procedure correctly generates a VRR path from $p^S$. \hfill \Box

Therefore we can use DAG-VRR-Path sample from $p^S$. Now we just need to analyze its running time ERPV. The generation is still similar to the LT reverse simulation. For any $u \notin S$, let $\tau(u)$ be the time needed to do a reverse simulation step from $u$. For the LT model, a simple binary search implementation takes $\tau(u) = O(\log_d d_u)$ time, where $d_u$ is the indegree of $u$. For an RR path $P$, let $\omega(P) = \sum_{u \in V \setminus S} \tau(u)$ be the total time needed to generate path $P$. Let $\tau = \sum_{u \in V \setminus S} \tau(u)$.

**Lemma 5.** Let $\tilde{\omega}$ be a randomly sampled node from $V \setminus S$, with sample probability proportional to $\omega(\tilde{\omega})$. Let $P$ be a random RR path generated by DAG-VRR-Path, then we have $\text{ERPV} = \mathbb{E}_{p \sim \text{RR}}[\omega(P) = \frac{\sum_{u \in V \setminus S} \tau(u)}{\sum_{u \in V \setminus S} \tau(u)}]$.

**Proof.** For a fixed RR path $P$, let $p(P)$ be the probability of $\tilde{\omega} \in V(P)$. Then it is clear that

\[ p(P) = \frac{\mathbb{E}_{\tilde{\omega} \sim \text{RR}}[\tilde{\omega} \in V(P)]}{\mathbb{E}_n[\tilde{\omega} \sim \text{RR}]} = \frac{\omega(P)}{\tau}. \]

Let $P$ be a random RR path generated by Algorithm 4. Then

\[ \mathbb{E}_{p \sim \text{RR}}[\omega(P)] = \tau \cdot \mathbb{E}_{p \sim \text{RR}}[p(P)] = \tau \cdot \mathbb{E}_{p \sim \text{RR}}[\mathbb{E}_n[\tilde{\omega} \in V(P)]] = \tau \cdot \mathbb{E}_n[\mathbb{E}_{p \sim \text{RR}}[\tilde{\omega} \in V(P)]] = \frac{\tau}{\sum_{u \in V \setminus S} \tau(u)} \cdot \mathbb{E}_n[\tilde{\omega} \sim \text{RR}], \]

where the last equality is by Lemma 2. \hfill \Box

Finally, applying the above ERPV result to Theorem 1, we can obtain the following

**Theorem 4.** Algorithm DAG-VRR-Path correctly samples a VRR path. The expected running time of RAIMM with DAG-VRR-Path sampling is $O\left(\log^2 n \log \log n \log^2 \log n \cdot \tau \cdot \mathbb{E}_n[\tilde{\omega} \sim \text{RR}][\tilde{\omega} \in V(P)]\right)$, where $\tau = O(\sum_{u \in V \setminus S} \log d_u) = O(n \log n)$.

Notice that when $q \geq 1$, we have $\mathbb{E}_n[\tilde{\omega} \sim \text{RR}][\tilde{\omega} \in V(P)] \leq \text{OPT}$. Therefore, in this case we will have a near-linear-time algorithm, just as the original influence maximization algorithm IMM.

The above result relies on that $G$ is a DAG. When the original graph is not DAG, we can transform the graph to a DAG, similar to the DAG generation algorithm in the LDAG algorithm (Algorithm 3 in [6]). The difference is that in [6] it is generating a DAG to approximate the influence towards a root $v$. Instead, in our case we want to generate a DAG that approximates the influence from seed set $S$ to other nodes. But the approach is similar, and we can efficiently implement this DAG generation process by a Dijkstra short-path-like algorithm just as in [6].

## 4 EXPERIMENTAL EVALUATION

### 4.1 Data and Algorithms

**DBLP.** The DBLP dataset [24] is a network of data mining, where every node is an author and every edge means the two authors collaborated on a paper. The original DBLP is a graph containing 6.546.28 × 10^5 nodes and 3.980.318 × 10^6 directed edges. However, due to the limited memory of our computer (16GB memory, 1.4GHz quad-core Intel CPU), it hardly launches the whole test on such graph. So we randomly sample 1.000.00 × 10^5 nodes and their 7.471.78 × 10^5 directed edges from the original graph, which is still the largest size of graph among all four datasets.
**NetHEPT.** The NetHEPT dataset [5] is extensively used in many influence maximization studies. It is an academic collaboration network from the “High Energy Physics Theory” section of arXiv from 1991 to 2003, where nodes represent the authors and each edge represents one paper co-authored by two nodes. We clean the dataset by removing duplicated edges and obtain a directed graph \( G = (V, E) \), \(|V| = 1.5233 \times 10^4\), \(|E| = 6.2774 \times 10^4\) (directed edges).

**Flixster.** The Flixster dataset [1] is a network of American social movie discovery services. To transform the dataset into a weighted graph, each user is represented by a node, and a directed edge from node \( u \) to \( v \) is formed if \( v \) rates one movie shortly after \( u \) does so on the same movie. The Flixster graph contains \( 2.9935 \times 10^4\) nodes and \( 6.2777 \times 10^4\) directed edges.

**DM.** The DM dataset [24] is a network of data mining researchers extracted from the ArnetMiner archive (aminer.org), where nodes present the researchers and each edge is the paper co-authorship between any two researchers. DM is the small size dataset here, which only includes \( 6.79 \times 10^2\) nodes and \( 3.374 \times 10^3\) directed edges.

### 4.2 Algorithms

We test all four algorithms proposed in the experiment, for the AdvIM task with different settings. Some further details of each algorithm are explained below.

**AA-FF.** This is the forward forest greedy algorithm. The number of forward-forest simulations of AA-FF is the same as the reverse-reachable approaches in Theorem 2. However, instead of using VRR paths, AA-FF simulate a forward forest, i.e., a set of trees, in each propagation, which is required to occupy a huge computer memory to ensure theoretical guarantee. To make it practical, we follow the standard practice in the literature and set the number of simulations as \( 10000\) [5, 16, 29]. The pseudo code and full analysis of AA-FF is given in the appendix of the full version [23].

**AA-IMM-Naive.** This is the AAIMM algorithm with naive VRR path simulation, as given in Algorithm 1 and Algorithm 2. AA-IMM-Naive needs to generate plenty of RR paths for enough VRR paths since most naive RR paths can not touch the seed set \( S \). Compared

![Figure 1](image-url)
to AA-FF, AA-IMM-Naive uses much less computer memory cost to ensure the theoretical guarantee due to the RIS approach.

**AA-IMM-FB.** This is the AAIMM algorithm with forward-backward VRR path simulation, as given in Algorithm 1 and Algorithm 3. Unlike AA-IMM-Naive, no path will be wasted in AA-IMM-FB, since all paths here are VRR paths that are randomly selected from the forward forests. To fairly compare the running time with AA-FF, we choose to sample the same number of simulations of the forward forests. The results show that compared with AA-FF, AA-IMM-FB can save more computer memory and computation power.

**AA-IMM-DAG.** This is the DAG-based AAIMM algorithm, as given in Algorithm 1 and Algorithm 4. Before simulation the VRR paths from DAG, we need first create a DAG as same as [6]. Compared to previous RIS approaches, AA-IMM-DAG is the fastest approach for VRR path sampling.

We also use 10000 Monte Carlo simulations for influence spread estimation after adversarial attacks for all the above approaches.

### 4.3 Result

We test all four algorithms proposed in the experiment, for the AdvIM with $k$ seeds budget, such as AA-FF, AA-IMM-Naive, AA-IMM-FB and AA-IMM-DAG algorithms. We use $R = 10000$ for the AA-FF and AA-IMM-FB on all datasets due to the high-cost computing resource and memory usage. Note that, due to the high memory cost, AA-FF can not finish the whole test even $R = 10000$. In all tests, we set the seed set $k = 50, 100, 200, 300$ for DM, Flixster, NetHEPT and DBLP respectively, and we also test different combinations of $q_N$ and $q_E$. For clear representation, we leave out “AA-” in the algorithms’ names in Figure 1 and Figure 2.

**Influence Spread Performance.** From Figure 1, it is not hard to see that the influence of the selected seed set is decreasing while increasing either the node budget $q_N$ or the edge budget $q_E$ over all 16 tasks. On DBLP, we can see that $AA-IMM-FB > AA-IMM-DAG > AA-IMM-DAG$ based on the performance on influence reduction, and AA-FF can not finish the test with $R = 10000$. NetHEPT has similar ranking results as DBLP that we have $AA-IMM-FB = AA-FF > AA-IMM-Naive > AA-IMM-DAG$ in most cases. However, on Flixster, the ranking becomes very different. AA-IMM-DAG becomes the best in most cases, and AA-FF and AA-IMM-FB show the worst influence reduction performance in Figures 1 (c) and (g). On DM, all methods perform close to each other. In summary, the results demonstrate that all algorithms perform close to each other. In general, either AA-IMM-DAG and AA-FF can achieve the best performance in different tasks. In this case, the running time becomes the key while applying the algorithms in real applications.

**Running time.** Figure 2 reports the running time of all the tested algorithms on the four datasets. One clear conclusion is that all IMM algorithms are much more efficient than AA-FF that even fails to finish the test on DBLP. Among three RIS algorithms, it is obvious that AA-IMM-DAG is the fastest algorithm, AA-IMM-Naive is the second, and the AA-IMM-FB is the slowest one. However, from DBLP’s results, AA-IMM-Naive is slower than AA-IMM-FB, which is not due to the limited number of simulations of the AA-IMM-FB, i.e., $R = 10000$. More importantly, AA-IMM-DAG is at least 10 times faster than all other algorithms, including other RIS-based approaches. After combining the influence spread and running time performance together, we recommend AA-IMM-DAG and AA-IMM-FB algorithms to be the best two choices for AdvIM, and AA-FF to be the worst choice due to the cost of high computer memory and high running time.

### 4.4 Discussion

From the experiments, we found several interesting aspects. The most important is why AA-FF algorithm took so much memory and computing resource compared to VRR path simulation approaches. The primary reason is that AA-FF algorithm takes too much memory space. For RR path simulation, no matter how big the original seed set is, we only save one single path per simulation. However, in each AA-FF simulation, the size of the forest is related not only to the graph’s properties but also to the size of the target seed set. While the target seed set contains 100 nodes, there are 100 sub-tree in each simulation by the AA-FF algorithm. It means AA-FF can not be practical in a real application. For example, if we want to stop COVID-19 with a virus spread graph, the seed sets may be thousands in a vast network. In this case, VRR-based path simulation is the best for a large graph simulation for Adv-IM.
REFERENCES

[1] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. 2012. Topic-aware social influence propagation models. In Proceeding of the 12th IEEE ICDM International Conference on Data Mining. 81–90.

[2] Smriti Bhagat, Amit Goyal, and Laks V. S. Lakshmanan. 2012. Maximizing product adoption in social networks. In Proceedings of the Fifth WSDM Conference on Web Search and Data Mining. 603–612.

[3] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 946–957.

[4] Ceren Budak, Deryakant Agrawal, and Amir El Abbadi. 2011. Limiting the spread of misinformation in social networks. In Proceedings of the 20th WWW International Conference on World Wide Web. 665–674.

[5] Ning Chen. 2008. On the approximability of influence in social networks. In Proceedings of the Nineteenth Annual SIAM SODA Symposium on Discrete Algorithms. 1029–1037.

[6] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincon, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. 2011. Influence maximization in social networks when negative opinions may emerge and propagate. In Proceedings of the Eleventh SIAM SDM International Conference on Data Mining. 379–390.

[7] Wei Chen, Tian Lin, Zihan Tan, Mengfei Zhao, and Xuren Zhou. 2016. Robust Influence Maximization. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 795–804.

[8] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 199–208.

[9] Wei Chen, Yifei Yuan, and Li Zhang. 2010. Scalable Influence Maximization in Social Networks under the Linear Threshold Model. In Proceedings of the 10th IEEE ICDM International Conference on Data Mining. 88–97.

[10] Pedro Domingos and Matthew Richardson. 2001. Mining the network value of customers. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. 57–66.

[11] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. 1978. An analysis of approximations for maximizing submodular set functions II. In Polyhedral combinatorics. Springer.

[12] Lifang He, Chun-Tu Lu, Jiaqi Ma, Jianping Cao, Linlin Shen, and Philip S Yu. 2016. Joint community and structural hole spanner detection via harmonic modularity. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 875–884.

[13] Xinran He and David Kempe. 2016. Robust Influence Maximization. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 885–894.

[14] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. 2012. Influence Blocking Maximization in Social Networks under the Competitive Linear Threshold Model. In Proceedings of the Twelfth SIAM SDM International Conference on Data Mining. 463–474.

[15] Kyomin Jung, Wooram Heo, and Wei Chen. 2012. IRIE: Scalable and Robust Influence Maximization in Social Networks. In Proceeding of the 12th IEEE ICDM International Conference on Data Mining. 918–923.

[16] David Kempe, Jon M. Kleinberg, and Eva Tardos. 2003. Maximizing the spread of influence through a social network. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 137–146.

[17] Elias Boutros Khalil, Bistra Dilkina, and Le Song. 2014. Scalable diffusion-aware optimization of network topology. In The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1226–1235.

[18] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. 2007. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 420–429.

[19] Wei Lu, Wei Chen, and Laks V.S. Lakshmanan. 2015. From competition to complementarity: comparative influence diffusion and maximization. Proceedings of the VLDB Endowment 9, 2 (2015), 60–71.

[20] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. 1978. An analysis of the approximations for maximizing submodular set functions. Mathematical Programming.

[21] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. 2016. Stop-and-Start: Optimal Sampling Algorithms for Viral Marketing in Billion-Scale Networks. In Proceedings of the 2016 ACM SIGKDD International Conference on Management of Data. 695–710.

[22] Matthew Richardson and Pedro Domingos. 2002. Mining knowledge-sharing sites for viral marketing. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 61–70.

[23] Lichao Sun, Xiaobin Rui, and Wei Chen. 2022. Scalable Adversarial Attack Algorithms on Influence Maximization. arXiv:2209.00892 (2022). https://doi.org/10.48550/arXiv.2209.00892.

[24] Jie Tang, Jiemeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 807–816.

[25] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online Processing Algorithms for Influence Maximization. In Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data. 75–86.

[26] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: a martingale approach. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 1539–1554.

[27] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: near-optimal time complexity meets practical efficiency. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. 75–86.

[28] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. 2007. A framework for community identification in dynamic social networks. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 717–726.

[29] Chi Wang, Wei Chen, and Yajun Wang. 2012. Scalable influence maximization for independent cascade model in large-scale social networks. Data Mining and Knowledge Discovery (DMKD). 25, 3 (2012), 545–576.