Blocking Self-avoiding Walks Stops Cyber-epidemics: A Scalable GPU-based Approach

Hung T. Nguyen, *Student Member, IEEE,* Alberto Cano, *Member, IEEE,* Tam Vu, *Member, IEEE,* and Thang N. Dinh, *Member, IEEE*

Abstract—Cyber-epidemics, the widespread of fake news or propaganda through social media, can cause devastating economic and political consequences. A common countermeasure against cyber-epidemics is to disable a small subset of suspected social connections or accounts to effectively contain the epidemics. An example is the recent shutdown of 125,000 ISIS-related Twitter accounts. Despite many proposed methods to identify such a subset, none are scalable enough to provide high-quality solutions in nowadays' billion-size networks. To this end, we investigate the Spread Interdiction problems that seek the most effective links (or nodes) for removal under the well-known Linear Threshold model. We propose novel CPU-GPU methods that *scale to networks with billions of edges*, yet possess *rigorous theoretical guarantee* on the solution quality. At the core of our methods is an $O(1)$-*space* out-of-core algorithm to generate a new type of random walks, called *Hitting Self-avoiding Walks* (HSAWs). Such a low memory requirement enables handling of big networks and, more importantly, hiding latency via scheduling of millions of threads on GPUs. Comprehensive experiments on real-world networks show that our algorithms provide much higher quality solutions and are several orders of magnitude faster than the state-of-the art. Comparing to the (single-core) CPU counterpart, our GPU implementations achieve significant speedup factors up to 177x on a single GPU and 338x on a GPU pair.

Index Terms—Spread Interdiction, Self-avoiding Walks, GPUs

1 Introduction

Cyber-epidemics have caused significant economical and political consequences, and increasingly do so in the future due to the increasing popularity of social networks. Such widespread of fake news and propaganda has the potential to pose serious threats to global security. For example, through social media, terrorists have recruited thousands of supporters who have carried terror acts including bombings in the US, Europe, killing dozens of thousands of innocents, and created worldwide anxiety [1]. The rumor of explosions at the White House injuring President Obama caused $136.5 billion loss in stock market [2] or the recent burst of fake news has significantly influenced the 2016 election [3].

To contain those cyber-epidemics, one common strategy is to disable user accounts or social connects that could potentially be vessels for rumor propagation through the "word-of-mouth" effect. For example, Twitter has deleted 125,000 accounts linked to terrorism since the middle of 2015 and U.S. officials have called for shutting down al-shababs Twitter accounts [4]. Obviously, removing too many accounts/links will negatively affect the legitimate experience, possibly hindering the freedom of speech. Thus, it is critical to identify small subsets of social links/user accounts whose removal effectively contains the epidemics.

Given a social network, which can be abstracted as a graph in which nodes represent users and edges represent their social connections, the above task is equivalent to the problem of identifying nodes and edges in the graph to remove such that it minimizes the (expected) spread of the rumors under a diffusion model. In a "blind interdiction" manner, [5], [6] investigate the problem when no information on the sources of the rumors are available. Given the infected source nodes, Kimura et al. [7] and [8] proposed heuristics to remove edges to minimize the spread from the sources. Remarkably, Khalil et al. [9] propose the first $1 - 1/e - \epsilon$-approximation algorithm for the edges removal problem under the linear threshold model [10]. However, the number of samples needed to provide the theoretical guarantee is too high for practical purpose. Nevertheless, none of the proposed methods can scale to large networks with billions of edges and nodes.

In this paper, we formulate and investigate two Spread Interdiction problems, namely *Edge-based Spread Interdiction* (eSI) and *Node-based Spread Interdiction* (nSI). The problems consider a graph, representing a social network and a subset of suspected nodes that might be infected with the rumor. They seek for a size-k set of edges (or nodes) whose removal minimize the spread from the suspected nodes under the well-known linear threshold (LT) model [10]. Our major contribution is the two hybrid GPU-based algorithms, called eSIA and nSIA, that possess distinguished characteristics:

- **Scalability**: Thanks to the highly efficient self-avoiding random walks generation on GPU(s), our algorithms run *several orders of magnitude faster* than its CPU's counterpart as well as the state-of-the-art method [9]. The proposed methods take only seconds on networks with billions of edges and can work on even bigger networks by stretching the data across multiple GPUs.

- *H.T. Nguyen, A. Cano, and T.N. Dinh are with the Department Computer Science, Virginia Commonwealth University, Richmond, VA, 23284. E-mail: {hungnt,acano,tndinh}@vcu.edu*
- *T. Vu is with the Department Computer Science, University of Colorado Boulder, Boulder, CO, 80309. E-mail: tam.vu@colorado.edu*
- *Corresponding author email tndinh@vcu.edu*

- **Rigorous quality guarantee**: Through extensive analysis, we show that our methods return $(1 - 1/e - \epsilon)$-approximation solutions w.h.p. Importantly, our methods can effectively determine a minimal number of HSAW samples to achieve the theoretical guarantee for a given $\epsilon > 0$. In practice, our solutions are consistently 10%-20% more effective than the runner up when comparing to the centrality-based, influence-maximization-based methods, and the current state of the art in [9].

The foundation of our proposed methods is a theoretical connection between Spread Interdiction and a new type of random walks, called *Hitting Self-avoiding Walks* (HSAWs). The connection allows us to find the most effective edges (nodes) for removal through finding those who appear most frequently on the HSAWs. The bottleneck of this approach is, however, the generation of HSAWs, which requires repeated generation of self-avoiding walks until one reaches a suspected node. Additionally, the standard approach to generate self-avoiding walks requires $\Omega(n)$ space per thread to store whether each node has been visited. This severely limits the number of threads that can be launched and executed concurrently.

To tackle this challenge, we propose a novel $O(1)$-*space* out-of-core algorithm to generate HSAW. Such a low memory requirement enables handling of big networks on GPU and, more importantly, hiding latency via scheduling of millions of threads. Comparing to the (single-core) CPU counterpart, our GPU implementations achieve significant speedup factors up to 177x on a single GPU and 388x on a GPU pair, making them several orders of magnitude faster than the state-of-the art method [9].

Our contributions are summarized as follows:

- We formulate the problem of stopping the cyber-epidemics by removing nodes and edges as two interdiction problems and establish an important *connection between the Spread Interdiction problems and blocking Hitting Self-avoiding Walk* (HSAW).
- We propose out-of-core $O(1) - space$ HSAW sampling algorithm that allows concurrent execution of millions of threads on GPUs. For big graphs that do not fit into a single GPU, we also provide distributed algorithms on multiple GPUS via the techniques of graph partitioning and node replicating. Our sampling algorithm might be of particular interest for those who are into sketching influence dynamics of billion-scale networks.
- Two $(1 - 1/e - \epsilon)$-approximation algorithms, namely, eSIA and nSIA, for the edge and node versions of the spread interdiction problems. Our approaches bring together rigorous theoretical guarantees and practical efficiency.
- We conduct comprehensive experiments on real-world networks with up to 1.5 billion edges. The results suggest the superiority of our methods in terms of solution quality (10%-20% improvement) and running time (2-3 orders of magnitude faster).

Organization. We present the LT model and formulate two Spread Interdiction problems on edges and nodes in Section 2. Section 3 introduces Hitting Self-avoiding Walk (HSAW) and proves the monotonicity and submodularity, followed by HSAW sampling algorithm in Section 4 with parallel and distributed implementations on GPUs. The complete approximation algorithms are presented in Section 5. Lastly, we present our experimental results in Section 6, related work in Section 7 and conclusion in Section 8. We summarize the commonly used notations in Table 1.

TABLE 1: Table of notations

Notation	Description
$\mathcal{G}, G \sim \mathcal{G}$	Given stochastic graph $\mathcal{G} = (V, E, w)$ and a sample graph G of $\mathcal{G}$
$\mathcal{V}_I$	Set of infected nodes
$\mathbb{I}_{\mathcal{G}}(S)$	Influence Spread of the set S of nodes in $\mathcal{G}$
$\mathbb{D}(T, \mathcal{V}_I)$	Influence Suspension of the edge set $T \subseteq E$
$\mathbb{D}^{(n)}(S, \mathcal{V}_I)$	Influence Suspension of the node set $S \subseteq V$
$\mathsf{OPT}_k, \mathsf{OPT}_k^{(n)}$	The maximum influence suspensions by removing k edges and nodes, respectively
$\hat{T}_k, \hat{S}_k$	The returned size-k edge set of eSIA and nSIA
T_k^*, S_k^*	An optimal size-k set of edges and nodes
$\mathcal{R}_t, \mathcal{R}'_t$	Sets of random HSAW samples in iteration t
$\mathrm{Cov}_{\mathcal{R}_t}(T)$	#HSAW $h_j \in \mathcal{R}_t$ intersecting with T
Λ	$\Lambda = (2 + \frac{2}{3}\epsilon) \ln(\frac{3}{\delta}) \frac{1}{\epsilon^2}$
$N_{\max}$	$N_{\max} = O(m \cdot \frac{\ln(1/\delta) + \ln \binom{m}{k}}{k\epsilon^2})$

2 Models and Problem Definitions

We consider a social network represented by a directed stochastic graph $\mathcal{G} = (V, E, w)$ that contains $|V| = n$ nodes and $|E| = m$ weighted edges. Each edge $(u, v) \in E$ is associated with an infection weight $w(u, v) \in [0, 1]$ which indicates the likelihood that u will infect v once u gets infected.

Assume that we observe in the network a set of suspected nodes V_I that might be infected with misinformation or viruses. However, we do not know which ones are actually infected. Instead, the probability that a node $v \in V_I$ is given by a number $p(v) \in [0, 1]$. In a social network like Twitter, this probability can be obtained through analyzing tweets' content to determine the likelihood of misinformation being spread. By the same token, in computer networks, remote scanning methods can be deployed to estimate the probability that a computer gets infected by a virus.

The Spread Interdiction problems aim at selecting a set of nodes or edges whose removal results in maximum influence suspension of the infected nodes. We assume a subset C of candidate nodes (or edges) that we can remove from the graph. The C can be determined depending on the situation at hand. For example, C can contain (highly suspicious) nodes from V_I or even nodes outside of V_I, if we wish to contains the rumor rapidly. Similarly, C can contain edges that are incident to suspected nodes in V_I or $C = E$ if we wish to maximize the effect of the containment.

We consider that the infection spreads according to the well-known *Linear Threshold (LT)* diffusion model [10].

2.1 Linear Threshold Model

In the LT model, each user v selects an activation threshold θ_v *uniformly random* from $[0, 1]$. The edges' weights must satisfy a condition that, for each node, the sum of all incoming edges' weights is at most 1, i.e., $\sum_{u \in V} w(u, v) \leq 1, \forall v \in V$. The diffusion happens in discrete time steps $t = 0, 1, 2, \ldots, n$. At time $t = 0$, a set of users $S \subseteq V$, called

the *seed set*, are infected and all other nodes are not. We also call the infected nodes *active*, and uninfected nodes *inactive*. An inactive node v at time t becomes active at time $t+1$ if $\sum_{\text{active neighbors } u \text{ of } v} w(u,v) \geq \theta_v$. The infection spreads until no more nodes become active.

Given $\mathcal{G} = (V, E, w)$ and a seed set $S \subset V$, the *influence spread* (or simply *spread*) of S, denoted by $\mathbb{I}_{\mathcal{G}}(S)$, is the expected number of infected nodes at the end of the diffusion process. Here the expectation is taken over the randomness of all thresholds θ_v.

One of the extensively studied problems is the influence maximization problem [10]. The problem asks for a seed set S of k nodes to maximize $\mathbb{I}_{\mathcal{G}}(S)$. In contrast, this paper considers the case when the seed set (or the distribution over the seed set) is given and aims at identifying a few edges/nodes whose removals effectively reduce the influence spread.

LT live-edge model. In [10], the LT model is shown to be equivalent to the *live-edge* model where each node $v \in V$ picks at most one incoming edge with a probability equal to the edge weight. Specifically, a *sample graph* $G = (V, E' \subset E)$ is generated from $\mathcal{G}$ according to the following rules: 1) for each node v, at most one incoming edge is selected; 2) the probability of selecting edge (u, v) is $w(u, v)$ and there is no incoming edge to v with probability $(1 - \sum_{u \in N^-(v)} w(u, v))$.

Then the influence spread $\mathbb{I}_{\mathcal{G}}(S)$ is equal to the expected number of nodes reachable from S in sample graph G, i.e.,

$$\mathbb{I}_{\mathcal{G}}(S) = \sum_{G \sim \mathcal{G}} \{\#\text{nodes in } G \text{ reachable from } S\} \Pr[G \sim \mathcal{G}],$$

where $G \sim \mathcal{G}$ denotes the sample graph G induced from the stochastic graph $\mathcal{G}$ according to the live-edge model.

For a sample graph G and a node $v \in V$, define

$$\chi^G(S, v) = \begin{cases} 1 & \text{if } v \text{ is reachable from } S \text{ in } G \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We can rewrite the equation for influence spread as

$$\mathbb{I}_{\mathcal{G}}(S) = \sum_{v \in V} \sum_{G \sim \mathcal{G}} \chi^G(S, v) \Pr[G \sim \mathcal{G}] = \sum_{v \in V} \mathbb{I}_{\mathcal{G}}(S, v), \quad (2)$$

where $\mathbb{I}_{\mathcal{G}}(S, v)$ denotes the probability that node $v \in V$ is eventually infected by the seed set S.

Learning Parameters from Real-world Traces. Determining the infection weights in the diffusion models is itself a hard problem and have been studied in various researches [11], [12]. In practice, this infection weight $w(u, v)$ between nodes u and v is usually estimated by the interaction frequency from u to v [10], [13] or learned from additional sources, e.g., action logs [11], [14].

2.2 Spread Interdiction in Networks

Denote by $\mathcal{V}_I = (V_I, p)$, the set of suspected nodes $\mathcal{V}_I$ and their probabilities of being the sources. $\mathcal{V}_I$ defines a probability distribution over possible seed sets. The probability of a particular seed set $X \subseteq V_I$ is given by

$$\Pr[X \sim \mathcal{V}_I] = \prod_{u \in X} p(u) \prod_{v \in V_I \setminus X} (1 - p(v)). \quad (3)$$

By considering all possible seed sets $X \sim \mathcal{V}_I$, we further define the expected influence spread of $\mathcal{V}_I$ as follows,

$$\mathbb{I}_{\mathcal{G}}(\mathcal{V}_I) = \sum_{X \sim \mathcal{V}_I} \mathbb{I}_{\mathcal{G}}(X) \Pr[X \sim \mathcal{V}_I]. \quad (4)$$

We aim to remove k nodes/edges from the network to minimize the spread of infection from the suspected nodes $\mathcal{V}_I$ (defined in Eq. 4) in the residual network. Equivalently, the main goal in our formulations is to find a subset of edges (node) T that maximize the *influence suspension* defined as,

$$\mathbb{D}(T, \mathcal{V}_I) = \mathbb{I}_{\mathcal{G}}(\mathcal{V}_I) - \mathbb{I}_{\mathcal{G}'}(\mathcal{V}_I), \quad (5)$$

where $\mathcal{G}'$ is the residual network obtained from $\mathcal{G}$ by removing edges (nodes) in S. When S is a set of nodes, all the edges adjacent to nodes in S are also removed from the $\mathcal{G}$.

We formulate the two interdiction problems as follows.

Definition 1 (Edge-based Spread Interdiction (eSI)). Given $\mathcal{G} = (V, E, w)$, $\mathcal{V}_I$, a set of suspected nodes and their probabilities of being infected $\mathcal{V}_I = (V_I, p)$, a candidate set $C \subseteq E$ and a budget $1 \leq k \leq |C|$, the eSI problem asks for a k-edge set $T^* \subseteq E$ that maximizes the influence suspension $\mathbb{D}(T_k, \mathcal{V}_I)$.

$$T^* = \arg\max_{T_k \subseteq C, |T_k| = k} \mathbb{D}(T_k, \mathcal{V}_I), \quad (6)$$

where

$$\mathbb{D}(T_k, \mathcal{V}_I) = \mathbb{I}_{\mathcal{G}}(\mathcal{V}_I) - \mathbb{I}_{\mathcal{G}'}(\mathcal{V}_I). \quad (7)$$

Definition 2 (Node-based Spread Interdiction (nSI)). Given a stochastic graph $\mathcal{G} = (V, E, w)$, a set of suspected nodes and their probabilities of being infected $\mathcal{V}_I = (V_I, p)$, a candidate set $C \subseteq V$ and a budget $1 \leq k \leq |C|$, the nSI problem asks for a k-node set S^* that maximizes the influence suspension $\mathbb{D}^{(n)}(S_k, \mathcal{V}_I)$.

$$S^* = \arg\max_{S_k \subseteq C, |S_k| = k} \mathbb{D}^{(n)}(S_k, \mathcal{V}_I), \quad (8)$$

where

$$\mathbb{D}^{(n)}(S_k, \mathcal{V}_I) = \mathbb{I}_{\mathcal{G}}(\mathcal{V}_I) - \mathbb{I}_{\mathcal{G}'}(\mathcal{V}_I). \quad (9)$$

is the influence suspension of S_k as defined in Eq. 5.

We also abbreviate $\mathbb{D}^{(n)}(S_k, \mathcal{V}_I)$ by $\mathbb{D}^{(n)}(S_k)$ and $\mathbb{D}(T_k, \mathcal{V}_I)$ by $\mathbb{D}(T_k)$ when the context is clear.

Complexity and Hardness. The hardness results of nSI and eSI problems are stated in the following theorem.

Theorem 1. nSI and eSI are NP-hard.

The proof is in the Appendix A.1. In the above definitions, the suspected nodes in $\mathcal{V}_I = (V_I, p)$ can be inferred from the frequency of suspicious behaviors or their closeness to known threats. These probabilities are also affected by the seriousness of the threats.

Extension to Cost-aware Model. One can generalize the nSI and eSI problems to replace the set of candidate C with an assignment of removal costs for edges (nodes). This can be done by incorporating the cost-aware version of the max-coverage problem in [15], [16]. For the sake of clarity, we, however, opt for the uniform cost version in this paper.

Comparison with Edge Deletion problem in [9] Different from the recent work in [9] where they select k edges to maximize the **sum of influence suspensions** of all the nodes in $\mathcal{V}_I$, our eSI problem considers $\mathcal{V}_I$ as a whole and maximize the influence suspension of $\mathcal{V}_I$. The formulation in [9] reflects the situation that only a *single node* in $\mathcal{V}_I$ will continue spreading the epidemics and every node has the same chance of being that spreading node. In contrast, eSI considers a more practical situation in which there can be

multiple spreading nodes and every node $v \in \mathcal{V}_I$ can be a spreading node with probability $p(v)$. That is the common case in cyber-epidemics when a node gets infected with misinformation or viruses, it will subsequently spread that to its neighbors.

2.3 Interdictions via Naive Graph Sampling Approach

A first-to-try approach for addressing influence-related, e.g. influence suspension, optimization problems is the common naive Greedy algorithm that consists of two steps: 1) generating many sample graphs to approximate the influence suspension function and, 2) applying standard Greedy algorithm, that iteratively select nodes with highest influence suspension approximates, to find a solution. However, this approach incurs excessive computational cost in order to provide rigorously good solution guarantees and thus, is not scalable for networks in practice. This high computational burden stems from three factors:

- The graph sampling simulation is very expensive, i.e. $O(|V| + |E|)$ - linear to size of the graph.
- Computing the influence suspension is #P-hard, i.e. inherited from the #P-hardness of influence computation [17] and hence, approximating influence suspension accurately, i.e. within an error of ϵ with probability $1-\delta$, needs many sample graphs, i.e. $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta})|V|)$ [18].
- To select a node, the Greedy algorithm needs to approximate influence suspension of all the nodes that have not been selected, i.e. $O(|V|)$.

Thus, to find a provably good solution for influence suspension, the time complexity is at least $O(k(|V| + |E|)|V|^2\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ which is cubic in network size and cannot run for large networks.

In fact, the recent work [9], following this graph sampling approach, has to heuristically fix a number of sample graphs, e.g. 500 in [9], which introduces an unknown error factor of α [9], depending on the quality of influence suspension estimation by the fixed number of sample graphs, in their approximation guarantee.

3 Spread Interdictions via Blocking Hitting Self-avoiding Walks (HSAWs)

In this section, we first introduce a new type of *Self-avoiding Walk* (SAW), called *Hitting SAW* (HSAW), under the LT model. These HSAWs and how to generate them are keys of our proofs and algorithms. Specifically, we prove that the Spread Interdiction problems are equivalent to identifying the sets of "most frequent edges/nodes" among a collection of HSAWs.

3.1 HSAW Definition and Properties

First, we define *Hitting Self-avoiding Walk* (HSAW) for a sample graph G of $\mathcal{G}$.

Definition 3 (Hitting Self-avoiding Walk (HSAW)). Given a sample graph $G = (V, E)$ of a stochastic graph $\mathcal{G} = (V, E, w)$ under the live-edge model (for LT), a sample set $X \subseteq \mathcal{V}_I$, a walk $h =< v_1, v_2, \ldots, v_l >$ is called a hitting self-avoiding walk if $\forall i \in [1, l-1], (v_{i+1}, v_i) \in E, \forall i, j \in [1, l], v_i \neq v_j$ and $h \cap X = \{v_l\}$.

An HSAW h starts from a node v_1, called the source of h and denoted by $\mathsf{src}(h)$, and consecutively walks to an incoming neighboring node without visiting any node more than once. From the definition, the distribution of HSAWs depends on the distribution of the sample graphs G, drawn from $\mathcal{G}$ following the live-edge model, the distributions of the infection sources $\mathcal{V}_I$, and $\mathsf{src}(h)$.

HSAW Properties. According to the live-edge model (for LT), each node has at most one incoming edge. This leads to three important properties.

Self-avoiding: An HSAW has no duplicated nodes. Otherwise, there is a loop in h and at least one of the node on the loop (that cannot be v_l) will have at least two incoming edges, contradicting the live-edge model for LT.

Walk Uniqueness: Given a sample graph $G \sim \mathcal{G}$ and $X \sim \mathcal{V}_I$, for any node $v \in V \backslash X$, there is at most one HSAW h that starts at node v. To see this, we can trace from v until reaching a node in X. As there is at most one incoming edge per node, the trace is unique.

Walk Probability: Given a stochastic graph $\mathcal{G} = (V, E, w)$ and $\mathcal{V}_I$, the probability of having a particular HSAW $h =< v_1, v_2, \ldots, v_l >$, where $v_l \in \mathcal{V}_I$, is computed as follows,

$$\begin{aligned}\Pr[h \in \mathcal{G}] &= p(v_l) \prod_{u \in V_I \cap h, u \neq v_l} (1 - p(u)) \sum_{G \sim \mathcal{G}} \Pr[G \sim \mathcal{G}] \cdot \mathbf{1}_{h \in G} \\ &= p(v_l) \prod_{u \in V_I \cap h, u \neq v_l} (1 - p(u)) \prod_{i=1}^{l-1} w(v_{i+1}, v_i), \qquad (10)\end{aligned}$$

where $h \in \mathcal{G}$ if all the edges in h appear in a random sample $G \sim \mathcal{G}$.

Thus, based on the properties of HSAW, we can define a probability space Ω_h which has the set of elements being all possible HSAWs and the probability of a HSAW computed from Eq. 10.

3.2 Spread Interdiction $\leftrightarrow$ HSAW Blocking

From the probability space of HSAW in a stochastic network $\mathcal{G}$ and set $\mathcal{V}_I$ of infected nodes, we prove the following important connection between influence suspension of a set of edges and a random HSAW. We say $T \subseteq E$ *interdicts* an HSAW h_j if $T \cap h_j \neq \emptyset$. When T interdicts h_j, removing T will disrupt h_j, leaving $\mathsf{src}(h_j)$ uninfected.

Theorem 2. Given a graph $\mathcal{G} = (V, E, w)$ and a set $\mathcal{V}_I$, for any random HSAW h_j and any set $T \in E$ of edges, we have

$$\mathbb{D}(T, \mathcal{V}_I) = \mathbb{I}_{\mathcal{G}}(\mathcal{V}_I) \Pr[T \text{ interdicts } h_j]. \qquad (11)$$

The proof is presented in Appendix A.2. Theorem 2 states that the influence suspension of a set $T \subseteq E$ is proportional to the probability that T intersects with a random HSAW. Thus, to maximize the influence suspension, we find a set of edges that hits the most HSAWs. This motivates our sampling approach:

(1) Sample Γ random HSAWs to build an estimator the influence suspensions of many edge sets,
(2) Apply Greedy algorithm over the set of HSAW samples to find a solution $\hat{T}_k$ that blocks the most HSAW samples.

The challenges in this approach are how to efficiently generate random HSAWs and what the value of Γ is to provide guarantee.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2904969, IEEE Transactions on Knowledge and Data Engineering

As a corollary of Theorem 2, we obtain the monotonicity and submodularity of the influence suspension function $\mathbb{D}(T_k, \mathcal{V}_I)$.

Corollary 1. The influence suspension function $\mathbb{D}(T)$ where T is the set of edges, under the LT model is monotone,

$$\forall T \subseteq T', \mathbb{D}(T) \leq \mathbb{D}(T'), \tag{12}$$

and submodular, i.e. for any $(u, v) \notin T'$,

$$\mathbb{D}(T \cup \{v\}) - \mathbb{D}(T) \geq \mathbb{D}(T' \cup \{(u, v)\}) - \mathbb{D}(T'). \tag{13}$$

The proof is presented in our appendix. The monotonicity and submodularity indicate that the above greedy approach will return $(1 - 1/e - \epsilon)$ approximation solutions, where $\epsilon > 0$ depends on the number of generated HSAW. To provide a good guarantee, a large number of HSAW are needed, making generating HSAW the bottleneck of this approach. However, unlike expensive sample graph generation which requires storing the whole graph in every simulation, we propose a constant-space algorithm to generate HSAWs which makes it perfectly suited on GPU platform for the massive generation of HSAW samples.

4 Scalable HSAW Sampling Algorithm

We propose our sampling algorithm to generate HSAW samples on massive parallel GPU platform. We begin with the simple CPU-based version of the algorithm.

Algorithm 1: HSAW Sampling Algorithm

Input: Graph $\mathcal{G}$, suspect set $\mathcal{V}_I$ and $p(v), \forall v \in \mathcal{V}_I$
Output: A random HSAW sample h_j

1 **while** True **do**
2 Pick a node v uniformly at random;
3 Initialize $h_j = \emptyset$;
4 **while** True **do**
5 $h_j = h_j \cup \{(u, v)\}$ ($h_j = h_j \cup \{u\}$ for node version);
6 Use *live-edge* model to select an edge $(u, v) \in E$;
7 **if** *no edge selected* **then**
8 **break**;
9 **if** *edge* (u, v) *is selected* **then**
10 **if** $u \in V_I$ **and** *rand*() $\leq p(u)$ **then**
11 **return** h_j;
12 **if** $u \in h_j$ **then**
13 **break**;
14 Set $v = u$;

4.1 CPU-based Algorithm to Generate HSAWs

Alg. 1 describes our HSAW sampling procedure which is based on the *live-edge* model in Section 2.

The algorithm follows a *rejection sampling scheme* which repeatedly generate random SAW (Lines 2-14) until getting a HSAW (Line 1). The SAW sampling picks a random node v and follows the live-edge model to select an incoming edge to (u, v) (Lines 5-14). Then it replaces v with u and repeat the process until either: 1) no live-edge is selected (Lines 7-8) indicating that h_j does not reach to an infected node; 2) h_j hits to a node in $\mathcal{V}_I$ and that node is actually an infected node (Lines 10-11) or 3) edge (u, v) is selected but u closes a cycle in h_j. Only in the second case, the algorithm terminates and return the found HSAW.

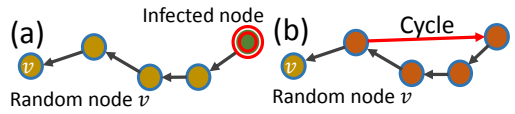


Fig. 1: (a) no cycle, HSAW found; (b) a single cycle, no HSAW

The algorithm is illustrated in Fig. 1. In (**a**), the simple path travels through several nodes and reaches an infected node. In (**b**), the algorithm detects a cycle.

Algorithm 2: ThreadSample - Sampling on GPU thread

Input: l and *ThreadID*
Global pool H of HSAW samples;
Initialize a generator PRG.*Seed* $\leftarrow$ PRG2(*ThreadID*);
for i = *1* **to** *8* **do**
  PRG.*next()*; // Burn-in period
for i = *1* **to** l **do**
  $Seed_h$ = PRG.*next*(); $Len_h = 0$;
  Use PRG to pick a node v uniformly at random;
  while True **do**
    Use PRG to select an edge $(u, v) \in E$ following the *live-edge* LT model;
    if *no edge selected* **or** $Len_h \geq n$ **then**
      break;
    if *edge* (u, v) *is selected* **then**
      if $u \in V_I$ **then**
        Use PRG with probability $p(u)$:
          $H = H \cup < Seed_h, Len_h + 1 >$;
          break;
      if *cycle detected at* u **then**
        break;
      Set $v = u$; $Len_h = Len_h + 1$;

Algorithm 3: GPU Parallel HSAW Sampling Algorithm

Input: Graph $\mathcal{G}$, $\mathcal{V}_I$, $p(v), \forall v \in \mathcal{V}_I$
Output: $\mathcal{R}$ - A stream of HSAW samples

$i = 0$;
while True **do**
  Initialize global $H = \emptyset, l = 10$; $\text{thread}_{\max}$ depends on GPU model;
  Call ThreadSample(*ThreadID*, l) $\forall ThreadID = 1..\text{thread}_{\max}$;
  foreach $< Seed_h, Len_h > \in H$ **do**
    Reconstruct h from $Seed_h, Len_h$;
    if h *has no cycle* **then**
      $i \leftarrow i + 1$;
      Add $R_i = \{\text{edges in } h\}$ to stream $\mathcal{R}$;

4.2 Parallel HSAW Generation on GPU(s)

GPUs with the massive parallel computing power offer an attractive solution for generating HSAW, the major bottleneck. As shown in the previous subsection, generating a HSAW requires repeatedly generating SAWs. Since the SAW samples are independent, if we can run millions of SAW generation threads on GPUs, we can maximize the utility of GPUs' cores and minimize the stalls due to pipelines hazard or memory accesses, i.e., minimize *latency hiding*. Moreover,

only the hitting SAW need to be transported back to CPU, thus the GPU-CPU communication is minimal.

Challenges. Due to the special design of GPU with a massive number of parallel threads, in the ideal case, we can speed up our algorithms vastly if memory accesses are *coalesced* and there is no *warp divergence*. However, designing such algorithms to fully utilize GPUs requires attention to the GPU architecture [19], [20].

Moreover, executing millions of parallel threads means each thread has little memory to use. Unfortunately, the CPU-based Algorithm to generate HSAW(Alg. 1) can use up to $\Omega(n)$ space to track which nodes have been visited. For large networks, there is not enough memory to launch a large number of threads.

We tackle the above challenges and design a new lightweight HSAW generation algorithm. Our algorithm, presented in Alg. 2, requires only $O(1)$ space per thread. Thus, millions of threads can be invoked concurrently to maximize the efficiency. The algorithm ThreadSample in Alg. 2 consists of three efficient techniques: $O(1)$-space Path-encoding, $O(1)$-space Infinite Cycle Detection and Sliding Window Early Termination to generate HSAW.

4.2.1 Memory-efficient Path-encoding

The first technique $O(1)$-space Path-encoding aims at generating SAW samples on GPU cores using only constant memory space. We take advantage of a typical feature of modern pseudo-random number generators that a random number is generated by a function with the input (seed) being the random number generated in the previous round,

$$r_i = f(r_{i-1}) \qquad (i \geq 1) \tag{14}$$

where r_0 is the initial seed that can be set by users. Those generators are based on linear recurrences and proven in [21] to be extremely fast and passing strong statistical tests.

Thus, if we know the value of the random seed at the beginning of the SAW generator and the number of traversal steps, we can reconstruct the whole walks. As a result, the SAW sampling algorithm only needs to store the set of initial random seeds and the walk lengths. The Alg. 2 is similar to Alg. 1 except it does not return a SAW but only two numbers $Seed_h$ and Len_h that encode the walk.

To detect cycle (line 17), ThreadSample use the following two heuristics to detect most of the cycles. As the two heuristics can produce false negative (but not false positive), there is a small chance that ThreadSample will return some walks with cycles. However, the final checking of cycle in Alg. 3 will make sure only valid HSAW will be returned.

4.2.2 Infinite Cycle Detection

To detect cycle in SAW sampling (line 17 in Alg. 2), we adopt two constant space Cycle-detection algorithms: the Floyd's [22] and Brent's algorithms [22].

The Floyd's algorithm only requires space for two pointers to track a generating path. These pointers move at different speeds, i.e., one is twice as fast as the other. Floyd's guarantees to detect the cycle in the first traversing round of the slower pointer on the cycle and in the second round of the faster one. The Floyd's algorithm maintains two sampling paths pointed by two pointers and thus, needs two identical streams of random live-edge selections.

Differently, Brent's algorithm cuts half of the computation of Floyd's algorithm by requiring a single stream of live-edge selections. The algorithm compares the node at position $2^{i-1}, i \geq 1$ with each subsequent sequence value up to the next power of two and stops when it finds a match. In our experiments, this algorithm is up to 40% faster than Floyd's. Overall, both of the algorithms only need memory for two pointers and have the complexity of $O(|h_j|)$ - size of the generated HSAW. The Brent's algorithm combined with cycle detection results in a speedup factor of 51x in average compared to that on a single CPU core.

4.2.3 Short Cycle Detection with Cuckoo Filter

Many cycles, if they exist, often have a small size. Thus, we use Cuckoo filter [23] of a small fixed size k to index and detect the cycle among the last k visited nodes. Our experimental results (with $k = 2$) show that this short cycle detection improves further other acceleration techniques to a speedup factor of 139x.

4.2.4 Combined Algorithm

The combined algorithm of generating HSAW on GPU is presented in Alg. 3 which generates a stream of HSAW samples $h_1, h_2, \ldots$. The main component is a loop of multiple iterations. Each iteration calls $\mathsf{thread}_{\max}$, i.e. maximum number of threads in the GPU used, threads to execute Alg. 2 that runs on a GPU core and generates at most l HSAW samples. Those samples are encoded by only two numbers $Seed_h$ and Len_h which denotes the starting seed of the random number generator and the length of that HSAW. Based on these two numbers, we can reconstruct the whole HSAW and recheck the occurrence of a cycle. If no cycles detected, a new HSAW $R_i = \{\text{edges in } h\}$ is added to the stream. The small parameter l prevents thread divergence.

Recall that Alg. 2 is similar to that of Alg. 1 except:

1) It only stores three numbers: node v, $Seed_h$ and Len_h.
2) It uses two random number generator PRG and PRG2 which are in the same class of linear recurrence (Eq. 14). PRG goes through the burn-in period to guarantee the randomness (Lines 3-4).
3) Cycle detection in Line 17 can be Floyd's, Brent's or one with Cuckoo Filter (this requires rechecking in Alg. 3).

Thus, the algorithms require only a constant space and has the same time complexity as HSAW sampling in Alg. 1. An illustration of our sampling algorithm on one GPU core is presented in Fig. 2.

4.3 Distributed Algorithm on Multiple GPUs

In case the graph cannot fit into the memory of a single GPU, we will need to distribute the graph data across multiple GPUs. We refer to this approach as *Distributed algorithm on Multiple GPUs*.

We use the folklore approach of partitioning the graph into smaller (potentially overlapping) partitions. Ideally, we aim at partitioning the graph that minimizes the inter-GPU communication. This is equivalent to minimizing the chance of a HSAW crossing different partitions. To do this, we first apply the standardized METIS [24] graph partitioning techniques into p partitions where p is the number of GPUs. Each GPU will then receive a partition and generate samples from that subgraph. The number of samples generated by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2904969, IEEE Transactions on Knowledge and Data Engineering

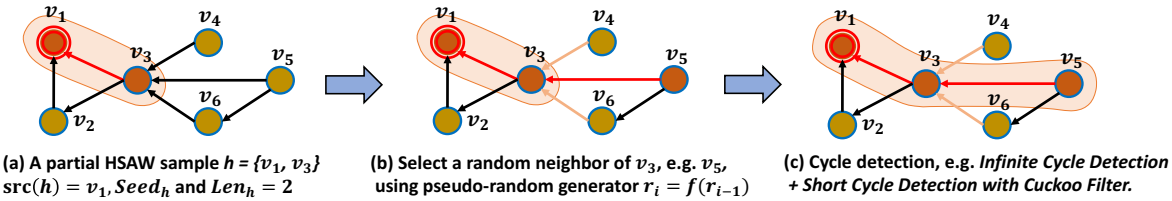


Fig. 2: An illustration of HSAW() sampling algorithm in a single GPU core: **(a)** A partial HSAW sample $h = \{v_1, v_3\}$ is underway that requires only three constants, i.e., $\mathsf{src}(h), Seed_h, Len_h$, to store; **(b)** A random in-neighbor of the last visited node v_3 is selected using the pseudo-random generator function $r_f = f(f_{i-1})$, e.g., assuming that among three in-neighbors v_4, v_5, v_6 of v_3, v_5 is selected; **(c)** After a neighbor is randomly sampled, cycle detection algorithms are revoked to check whether the sampled neighbor forms any cycle, e.g., here, v_5 does not form any cycle with previously visited nodes in the walk and, thus, the sampling algorithm continues repeating the last two steps **(b)** and **(c)** from v_5.

each GPU is proportional to the number of nodes in the received partition. We further reduce the crossing HSAW by extending each partition to include nodes that are few hop(s) away. The number of hops away is called *extension parameter*, denoted by h. We use $h = 1$ and $h = 2$ in our experiments.

5 Approximation Algorithms

This section focuses on detecting the minimal number of HSAW samples to find an $(1-1/e-\epsilon)$ approximate solution. We adopt the recent Stop-and-Stare framework from [25] and present a concise stopping condition based on stopping time from martingale theory [26] that fixes the technical error[1] in the original paper [25] as pointed out in [28].

Algorithm 4: Greedy algorithm

Input: A set $\mathcal{R}_t$ of HSAW samples, candidate set C and k.
Output: An $(1-1/e)$-optimal solution $\hat{T}_k$ on samples.

$\hat{T}_k = \emptyset$;
for $i = 1$ **to** k **do**
  $\hat{e} \leftarrow \arg\max_{e \in C \setminus \hat{T}_k} (\mathrm{Cov}_{\mathcal{R}_t}(\hat{T}_k \cup \{e\}) - \mathrm{Cov}_{\mathcal{R}_t}(\hat{T}_k))$;
  Add $\hat{e}$ to $\hat{T}_k$;
return $\hat{T}_k$;

Algorithm 5: Check algorithm

Input: $\hat{T}_k, \mathcal{R}_t, \mathcal{R}'_t, \epsilon, \delta$ and t.
Output: True if the solution $\hat{T}_k$ meets the requirement.

$\epsilon_1 = \mathrm{Cov}_{\mathcal{R}_t}(\hat{T}_k)/\mathrm{Cov}_{\mathcal{R}'_t}(\hat{T}_k) - 1$;
Compute error bounds ϵ_2 (Eq 16) and ϵ_3 (Eq. 17);
$\epsilon_t = (\epsilon_1 + \epsilon_2 + \epsilon_1\epsilon_2)(1 - 1/e - \epsilon) + (1 - 1/e)\epsilon_3$;
if $\epsilon_t \leq \epsilon$ **then**
  return True;
return False;

5.1 Edge-based Spread Interdiction Algorithm

Stop-and-Stare framework and subtle error in [25]. The main idea is to use a second set of samples to verify the solution found by the greedy algorithm on the first set of samples. The verification is performed by estimating the quality of the test solution and the optimal with high probability, then, a worst-case error bound is computed and compared with the desired approximation error. However, this verification creates a dependency between the stopping time and the correctness of the returned solution at that time. Thus, the standard concentration bounds on independent identically distributed random variables are not applicable. This was the mistake of [25] in proving the approximation guarantee of the Stop-and-Stare algorithms as mentioned in [28].

1. The errors in [25] are fixed in [27]. Our proof is simpler and tighter thanks to the better bounds inheriting from martingale stopping time.

Algorithm 6: Edge Spread Interdiction Alg. (eSIA)

Input: Graph $\mathcal{G}$, $\mathcal{V}_I$, $p(v), \forall v \in \mathcal{V}_I$, k, $C \subseteq E$ and $0 \leq \epsilon, \delta \leq 1$.
Output: $\hat{T}_k$ - An $(1 - 1/e - \epsilon)$-near-optimal solution.

Compute Λ (Eq. 18), $N_{\max}$ (Eq. 15); $t = 0$;
A stream of HSAW $h_1, h_2, \ldots$ is generated by Alg. 3 on GPU;
repeat
  $t = t + 1$; $\mathcal{R}_t = \{R_1, \ldots, R_{\Lambda 2^{t-1}}\}$; $\mathcal{R}'_t = \{R_{\Lambda 2^{t-1}+1}, \ldots, R_{\Lambda 2^t}\}$;
  $\hat{T}_k \leftarrow \mathsf{Greedy}(\mathcal{R}_t, C, k)$;
  if $\mathsf{Check}(\hat{T}_k, \mathcal{R}_t, \mathcal{R}'_t, \epsilon, \delta) =$ True **then**
    return $\hat{T}_k$;
until $|\mathcal{R}_t| \geq N_{\max}$;
return $\hat{T}_k$;

Martingale theory and stopping time. Fortunately, the sequence of random variables constructed from the samples can be shown to form a martingale with stopping time being the point that the algorithm satisfies the checking condition and terminates. Hence, the following concentration inequality [26] applies:

Lemma 1 ([26]). Let X be a martingale satisfying

- $\mathrm{Var}[X_i | X_{i-1}, \ldots, X_1] \leq \sigma_i^2$, for $1 \leq i \leq l$,
- $|X_i - X_{i-1}| \leq M$, for $1 \leq i \leq l$.

Then, we have

$$\Pr\left[X - \mathbb{E}[X] \geq \lambda\right] \leq \exp\left(\frac{-\lambda^2}{2(\sum_{i=1}^{l} \sigma_i^2 + M\lambda/3)}\right)$$

To apply the above bound, we need a maximum value l on the stopping time of the algorithm. In our cases of edge-based spread interdiction problem using HSAW samples, similarly to [13], [25], [29], we first derive a maximum number of HSAW samples, denoted by $N_{\max}$, such that the

greedy algorithm (Alg. 4) on $N_{\max}$ random HSAW samples finds an $(1-1/e-\epsilon)$-approximate solution w.h.p.

$$N_{\max} = O\left(m \cdot \frac{\ln(1/\delta) + \ln\binom{m}{k}}{k\epsilon^2}\right). \tag{15}$$

Note that $\frac{k}{m} \leq \frac{\mathsf{OPT}_k}{\mathbb{I}_{\mathcal{G}}(\mathcal{V}_I)}$ since the optimal solution OPT_k with k edges must cover at least a fraction $\frac{k}{m}$ the total influence of $\mathbb{I}_{\mathcal{G}}(\mathcal{V}_I)$.

We obtain the following lemma similar to Theorem 1 in [13].

Lemma 2. Let R_t be a set of $N_{\max}$ random HSAW samples and $\hat{T}_k$ be the set of edges selected by greedy algorithm (Alg. 4). We have

$$\Pr\left[|R_t| = N_{\max} \wedge \mathbb{D}(\hat{T}_k) \leq (1-1/e-\epsilon)\mathit{OPT}_k\right] \leq 1 - \frac{\delta}{3}$$

eSIA Algorithm. The complete algorithm eSIA is presented in Alg. 6 which uses the following procedures:

- Greedy (Alg. 4) selects a candidate solution $\hat{T}_k$ from a set of HSAW samples $\mathcal{R}_t$. This implements the greedy scheme that iteratively selects from E an edge that maximizes the marginal gain. The algorithm stops after selecting k edges.
- Check (Alg. 5) verifies if the candidate solution $\hat{T}_k$ satisfies the given precision error ϵ. It computes an error bound, i.e. ϵ_t, for the candidate solution $\hat{T}_k$ through $\epsilon_1, \epsilon_2, \epsilon_3$ (Lines 4-6), and compares that with the input ϵ. ϵ_1 measures the discrepancy of estimating $\hat{T}_k$ influence suspension using two set of HSAW samples $\mathcal{R}_t$ and RIS' while the computations of ϵ_2, ϵ_3 reflect the estimation errors for estimating $\mathbb{D}(\hat{T}_k)$ on $\mathcal{R}'_t$ and the optimal solution of T^*_k on $\mathcal{R}_t$. Let $\alpha = \frac{1}{3} - \frac{N_{\max}}{N}, \beta = \log(3/\delta), \alpha' = \frac{1}{3} + \frac{N_{\max}}{N}$ and $\beta' = \log(\frac{3\binom{n}{k}}{\delta})$,

$$\epsilon_2 = \frac{\frac{\beta}{N}\left(\alpha + \sqrt{\alpha^2 + \frac{2\hat{\mu}'_{\hat{T}_k} N_{\max}}{\beta}}\right)}{\mu'_{\hat{T}_k} - \frac{\beta}{N}\left(\alpha + \sqrt{\alpha^2 + \frac{2\hat{\mu}'_{\hat{T}_k} N_{\max}}{\beta}}\right)}. \tag{16}$$

$$\epsilon_3 = \frac{\frac{\beta'}{N}\left(\alpha' + \sqrt{\alpha'^2 + \frac{2\hat{\mu}_{\hat{T}_k} N_{\max}}{(1-1/e)\beta'}}\right)}{\frac{\mu_{\hat{T}_k}}{1-1/e} + \frac{\beta'}{N}\left(\alpha' + \sqrt{\alpha'^2 + \frac{2\hat{\mu}_{\hat{T}_k} N_{\max}}{(1-1/e)\beta'}}\right)}. \tag{17}$$

These error parameters are derived from Lemma 1 (See Appendix A.3 for details).

Main Algorithm eSIA (Alg. 6) first computes the upper-bound on the necessary HSAW samples $N_{\max}$ (Eq. 15) and

$$\Lambda = (2 + \frac{2}{3}\epsilon)\ln(\frac{3}{\delta})\frac{1}{\epsilon^2}, \tag{18}$$

Then, it enters a loop of iterations. In iteration $t \geq 1$, eSIA uses the set $\mathcal{R}_t$ of first $\Lambda 2^{t-1}$ HSAW samples to find a candidate solution $\hat{T}_k$ by the Greedy algorithm (Alg. 4). Afterwards, it checks the quality of $\hat{T}_k$ by the Check procedure (Alg. 5) using the second set of $\Lambda 2^{t-1}$ HSAW samples. If the Check returns True, i.e. $\epsilon_t \leq \epsilon$, $\hat{T}_k$ is returned as the final solution. Otherwise, the Check algorithm fails to verify the candidate solution $\hat{T}_k$, then, eSIA will be terminated by the guarding condition $|\mathcal{R}_t| \geq N_{\max}$ (Line 9).

Approximation Guarantee Analysis. We prove that eSIA returns an $(1-1/e-\epsilon)$-approximate solution for the eSI problem with probability at least $1-\delta$ where $\epsilon \in (0,1)$ and $1/\delta = O(n)$ are the given precision parameters.

Theorem 3. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, a probabilistic set $\mathcal{V}_I$ of suspected nodes, candidate edge set $C \subseteq E$, $0 < \epsilon < 1$, $1/\delta = O(n)$ as the precision parameters and budget k, ***eSIA*** returns an $(1-1/e-\epsilon)$-approximate solution $\hat{T}_k$ with probability at least $1-\delta$,

$$\Pr[\mathbb{D}(\hat{T}_k) \geq (1-1/e-\epsilon)\mathit{OPT}_k] \geq 1-\delta. \tag{19}$$

The proof is presented in Appendix A.3.

5.2 Node-based Spread Interdiction Algorithm

Similar to Theorem 2, we can also establish the connection between identifying nodes for removal and identifying nodes that appear frequently in HSAWs.

Theorem 4. Given $\mathcal{G} = (V, E, w)$, a random HSAW sample h_j and a probabilistic set $\mathcal{V}_I$, for any set $S \in V$,

$$\mathbb{D}^{(n)}(S, \mathcal{V}_I) = \mathbb{I}_{\mathcal{G}}(\mathcal{V}_I)\Pr[S \text{ interdicts } h_j].$$

Thus, the nSIA algorithm for selecting k nodes to remove and maximize the influence suspension is similar to eSIA except:

1) The Greedy algorithm selects nodes with maximum marginal gains into the candidate solution $\hat{S}_k$.
2) The maximum HSAW samples is computed as follows,

$$N^{(n)}_{\max} = O\left(n \cdot \frac{\ln(1/\delta) + \ln\binom{n}{k}}{k\epsilon^2}\right). \tag{20}$$

The approximation guarantee is stated below.

Theorem 5. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$, a probabilistic set $\mathcal{V}_I$ of possible seeds with their probabilities, $C \subseteq V$, $0 < \epsilon < 1$, $1/\delta = O(n)$ and a budget k, ***nSIA*** returns an $(1-1/e-\epsilon)$-approximate solution $\hat{S}_k$ with probability at least $1-\delta$,

$$\Pr\left[\mathbb{D}^{(n)}(\hat{S}_k) \geq (1-1/e-\epsilon)\mathit{OPT}^{(n)}_k\right] \geq 1-\delta, \tag{21}$$

where S^*_k is an optimal solution of k nodes.

6 Experimental Evaluation

In this section, we present the results of our comprehensive experiments on real-world networks. The results suggest the superiority of nSIA and eSIA over the other methods.

6.1 Experimental Settings

Algorithms compared. For each of the studied problems, i.e., nSI and eSI, we compare three sets of algorithms:

- nSIA and eSIA - our proposed algorithms, each of which has five implementations: single/multi-core CPU, and single/parallel/distributed GPU accelerations.
- InfMax-V and InfMax-V_I - algorithms for Influence Maximization problem, that finds the set of k nodes in C that have the highest influence. For the edge version,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2904969, IEEE Transactions on Knowledge and Data Engineering

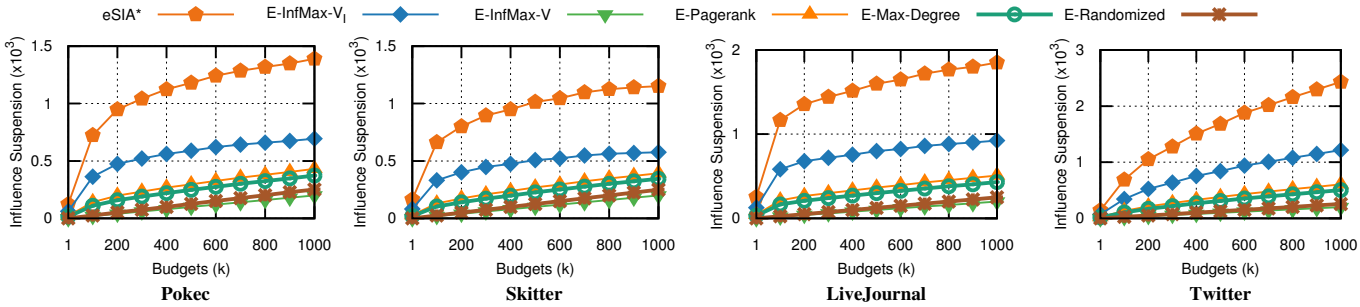


Fig. 3: Interdiction Efficiency of different approaches on eSI problem (eSIA* denotes the general eSIA)

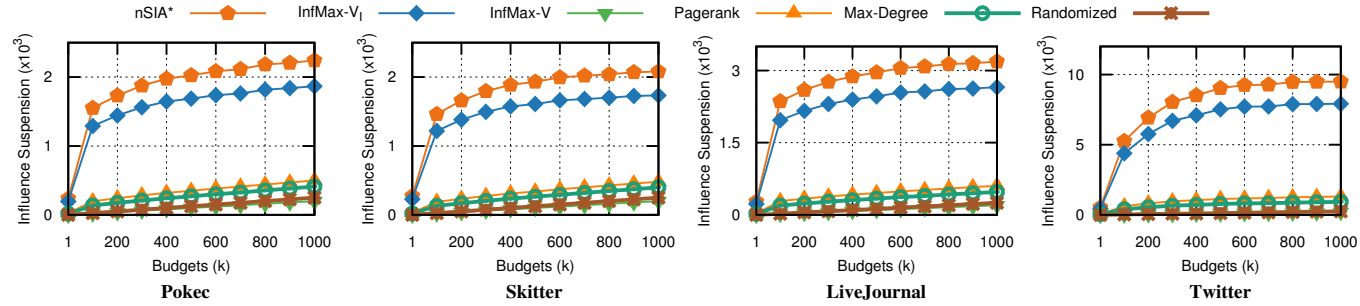


Fig. 4: Interdiction Efficiency of different approaches on nSI problem (nSIA* refers to general nSIA algorithm)

we follow [9] to select k edges that go *into* the highest influence nodes.

- GreedyCutting [9] on edge deletion problem.
- Baseline methods: we consider three common ranking measures: Pagerank, Max-Degree and Randomized.

Datasets. Table 2 provides the summary of the datasets.

TABLE 2: Datasets Statistics

Dataset	#Nodes (n)	#Edges (m)	Avg. Deg.	Type
DBLP(*)	655K	2M	6.1	Co-author
Pokec(*)	1.6M	30.6M	19.1	Social
Skitter(*)	1.7M	11.1M	6.5	Internet
LiveJournal(*)	4M	34.7M	8.7	Social
Twitter [30]	41.7M	1.5G	70.5	Social

(*)http://snap.stanford.edu/data/index.html

Measurements. We measure the performance of each algorithm in two aspects: Solution quality and Scalability. To compute the influence suspension, we adapt the EIVA algorithm in [25] to find an (ϵ, δ)-estimate $\hat{\mathbb{D}}(T, \mathcal{V}_I)$,

$$\Pr[|\hat{\mathbb{D}}(T, \mathcal{V}_I) - \mathbb{D}(T, \mathcal{V}_I)| \geq \epsilon \mathbb{D}(T, \mathcal{V}_I)] \leq \delta, \qquad (22)$$

where ϵ, δ are set to 0.01 and $1/n$ (see details in [25]).

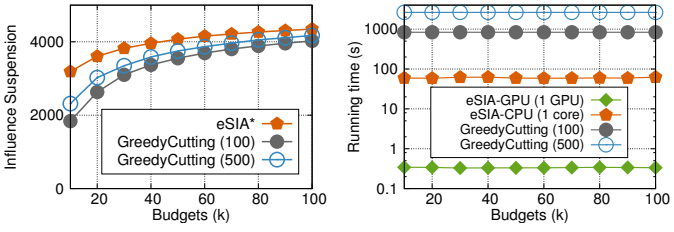
Parameter Settings. We follow a common setting in [13], [16], [25] and set the weight of edge (u, v) to be $w(u, v) = \frac{1}{d_{in}(v)}$. Here $d_{in}(v)$ denotes the in-degree of node v. For simplicity, we set $C = E$ (or $C = V$) in the edge (or node) interdiction problem(s). For nSIA and eSIA algorithms, we set the precision parameters to be $\epsilon = 0.1$ and $\delta = 1/n$ as a general setting. Following the approach in [9] the suspected set of nodes $\mathcal{V}_I$ contains randomly selected 1000 nodes with randomized probabilities between 0 and 1 of being real suspects. The budget value k is ranging from 100 to 1000.

All the experiments are carried on a CentOS 7 machine having 2 Intel(R) Xeon(R) CPUs X5680 3.33GHz with 6 cores each, 2 NVIDIA's Titan X GPUs and 100 GB of RAM. The algorithms are implemented in C++ with C++11 compiler and CUDA 8.0 toolkit.

6.2 Solution Quality

The results of comparing the solution quality, i.e., influence suspension, of the algorithms on four larger network datasets, e.g., Pokec, Skitter, LiveJournal and Twitter, are presented in Fig. 3 for eSI. Across all four datasets, we observe that eSIA significantly outperforms the other methods with widening margins when k increases. eSIA performs twice as good as InfMax-V_I and many times better than the rest.

Experiments on nSIA give a similar observation. The results of comparing the solution quality, i.e., influence suspension, of the algorithms on the four larger network datasets, e.g., Pokec, Skitter, LiveJournal and Twitter, are presented in Fig. 4 for nSI. Across all four datasets, we observe that nSIA significantly outperforms the other methods with widening margins when k increases. For example, on the largest Twitter network, nSIA is about 20% better than the runner-up InfMax-V_I and 10 times better than the rest.



(a) Influence Suspension **(b)** Running time

Fig. 6: Comparison between eSIA and GreedyCutting with Edge Deletion Problem on Skitter network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2904969, IEEE Transactions on Knowledge and Data Engineering

Dataset	1 CPU core		8 CPU cores		1 GPU		par-2 GPUs	
	time(s)	SpF	time(s)	SpF	time(s)	SpF	time(s)	SpF
DBLP	10.3	1	1.7	6.0	0.1	103	0.05	206
Pokec	36.6	1	5.6	6.5	0.3	122	0.2	183
Skitter	34.1	1	5.1	6.5	0.2	169	0.1	**338**
LiveJ	70.9	1	10.1	**7.0**	0.4	**177**	0.25	283
Twitter	2517.6	1	371.2	6.8	20.5	123	12.6	200

TABLE 3: Running time and Speedup Factor (SpF) eSIA on various platforms ($k = 100$, **par-2 GPUs** refers to parallel algorithm on 2 GPUs)

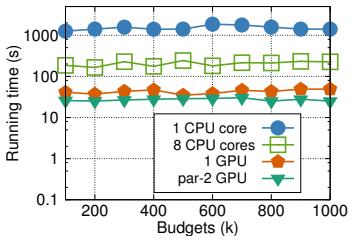


Fig. 5: Running time on Twitter

Comparison with GreedyCutting [9]. We compare eSIA with the GreedyCutting [9] which solves the slightly different Edge Deletion problem that interdicts the *sum* of nodes' influences while eSI minimizes the combined influence. Thus, to compare the methods for the two problems, we set the number of sources to be 1. Since we are interested in interdicting nodes with high impact on networks, we select top 10 nodes in Skitter network[2] with highest degrees and randomize their probabilities. We carry 10 experiments, each of which takes 1 out of 10 nodes to be the suspect. For GreedyCutting, we keep the default setting of 100 sample graphs and also test with 500 samples. We follow the edge probability settings in [9] that randomizes the edge probabilities and then normalizes by,

$$w(u,v) = \frac{w(u,v)}{\sum_{(w,v)\in E} w(w,v)} \tag{23}$$

so that the sum of edge probabilities into a node is 1. Afterwards, we take the average influence suspension and running time over all 10 tests and the results are drawn in Fig. 6.

Results: From Fig. 6, we see that clearly eSIA both obtains notably better solution quality, i.e., 10% to 50% higher, and runs substantially faster by a factor of up to 20 (CPU 1 core) and 1250 (1 GPU) than GreedyCutting. Comparing between using 100 and 500 sample graphs in GreedyCutting, we see improvements in terms of Influence Suspension when 500 samples are used, showing the *quality degeneracy* of using insufficient graph samples.

6.2.1 Analyzing Nodes Selected for Removal

We aim at analyzing which kind of nodes, i.e., those in $\mathcal{V}_I$ or popular nodes, selected by different algorithms.

Suspect Selection Ratio. We first analyze the solutions using Suspect Selection Ratio (SSR) which is the ratio of the number of suspected nodes selected to the budget k. We run the algorithms on all five datasets and take the average. Our results are presented in Table 4. We see that the majority of nodes selected by nSIA are suspected while the other methods except InfMax-V_I rarely select these nodes. InfMax-V_I restricts its selection in V_I and thus, always has a value of 1. The small fraction of nodes selected by nSIA not in V_I are possibly border nodes between V_I and the outside.

Interdiction Cost Analysis. We measure the cost of removing a set S of nodes by the following function.

$$Cost(S) = \sum_{v\in S}(1 - p(v))\log(d_{in}(v) + 1) \tag{24}$$

2. Skitter is the largest network that we could run GreedyCutting due to an unknown error returned by that algorithm on larger networks.

The $Cost(S)$ function consists of two parts: 1) probability of node being suspected and 2) the popularity of that node in the network implied by its in-degree. Interdicting nodes with higher probability or less popular results in smaller cost indicating less interruption to the operations of network.

The Interdiction costs of the solutions returned by different algorithms are shown in Table 4. We observe that nSIA introduces the least cost, even less than that of InfMax-V_I possibly because InfMax-V_I ignores the probabilities of nodes being suspected. The other methods present huge cost to networks since they target high influence nodes.

6.3 Scalability

This set of experiments is devoted for evaluating the running time of nSIA and eSIA implementations on multi-core CPUs and single or multiple GPUs on large networks.

6.3.1 Parallel implementations on GPU(s) vs CPUs

We experiment the different parallel implementations: 1) single/multiple GPU, 2) multi-core CPUs to evaluate the performance in various computational platforms. Due to the strong similarity between nSIA and eSIA in terms of performance, we only measure the time and speedup factor (SpF) for nSIA. We use two Titan X GPUs for testing multiple GPUs. The results are shown in Table 3 and Fig. 5.

Running time. From Table 3, we observe that increasing the number of CPUs running in parallel achieves an effective speedup of 80% per core meaning with 8 cores,nSIA runs 6.5 times faster than that on a single CPU. On the other hand, using one GPU reduces the running time by 100 to 200 times while two parallel GPUs helps almost double the performance, e.g., 200 vs. 123 times faster on Twitter. Fig. 5 confirms the speedups on different budgets.

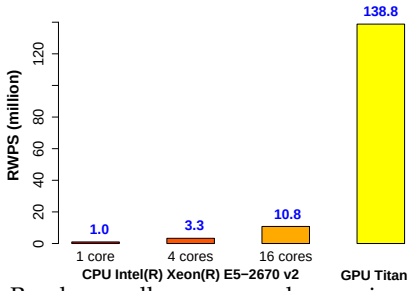


Fig. 7: Random walks per second on various platforms

Random Walk Generating Rate. We compare the rates of generating random walks (samples) on different parallel platforms, i.e., GPU and CPU. The Random Walks Per Second (RWPS) results on various platforms are described in Fig. 7.

Methods	Suspect Selection Ratio (SSR)					Interdiction Cost				
	200	400	600	800	1000	200	400	600	800	1000
nSIA	0.99	0.96	0.94	0.92	0.85	**303.55**	**541.73**	**720.73**	1070.50	**1204.53**
InfMax-V_I	1.00	1.00	1.00	1.00	1.00	333.90	600.60	825.75	**1020.00**	1246.03
InfMax-V	0.00	0.00	0.01	0.01	0.01	754.30	1302.85	1876.78	2427.58	2938.95
Pagerank	0.00	0.00	0.00	0.00	0.00	727.29	1562.06	2551.15	3301.35	4229.84
Max-Degree	0.00	0.00	0.00	0.00	0.00	820.03	1711.88	2660.15	3561.95	4505.40
Randomized	0.00	0.01	0.02	0.02	0.04	486.85	962.78	1424.65	1885.00	2364.33

TABLE 4: Interruption levels from removing nodes in the networks

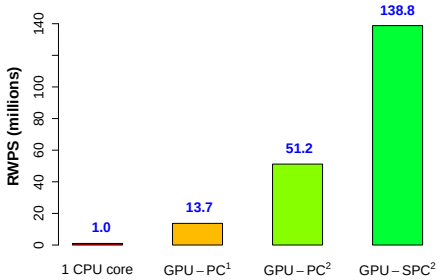


Fig. 8: Effects of different acceleration techniques

Unsurprisingly, the rate of random walk generation on CPU linearly depends on the number of cores achieving nearly 70% to 80% effectiveness. Between GPU and CPU, even with 16 cores of CPU, only 10.8 million random walks are generated per second that is around 13 times less than that on a GPU with 139 million.

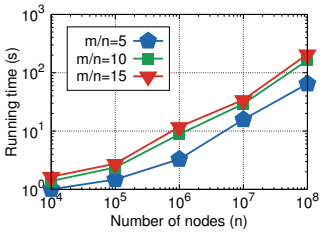


Fig. 9: Scalability tests on varying network sizes

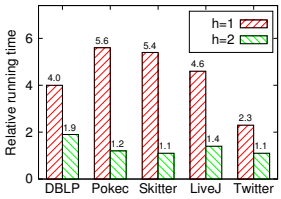


Fig. 10: Distributed algorithm on 2 GPUs

Scalability Test. We carry another test on the scalability of our GPU implementation. We create synthetic networks by GTgraph [31] with a number of nodes, denoted by n, increasing from tens of thousands, 10^4, to hundreds of millions, 10^8. For each value of n, we also test on multiple densities, i.e., the ratio of edges to nodes m/n. Specifically, we test with densities of $5, 10$ and 15. Our results are plotted in Fig. 9. The results show that the running time of nSIA increases almost linearly with the size of the network.

6.3.2 Distributed algorithm on multiple GPUs

We implemented our distributed nSIA algorithm on two GPUs and compared the performance with that on a single GPU. For the distributed version, we test on two values of extension parameter $h = 1$ and $h = 2$. The results are presented in Fig. 10. We see that the distributed algorithm on multiple GPUs is several times slower than that on a single GPU, illustrated with $h = 1$. However, this can be addressed by extending each partition to include nodes which are at most two hops away, illustrated with $h = 2$.

6.3.3 Effects of Acceleration Techniques on GPUs

We experimentally evaluate the benefit of our acceleration techniques. We compare 3 different versions of nSIA and eSIA: 1) GPU-PC^1 which employs $O(1)$-space Path-encoding and $O(1)$-space Cycle-detection by the slow Floyd's algorithm; 2) GPU-PC^2 which employs $O(1)$-space Path-encoding and $O(1)$-space Cycle-detection by the fast Brent's algorithm; 3) GPU-SPC^2 which applies all the techniques including the empirical Sliding-window early termination. We run four versions on all the datasets and compute the RWPS compared to that on a single-core CPU. The average results are in Fig. 8.

The experiment results illustrate the huge effectiveness of the acceleration techniques. Specifically, the $O(1)$-space Path-encoding combined with the slow Floyd's algorithm for Cycle-detection (GPU-PC^1) helps boost up the performance by 14x. When the fast Brent's algorithm is incorporated for $O(1)$-space Cycle-detection, the speedup is further increased to 51x while applying all the techniques effectively improves the running time up to 139x faster.

7 Related Work

Several works have studied removing/adding nodes/edges to minimize or maximize the influence of a node set in a network. [7], [8] proposed heuristic algorithms under the linear threshold model and its deterministic version. [32] studies the influence blocking problem under the competitive linear threshold model, that selects k nodes to initiate the inverse influence propagation to block the initial cascade. Misinformation containment has also been widely studied in the literature [33], [34]. Other than the LT model, the node and edge interdiction problems were studied under other diffusion models: [5] consider the SIR (Susceptible-Infected-Recovery) model while [35] considers the IC model.

The closest to our work is [9] in which the authors study two problems under the LT model: removing k edges to minimize the sum over influences of nodes in a set and adding k edges to maximize the sum. They prove the monotonicity and submodularity of their objective functions and then develop two approximation algorithms for the two corresponding problems. However, their algorithms do not provide a rigorous approximation factor due to relying on a fixed number of simulations. In addition, there is no efficient implementation for billion-scale networks.

Another closely related line of works is on Influence Maximization [10], [13], [25], [36] which selects a set of seed node that maximizes the spread of influence over the networks. Chen et al. [17] proved that estimating the influence of a set of nodes is #P-hard by counting simple paths (self-avoiding walks). Learning the parameters in

the propagation model has equally attracted great research interest [11], [12]. Network interdiction problems have intensively studied, e.g., interdicting maximum flow, shortest path, minimum spanning tree and many others (see [37] and the references therein).

GPUs have recently found effective uses in parallelizing and accelerating the practical performance of many problems [38]. Related to our spread interdiction problems, Liu et al. [39] propose a GPU-accelerated Greedy algorithm for the well-studied Influence Maximization problem to process sample graphs. In another direction, [40], [41] and several follow-ups study GPUs on the fundamental Breadth-First-Search problem achieving good performance improvement.

8 Conclusion

This paper aims at stopping an epidemic in a stochastic network $\mathcal{G}$ following the popular Linear Threshold model. The problems ask for a set of nodes (or edges) to remove from $\mathcal{G}$ such that the influence after removal is minimum or the *influence suspension* is maximum. We draw an interesting connection between the Spread Interdiction problems and the concept of *Self-avoiding Walk* (SAW). We then propose two near-optimal approximation algorithms. To accelerate the computation, we propose three acceleration techniques for parallel and distributed algorithms on GPUs. Our algorithms show significant performance advantage in both solution quality and running time over the state-of-the-art methods.

References

[1] 2017. [Online]. Available: http://globalsolutions.org/blog/2015/12/ISIS-Recruitment-Social-Media-Isolation-and-Manipulation

[2] 2017. [Online]. Available: http://www.cnbc.com/id/100646197

[3] H. A. M. Gentzkow, "Social media and fake news in the 2016 election," 2017. [Online]. Available: https://web.stanford.edu/~gentzkow/research/fakenews.pdf

[4] 2017. [Online]. Available: http://foreignpolicy.com/2011/12/20/u-s-officials-may-take-action-again-al-shababs-twitter-account

[5] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, "Gelling, and melting, large graphs by edge manipulation," in *CIKM*. New York, NY, USA: ACM, 2012, pp. 245–254.

[6] T. N. Dinh, H. T. Nguyen, P. Ghosh, and M. L. Mayo, "Social influence spectrum with guarantees: Computing more in less time," in *CSoNet*. Springer, 2015, pp. 84–103.

[7] M. Kimura, K. Saito, and H. Motoda, "Solving the contamination minimization problem on networks for the linear threshold model," in *PRICAI*. Springer, 2008, pp. 977–984.

[8] C. J. Kuhlman, G. Tuli, S. Swarup, M. V. Marathe, and S. S. Ravi, "Blocking simple and complex contagion by edge removal," in *ICDM*. IEEE, 2013, pp. 399–408.

[9] E. B. Khalil, B. Dilkina, and L. Song, "Scalable diffusion-aware optimization of network topology," in *KDD*. ACM, 2014, pp. 1226–1235.

[10] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *KDD*. ACM, 2003, pp. 137–146.

[11] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*. ACM, 2010, pp. 241–250.

[12] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi, "Measuring user influence in twitter: The million follower fallacy." *ICWSM*, vol. 10, no. 10-17, p. 30, 2010.

[13] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *SIGMOD*. ACM, 2015, pp. 1539–1554.

[14] Y. Zhang, A. Adiga, S. Saha, A. Vullikanti, and B. A. Prakash, "Near-optimal algorithms for controlling propagation at group scale on networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3339–3352, 2016.

[15] S. Khuller, A. Moss, and J. Naor, "The budgeted maximum coverage problem," *Info. Proc. Lett.*, vol. 70, no. 1, pp. 39–45, 1999.

[16] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Cost-aware targeted viral marketing in billion-scale networks," in *INFOCOM*. IEEE, 2016.

[17] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *ICDM*. IEEE, 2010, pp. 88–97.

[18] H. T. Nguyen, T. P. Nguyen, T. N. Vu, and T. N. Dinh, "Outward influence and cascade size estimation in billion-scale networks," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 1. ACM, 2017, pp. 63–63.

[19] X. Shi, X. Luo, J. Liang, P. Zhao, S. Di, B. He, and H. Jin, "Frog: Asynchronous graph processing on GPU with hybrid coloring model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 1, pp. 29–42, 2018.

[20] W. Lin, X. Xiao, X. Xie, and X.-L. Li, "Network motif discovery: A GPU approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 513–528, 2017.

[21] S.Vigna, "xorshift*/xorshift+ generators and the prng shootout," 2017. [Online]. Available: http://xoroshiro.di.unimi.it/

[22] D. Knuth, "The art of computer programming, vol ii: Seminumerical algorithms," 1998.

[23] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *CoNEXT*. ACM, 2014, pp. 75–88.

[24] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, "Improving graph partitioning for modern graphs and architectures," in *IA3*. ACM, 2015, p. 14.

[25] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *SIGMOD*. ACM, 2016, pp. 695–710.

[26] F. R. Chung and L. Lu, *Complex graphs and networks*. American Mathematical Soc., 2006, no. 107.

[27] "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," 2017. [Online]. Available: https://arxiv.org/abs/1605.07990

[28] K. Huang, S. Wang, G. Bevilacqua, X. Xiao, and L. V. Lakshmanan, "Revisiting the stop-and-stare algorithms for influence maximization," *VLDB*, pp. 913–924, 2017.

[29] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *SIGMOD*. ACM, 2014, pp. 75–86.

[30] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW*. ACM, 2010, pp. 591–600.

[31] 2017. [Online]. Available: http://www.cse.psu.edu/$\sim$kxm85/software/GTgraph/

[32] X. He, G. Song, W. Chen, and Q. Jiang, "Influence blocking maximization in social networks under the competitive linear threshold model." in *SDM*. SIAM, 2012, pp. 463–474.

[33] N. P. Nguyen, G. Yan, and M. T. Thai, "Analysis of misinformation containment in online social networks," *Computer Networks*, vol. 57, no. 10, pp. 2133–2146, 2013.

[34] K. K. Kumar and G. Geethakumari, "Detecting misinformation in online social networks using cognitive psychology," *HCIS*, vol. 4, no. 1, p. 1, 2014.

[35] M. Kimura, K. Saito, and H. Motoda, "Blocking links to minimize contamination spread in a social network," *TKDD*, vol. 3, no. 2, p. 9, 2009.

[36] N. Du, L. Song, M. Gomez-R, and H. Zha, "Scalable influence estimation in continuous-time diffusion networks," in *NIPS*, 2013, pp. 3147–3155.

[37] R. Zenklusen, "An o(1)-approximation for minimum spanning tree interdiction," in *FOCS*. IEEE, 2015, pp. 709–728.

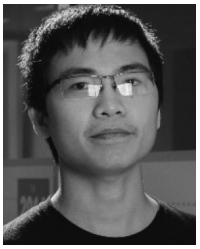
[38] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 1, p. e1232, 2018.

[39] X. Liu, M. Li, S. Li, S. Peng, X. Liao, and X. Lu, "IMGPU: GPU-accelerated influence maximization in large-scale social networks," *TPDS*, vol. 25, no. 1, pp. 136–145, 2014.

[40] D. Merrill, M. Garland, and A. Grimshaw, "Scalable GPU graph traversal," in *SIGPLAN Notices*, vol. 47, no. 8, 2012, pp. 117–128.

[41] H. Liu, H. H. Huang, and Y. Hu, "iBFS: Concurrent Breadth-First Search on GPUs," in *SIGMOD*. ACM, 2016, pp. 403–416.

[42] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," *Theory of Computing*, vol. 11, no. 4, pp. 105–147, 2015.



Hung T. Nguyen is currently a postdoctoral fellow at Carnegie Mellon University. Prior to that, he received his Ph.D. degree in Computer Science at Virginia Commonwealth University and BS in Information Technology from Vietnam National University. His research focuses on designing efficient approximation algorithms with optimality guarantees for problems in influence diffusion, such as influence maximization and source identification, on billion-scale networks under dynamic stochastic settings, and finding community structures in heterogeneous multiplex networks.

Alberto Cano is an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University, USA, where he heads the High-Performance Data Mining Lab. He obtained his BSc degrees in Computer Engineering and in Computer Science from the University of Cordoba, Spain, in 2008 and 2010, respectively, and his MSc and PhD degrees in Intelligent Systems and Computer Science from the University of Granada, Spain, in 2011 and 2014 respectively. His research is focused on machine learning, data mining, general-purpose computing on graphics processing units, Apache Spark, and evolutionary computation. He has published over 37 articles in high-impact factor journals, 39 contributions to international conferences, two book chapters, and one book in the areas of machine learning, data mining, and parallel, distributed, and GPU computing. His research is supported by an Amazon AWS Machine Learning award (2018) and the VCU Presidential Research Quest Fund (2018).

Tam Vu is an Assistant professor of Computer Science Department at University of Colorado Boulder. He directs Mobile and Networked Systems(MNS) Lab at the university, where he and his team conduct system research in the areas of wearable and mobile systems including mobile healthcare, mobile security, cyber physical system, and wireless sensing. His research has been recognized with NSF CAREER Award, two Google Faculty Awards, ten best paper awards, best paper nomination, and research highlights in flagship venues in mobile system research including MobiCom, MobiSys, and SenSys. He is also actively pushing his research outcomes to practice through technology transfer activities with 17 patents filed and leading two start-ups that he co-founded to commercialize them. He received Ph.D. from Rutgers University and B.S. from Ha Noi University of Science and Technology, Vietnam.

Thang N. Dinh received the Ph.D. degree in computer engineering from the University of Florida, in 2013. He is currently an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University. His research focuses on security and optimization challenges in complex systems, especially blockchain, social networks, critical infrastructures. He serves as the PC Chair of the BlockSEA'18, COCOON'16, CSoNet'14 and the TPC of several conferences including the IEEE INFOCOM, the ICC, the GLOBECOM. He is an Associate Editor for Springer Nature, Journal on Computational Science and Review Editor for Big Data