Coarsening Massive Influence Networks for Scalable Diffusion Analysis

Naoto Ohsaka [*#1], Tomohiro Sonobe [†#2], Sumio Fujita [§3], Ken-ichi Kawarabayashi [†#4]
[*] The University of Tokyo, [†] National Institute of Informatics
[§] Yahoo Japan Corporation, [#] JST, ERATO, Kawarabayashi Large Graph Project
ohsaka@is.s.u-tokyo.ac.jp[1], tomohiro_sonobe@nii.ac.jp[2]
sufujita@yahoo-corp.jp[3], k_keniti@nii.ac.jp[4]

ABSTRACT

Fueled by the increasing popularity of online social networks, social influence analysis has attracted a great deal of research attention in the past decade. The diffusion process is often modeled using *influence graphs*, and there has been a line of research that involves algorithmic problems in influence graphs. However, the vast size of today's real-world networks raises a serious issue with regard to computational efficiency.

In this paper, we propose a new algorithm for reducing influence graphs. Given an input influence graph, the proposed algorithm produces a vertex-weighted influence graph, which is compact and approximates the diffusion properties of the input graph. The central strategy of influence graph reduction is *coarsening*, which has the potential to greatly reduce the number of edges by merging a vertex set into a single weighted vertex. We provide two implementations; a *speed-oriented* implementation which runs in *linear time* with *linear space* and a *scalability-oriented* implementation which runs in practically *linear time* with *sublinear space*. Further, we present general frameworks using our compact graphs that accelerate existing algorithms for *influence maximization* and *influence estimation* problems, which are motivated by practical applications, such as viral marketing. Using these frameworks, we can quickly obtain solutions that have accuracy guarantees under a reasonable assumption. Experiments with real-world networks demonstrate that the proposed algorithm can scale to billion-edge graphs and reduce the graph size to up to 4%. In addition, our influence maximization framework achieves four times speed-up of a state-of-the-art *D-SSA* algorithm, and our influence estimation framework cuts down the computation time of a simulation-based method to 3.5%.

1. INTRODUCTION

Online social networks play an important role in the dissemination of content, such as rumors, ideas, and news, through word-of-mouth effects. In the past decade, social influence analysis has attracted significant attention in research communities. The stochastic process of network diffusion is often modeled on *influence graphs*, and there has been a line of research involving algorithmic problems in influence graphs, e.g., detecting the spread of contamination quickly [28], controlling the cascades of specific content via structure manipulation [8, 46], confining the propagation of misinformation [7], evaluating the diffusing power of a group of individuals, i.e., *influence estimation* [31,38], and extracting a small number of influential individuals, i.e., *influence maximization* [22], which are of great importance to marketing strategies [41]. Besides these studies, researchers have focused on the development of more sophisticated diffusion models [3, 17, 39], and learning the model parameters from a log data [13, 18].

However, the vast size of today's real-world networks raises a serious issue with regard to computational efficiency. For example, the Twitter social network contains 288 million users and 60 billion edges,[1] and the Facebook social network contains 1.4 billion users and 400 billion edges [11]. Indeed, for such extremely large networks, the input graph may not fit in main memory. Since existing in-memory algorithms for influence maximization, such as *PMC* [37], *TIM*$^+$ [44], *IMM* [43], and *D-SSA* [36], require random accesses, running them under a disk-based setting incurs prohibitive I/O costs.

One promising way to resolve this issue is to preprocess the input graph to reduce its size. Various graph reduction algorithms have been developed [19, 26, 42]; however, influence graphs behave *stochastically*, which makes it difficult to apply existing reduction algorithms.

1.1 Contributions

In this paper, we propose a scalable reduction algorithm designed for influence graphs. Given an input influence graph, the proposed algorithm produces a compact vertex-weighted influence graph that approximates the diffusion properties of the input graph under a well-studied diffusion model called independent cascade. The advantage of our algorithm is its scalability. We provide two implementations; a *speed-oriented* implementation which runs in *linear time* with *linear space* and a *scalability-oriented* implementation which runs in practically *linear time* with *sublinear space*. Further, we present general frameworks that accelerate existing algorithms for *influence estimation* and *influence maximization* problems, which are strongly motivated by practical applications, such as viral marketing. Using

[1]http://www.beevolve.com/twitter-statistics

SIGMOD '17, May 14–19, 2017, Chicago, IL, USA.
© 2017 ACM. ISBN 978-1-4503-4197-4/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3035918.3064045

these frameworks, we can quickly obtain solutions that have accuracy guarantees under a reasonable assumption.

The central strategy for influence graph reduction is a procedure called *coarsening*, which merges a vertex set into a single weighted vertex. In order to investigate what type of vertex set is desired to be coarsened, we analyze the impact of coarsening on the diffusion properties and the graph size. Based on our results, we introduce a novel notion of vertex set which is desired to be coarsened called *r-robust strongly connected components (SCCs)*. r-robust SCCs have the following advantages: *(1)* they are easy to find from large networks, *(2)* the trade-off between size reduction and loss of the diffusion properties is tunable with one parameter r, *(3)* coarsening them effectively reduces the number of edges, and *(4)* coarsening them preserves the diffusion properties of the input graph in practice. To enable our algorithm to work on massive graphs, we carefully design its implementations according to the available space.

Our intensive experiments on real-world large networks with up to billions of edges demonstrate the effectiveness, efficiency, and scalability of the proposed algorithm and influence analysis frameworks. Specifically, we confirm that *(1)* the linear-space implementation processes billion-edge graphs within one hour consuming hundreds of gigabytes while the sublinear-space implementation requires one-tenth of that space, *(2)* the proposed algorithm reduces the number of edges to up to 4%, *(3)* the proposed influence estimation framework cuts down the computation time of a simulation-based method [22] to 3.5%, and *(4)* the proposed influence maximization framework achieves up to four times speed-up of *D-SSA* [36], a recently developed state-of-the-art algorithm, without significant loss of solution quality.

Organization. The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces basic notions used throughout this paper. Section 4 explains and analyzes our coarsening procedure, and Section 5 proposes our algorithm and describes its implementations. Section 6 presents general frameworks that accelerate existing algorithms for influence estimation and influence maximization. Section 7 provides experimental results on real-world networks. Section 8 concludes the paper.

2. RELATED WORK

2.1 Reduction algorithms for influence graphs

A few studies have attempted to reduce influence graphs. Purohit *et al.* [40] proposed COARSENET, which produces a subgraph of an input graph by contracting unimportant edges that are identified based on the spectrum of the adjacency matrix. COARSENET requires $\mathcal{O}(|V|\Delta)$ time, where V is a vertex set and Δ is the maximum degree, which is expensive in practice. Note that our coarsening procedure is completely different from COARSENET. Mathioudakis *et al.* [33] proposed SPINE, which eliminates a specific number of edges preserving the likelihood given a log of past cascades. Note that SPINE works in a different scenario from our study.

The advantage of the proposed algorithm over these methods is its scalability and accuracy guarantee. Unlike the above methods, our algorithm runs in linear time with linear space, and the influences on the produced graphs are close to those on the input graph under a certain assumption.

2.2 Influence maximization

Motivated by an algorithmic study by Richardson and Domingos [15, 41], Kempe, Kleinberg, and Tardos [22] formulated the influence maximization problem as a discrete optimization problem to find a fixed-size set of vertices with the maximum influence under a specific diffusion model. They proved its NP-hardness, as well as the non-negativity, monotonicity, and submodularity of the influence function under two well-established diffusion models, i.e., independent cascade [16] and linear threshold [22]. Thus, a natural greedy strategy achieves a constant factor approximation to the optimal solution. The issue with this strategy is that no efficient method for influence computation is known. Kempe *et al.* resorted to expensive Monte-Carlo simulations. As a result, while providing high quality solutions, their greedy algorithm demonstrated poor scalability.

2.3 Scalable methods for influence maximization and influence estimation

Here, we briefly describe existing influence maximization methods, some of which can also be used for influence estimation. *Simulation*-based methods accurately estimate influence by simulating the diffusion process repeatedly for a given vertex set. Some existing approaches prune unnecessary influence computations [28, 37]. Other approaches employ a sample average approximation approach [10, 25, 37] and bottom-k sketches [12]. *Heuristic*-based methods avoid using Monte-Carlo simulations at the expense of solution quality. For example, heuristics have been developed that employ linear systems [20] or restrict the spread of influence to a particular group, including neighbors [10], communities [48], shortest paths [24], and the most probable influence paths [9]. The *sketch*-based methods developed by Borgs *et al.* [6] resolved the inefficiency of simulation-based methods without loss of accuracy guarantees. Rather than directly simulating the diffusion process, sketch-based methods rely on *reverse simulations* of the diffusion process and build sketches in advance to estimate the influence function efficiently. Subsequently, techniques for bounding the sketch size have been developed [14, 35, 36, 43, 44]. Nguyen, Thai, and Dinh [36] proposed the stop-and-stare algorithm (*SSA*) and its dynamic version (*D-SSA*) and empirically demonstrated that *SSA* and *D-SSA* are fastest among existing sketch-based methods. Despite such efforts, these algorithms still lack scalability against billion-edge-scale graphs in terms of both processing time and space consumption.

In Section 6, we present general frameworks that exploit the proposed coarsened influence graphs and enhance the scalability of existing algorithms.

3. PRELIMINARIES

Notations. For a positive integer ℓ, let $[\ell]$ denote the set $\{1, 2, \ldots, \ell\}$. We deal with two types of graphs. The first is a *deterministic graph* $G = (V, E)$, where V is a set of vertices and E is a set of directed edges. The second is an *influence graph* $\mathcal{G} = (V, E, p)$, where V is a set of vertices, E is a set of directed edges, and $p : E \to (0, 1]$ is an *influence probability function* representing the strength of influence between a pair of vertices. The influence subgraph of $\mathcal{G} = (V, E, p)$ induced by a vertex set $V' \subseteq V$ is denoted by $\mathcal{G}[V'] = (V', E', p')$, where E' is the set of edges spanned by V' and p' is the restriction of p to E'. Some graphs are associated with vertex weights $w : V \to \mathbb{N}$.

A graph (or vertex set) is considered *strongly connected* (SC) if vertices in it can reach each other. A *strongly connected component* (SCC) of a graph is a subgraph (or vertex set) that is strongly connected and maximal. Note that a family consisting of all SCCs forms a partition of the whole vertex set. For two partitions $\mathcal{P}$ and $\mathcal{Q}$ of the same set, if every element in $\mathcal{P}$ is a subset of some element in $\mathcal{Q}$, we say that $\mathcal{P}$ is a *refinement* of $\mathcal{Q}$, or $\mathcal{P}$ ($\mathcal{Q}$) is *finer* (*coarser*) than $\mathcal{Q}$ ($\mathcal{P}$). The *meet* of $\mathcal{P}$ and $\mathcal{Q}$, denoted $\mathcal{P} \wedge \mathcal{Q}$, is defined as the coarsest partition that is finer than both $\mathcal{P}$ and $\mathcal{Q}$.

3.1 Information diffusion model

Diffusion process. We adopt the most standard diffusion model called the *independent cascade* (IC), which was developed by Goldenberg, Libai, and Muller [16]. In the IC model, each vertex takes one of two states, *active* or *inactive*. An inactive vertex may become active, but not vice versa. Given an influence graph $\mathcal{G} = (V, E, p)$ and a *seed set* $S \subseteq V$, the diffusion process begins by activating vertices in S; all other vertices are inactive. Then, the process proceeds in discrete steps according to the following randomized rule. When a vertex u becomes active for the first time in step t, it is given a single chance to activate each current inactive vertex v among its out-neighbors. It succeeds with probability p_{uv}. If u succeeds, then v will become active in step $t+1$. Whether or not u succeeds, it cannot make any further attempt to activate v in subsequent steps. The process runs until no more activations are possible.

The *influence* of S in $\mathcal{G}$, denoted $\mathsf{Inf}_{\mathcal{G}}(S)$, is defined as the expected number of active vertices by initially activating vertices in S. If $\mathcal{G}$ has vertex weights, the influence is defined as the "sum of weights" of the active vertices.

Random-graph interpretation. Here, we describe the *random-graph interpretation* [22] of the IC model that characterizes its diffusion process. For an influence graph $\mathcal{G} = (V, E, p)$, let $\mathcal{D}_{\mathcal{G}}$ be the distribution over deterministic graphs (V, E'), where E' is obtained from E by maintaining each edge e with probability p_e. We use $G \sim \mathcal{D}_{\mathcal{G}}$ to mean that G is a *random graph* sampled from the distribution $\mathcal{D}_{\mathcal{G}}$. Hereafter, for an influence probability function $p : E \to (0, 1]$ and two edge subsets X and Y with $X \subseteq Y \subseteq E$, we denote the probability of obtaining X from Y by maintaining each edge with its influence probability as $p_{X|Y}$, i.e.,

$$p_{X|Y} = \prod_{e \in X} p_e \prod_{e \in Y \setminus X} (1 - p_e). \tag{1}$$

Note that the probability of sampling a fixed deterministic graph $G = (V, E')$ from $\mathcal{D}_{\mathcal{G}}$ is exactly $p_{E'|E}$. Then, Kempe *et al.* [22] proved that the influence of a seed set S in $\mathcal{G}$ is equal to the expected number (sum of weights) of vertices reachable from S in the random graph sampled from $\mathcal{D}_{\mathcal{G}}$, which is expressed as

$$\mathsf{Inf}_{\mathcal{G}}(S) = \mathbf{E}_{G \sim \mathcal{D}_{\mathcal{G}}}[\mathsf{R}_G(S)] = \sum_{E' \subseteq E} p_{E'|E} \cdot \mathsf{R}_{(V,E')}(S), \tag{2}$$

where $\mathsf{R}_G(S)$ is the number (sum of weights) of vertices reachable from S in G.

3.2 Influence estimation problem

Problem definition. Given an influence graph $\mathcal{G} = (V, E, p)$ and a seed set $S \subseteq V$, the *influence estimation* problem under the IC model requires computation of the influence $\mathsf{Inf}_{\mathcal{G}}(S)$ of S. It has been proven that exact influence computation is $\sharp$P-hard [9].

Naive simulation method. Since computing the influence function exactly is $\sharp$P-hard, *Monte-Carlo simulations* have been used for approximate computation. A naive simulation method repeatedly simulates the diffusion process for a given S and takes the average number (sum of weights) of activated vertices. Theoretically, $\Omega(\frac{|V|^2}{\epsilon^2} \ln \delta^{-1})$ simulations give a $(1 \pm \epsilon)$-approximation with probability of at least $1 - \delta$ [23]. In practice, tens of thousands of simulations are sufficient to obtain reasonable estimations [10, 22]. Note that the simulation cost depends on the number of examined edges in the diffusion process, and therefore, edge reduction is expected to drastically reduce computation cost.

3.3 Influence maximization problem

Problem definition. Given an influence graph $\mathcal{G} = (V, E, p)$ and an integer k, the *influence maximization* problem [22] under the IC model requires finding a seed set $S \subseteq V$ of size k that maximizes its influence $\mathsf{Inf}_{\mathcal{G}}(S)$. It has been proven that computing the optimal solution is NP-hard [22].

Naive greedy algorithm. Despite the NP-hardness of this problem, a natural greedy strategy achieves a constant factor approximation of the optimal solution due to the non-negativity, monotonicity, and submodularity of $\mathsf{Inf}_{\mathcal{G}}(\cdot)$. A set function $f : 2^V \to \mathbb{R}$ is *non-negative* if $f(S) \geq 0$ for all $S \subseteq V$, *monotone* if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq V$, and *submodular* if $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ for all $S \subseteq T \subseteq V$ and $v \in V \setminus T$. The greedy algorithm [22] begins with an empty seed set $S = \emptyset$, and then it adds a vertex v with the maximum marginal influence, i.e., $v = \mathrm{argmax}_{v' \in V \setminus S} \mathsf{Inf}_{\mathcal{G}}(S \cup \{v'\}) - \mathsf{Inf}_{\mathcal{G}}(S)$, to S until k vertices have been added. The following theorems guarantee that the greedy algorithm approximates the optimal solution within a factor that is slightly better than 0.63 by evaluating the influence function for $k \cdot |V|$ seed sets.

THEOREM 3.1 (NEMHAUSER, WOLSEY, AND FISHER [34]). *For a non-negative, monotone, and submodular function $f : 2^V \to \mathbb{R}$, let S be a set of size k obtained by the greedy algorithm. Then, it holds that $f(S) \geq (1 - \mathrm{e}^{-1}) f(S^*)$, where S^* is the optimal solution of size k.*

THEOREM 3.2 (KEMPE, KLEINBERG, AND TARDOS [22]). *For the IC model, the influence function is non-negative, monotone, and submodular.*

The issue with the greedy algorithm is that we cannot evaluate $\mathsf{Inf}_{\mathcal{G}}(\cdot)$ exactly. Although the naive simulation method provides accurate estimations, running it $k \cdot |V|$ times is computationally prohibitive even for small networks.

Sketching algorithm. Borgs *et al.* [6] established a sketching method to resolve the inefficiency issue of the above mentioned greedy algorithm. Beginning with an empty family $\mathcal{R} = \emptyset$, the sketching method repeats the following procedure: sample a vertex z from V uniformly at random, compute a set R of vertices that would influence z in an outcome of the diffusion process, and add R to $\mathcal{R}$. The above procedure continues until $\mathcal{R}$ includes a sufficiently large number of vertex sets. Then, it computes an approximate solution S for the *maximum coverage problem* which requires to select a set of k vertices that intersects the maximum number of vertex sets in $\mathcal{R}$, by the greedy algorithm, and returns S as a solution. Borgs *et al.* [6] proved that a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution for $\epsilon > 0$ can be obtained in near-linear time with a constant probability.

Although several techniques for bounding the sketch size have been proposed [14, 35, 36, 43, 44], they still lack scalability against graphs with billions of edges. In Section 6.2, we provide a general framework using our coarsened graphs that accelerates existing algorithms.

4. OUR STRATEGY

4.1 Definition of coarsening

We here describe the proposed influence graph reduction strategy. The central strategy is *coarsening*, which merges a certain vertex set into a single weighted vertex, in the intention of making no distinction among vertices in the set. Our coarsening procedure is formally defined as follows.

Definition 4.1 (Coarsened influence graph). *Let $\mathcal{G} = (V, E, p)$ be an influence graph and $\mathcal{P} = \{C_j\}_{j \in [\ell]}$ be a partition of V, where each C_j is strongly connected. Then, a* coarsened influence graph *obtained from $\mathcal{G}$ by coarsening $\mathcal{P}$ is defined as a vertex-weighted influence graph $\mathcal{H} = (W, F, q, w)$, where*

$$W = \{c_j \mid j \in [\ell]\}, \tag{3}$$

$$F = \{(c_x, c_y) \mid c_x \neq c_y, \exists (u, v) \in E, u \in C_x, v \in C_y\}, \tag{4}$$

$$q_{c_x c_y} = 1 - \prod_{\substack{(u,v) \in E \\ u \in C_x, v \in C_y}} (1 - p_{uv}) \qquad (\forall (c_x, c_y) \in F), \tag{5}$$

$$w(c_j) = |C_j| \qquad (\forall c_j \in W). \tag{6}$$

The correspondence mapping *$\pi : V \to W$ is defined as* $\pi(v) = c_j$ such that $v \in C_j$.

We abuse the notation to let π act on subsets by writing $\pi(S) = \{\pi(v) \mid v \in S\}$. It should also be noted that $F = \{(\pi(u), \pi(v)) \mid \pi(u) \neq \pi(v), (u, v) \in E\}$ and F contains no self-loops.

Example 4.2. *Let us take an example to give an intuition of the definition. Figures 1 and 2 show an influence graph $\mathcal{G}$ and a coarsened influence graph $\mathcal{H}$, respectively. Here, a vertex partition to be coarsened is $\mathcal{P} = \{C_1, C_2, C_3, C_4, C_5\} = \{\{v_1, v_2, v_3\}, \{v_4\}, \{v_5, v_6\}, \{v_7\}, \{v_8, v_9\}\}$. There is a one-to-one correspondence between a vertex set C_j in $\mathcal{P}$ and a vertex c_j in $\mathcal{H}$, i.e., $\mathcal{P}$ consists of five vertex sets, and thus, $\mathcal{H}$ consists of five vertices. Each vertex in $\mathcal{H}$ has a weight that is the size of the corresponding component in $\mathcal{G}$, e.g., $|C_1| = 3$; thus, $w(c_1) = 3$. When there is an edge connecting from C_x to C_y with $x \neq y$ in $\mathcal{G}$, then there is a corresponding edge from c_x to c_y in $\mathcal{H}$, e.g., (v_2, v_4) is in $\mathcal{G}$, and thus $(\pi(v_2), \pi(v_4)) = (c_1, c_2)$ is in $\mathcal{H}$. An activation through edge (c_x, c_y) in $\mathcal{H}$ corresponds to some activation among edges from C_x to C_y in $\mathcal{G}$, whose event probability is given by Eq. 5, e.g., influence probabilities of two edges connecting from C_1 to C_2 are 0.3 and 0.2, and thus, the influence probability of edge (c_1, c_2) is calculated as $1 - (1 - 0.3)(1 - 0.2) = 0.44$.*

Intuitively, for a seed set $S \subseteq V$, the diffusion process on $\mathcal{H}$ starting from $\pi(S)$ "emulates" the diffusion process on $\mathcal{G}$ starting from S, and we use $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$ as an approximation of $\mathsf{Inf}_{\mathcal{G}}(S)$. Hence, a coarsened influence graph $\mathcal{H}$ is preferable if *(1)* $\mathcal{H}$ is much smaller than $\mathcal{G}$ and *(2)* $\mathsf{Inf}_{\mathcal{H}}(\pi(\cdot))$ is close to $\mathsf{Inf}_{\mathcal{G}}(\cdot)$. Of course, there are exponentially many candidates for the vertex partition to be coarsened, and we cannot evaluate all the candidates. To resolve this issue, in subsequent sections, we will investigate what type of vertex partition is desired and discuss how to create such a partition.

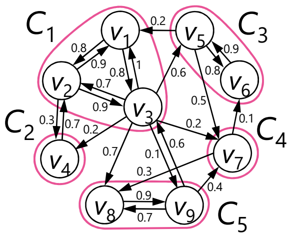


Figure 1: Influence graph $\mathcal{G}$.

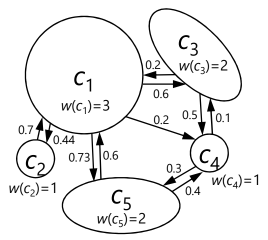


Figure 2: Coarsened influence graph $\mathcal{H}$.

4.2 Theoretical properties of coarsening

We investigate the impact of coarsening on the influence function and the graph size. For an influence graph $\mathcal{G} = (V, E, p)$ and a partition $\mathcal{P} = \{C_j\}_{j \in [\ell]}$ of V, let $\mathcal{H} = (W, F, q, w)$ be an influence graph and $\pi : V \to W$ be the correspondence mapping obtained from $\mathcal{G}$ by coarsening $\mathcal{P}$. Then, an *intermediate influence graph* between $\mathcal{G}$ and $\mathcal{H}$ is defined as $\mathcal{I} = (V, E, p')$, where $p'_{uv} = 1$ if $u, v \in C_j$ for some $j \in [\ell]$, otherwise $p'_{uv} = p_{uv}$.

Here, we show the equivalence between $\mathcal{I}$ and $\mathcal{H}$ in terms of the influence function. Therefore, it is sufficient to examine the influence on $\mathcal{I}$, which is of the same structure as $\mathcal{G}$, rather than $\mathcal{H}$.

Lemma 4.3. *For any $S \subseteq V$, $\mathsf{Inf}_{\mathcal{I}}(S) = \mathsf{Inf}_{\mathcal{H}}(\pi(S))$.*

Proof. For each $j \in [\ell]$, we define $E_j = \{(u, v) \in E \mid u, v \in C_j\}$ and $E_\mathrm{o} = E \setminus (\bigcup_{j \in [\ell]} E_j)$. Note that a family consisting of $E_1, \ldots, E_\ell$ and E_o forms a partition of E. For $X_j \subseteq E_j$, $p'_{X_j | E_j}$ is 1 if $X_j = E_j$, otherwise 0. Thus, by Eq. 2,

$$\mathsf{Inf}_{\mathcal{I}}(S) = \sum_{X \subseteq E} p'_{X|E} \cdot \mathsf{R}_{(V,X)}(S) \tag{7}$$

$$= \sum_{X_1 \subseteq E_1} p'_{X_1|E_1} \cdots \sum_{X_\ell \subseteq E_\ell} p'_{X_\ell|E_\ell} \sum_{X_\mathrm{o} \subseteq E_\mathrm{o}} p'_{X_\mathrm{o}|E_\mathrm{o}} \cdot \mathsf{R}_{(V, X_\mathrm{o} \cup \bigcup_j X_j)}(S) \tag{8}$$

$$= \sum_{X_\mathrm{o} \subseteq E_\mathrm{o}} p'_{X_\mathrm{o}|E_\mathrm{o}} \cdot \mathsf{R}_{(V, X_\mathrm{o} \cup \bigcup_j E_j)}(S). \tag{9}$$

For each $Y \subseteq F$, we define a family $\mathcal{X}_Y$ of edge sets as $\mathcal{X}_Y = \{X_\mathrm{o} \subseteq E_\mathrm{o} \mid \{(\pi(u), \pi(v)) \mid \pi(u) \neq \pi(v), (u, v) \in X_\mathrm{o}\} = Y\}$. Note that a collection of $\{\mathcal{X}_Y\}_{Y \subseteq F}$ forms a partition of the power set 2^{E_o}. Recall that $q_{Y|F} = \prod_{e \in Y} q_e \prod_{e \in F \setminus Y} (1 - q_e)$. From the construction of q (Eq. 5), it turns out that

$$q_{Y|F} = \sum_{X_\mathrm{o} \in \mathcal{X}_Y} p'_{X_\mathrm{o}|E_\mathrm{o}}. \tag{10}$$

For any s and t in V and $X_\mathrm{o} \in \mathcal{X}_Y$, "$s$ can reach t by passing through the edges in $X_\mathrm{o} \cup \bigcup_j E_j$" if and only if "$\pi(s)$

can reach $\pi(t)$ by passing through the edges in Y" since every subgraph (C_j, E_j) for $j \in [\ell]$ is strongly connected; therefore, it holds that $\mathsf{R}_{(V, X_o \cup \bigcup_j E_j)}(S) = \mathsf{R}_{(W,Y)}(\pi(S))$ for all $X_o \in \mathcal{X}_Y$. Thus,

$$\mathsf{Inf}_{\mathcal{I}}(S) = \sum_{Y \subseteq F} \sum_{X_o \in \mathcal{X}_Y} p'_{X_o | E_o} \cdot \mathsf{R}_{(V, X_o \cup \bigcup_j E_j)}(S) \tag{11}$$

$$= \sum_{Y \subseteq F} q_{Y|F} \cdot \mathsf{R}_{(W,Y)}(\pi(S)) = \mathsf{Inf}_{\mathcal{H}}(\pi(S)). \tag{12}$$

□

The gap of influence between $\mathcal{G}$ and $\mathcal{H}$. We first give a lower bound of the influence on $\mathcal{I}$. From the following lemma, whose proof is deferred to Appendix A, it turns out that $\mathsf{Inf}_{\mathcal{I}}(S) \geq \mathsf{Inf}_{\mathcal{G}}(S)$ for any $S \subseteq V$.

LEMMA 4.4. *Let $\mathcal{G} = (V, E, p)$ and $\mathcal{G}' = (V, E, p')$ be two influence graphs with the same structure, where $p_e \leq p'_e$ for every edge e. Then, $\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{G}'}(S)$ for any $S \subseteq V$.*

For an influence graph $\mathcal{G} = (V, E, p)$, the *strongly connected reliability* of $\mathcal{G}$, denoted $\mathsf{Rel}(\mathcal{G})$, is defined as the probability that the random graph sampled from $\mathcal{D}_{\mathcal{G}}$ is strongly connected, i.e.,

$$\mathsf{Rel}(\mathcal{G}) = \sum_{X \subseteq E} p_{X|E} \cdot \big[(V, X) \text{ is SC}\big], \tag{13}$$

where $[\cdot]$ is the Iverson bracket, which returns 1 if the given statement is true, and 0 otherwise. This definition can be generalized to the subgraph $\mathcal{G}[V'] = (V', E', p')$ of $\mathcal{G}$ induced by a vertex set $V' \subseteq V$, i.e.,

$$\mathsf{Rel}(\mathcal{G}[V']) = \sum_{X' \subseteq E'} p'_{X'|E'} \cdot \big[(V', X') \text{ is SC}\big]. \tag{14}$$

We now show an upper bound of the influence on $\mathcal{I}$, whose proof is deferred to Appendix A.

LEMMA 4.5. *For any $S \subseteq V$,*

$$\mathsf{Inf}_{\mathcal{I}}(S) \leq \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1} \cdot \mathsf{Inf}_{\mathcal{G}}(S). \tag{15}$$

By Lemmas 4.3, 4.4, and 4.5, we have the following:

THEOREM 4.6. *For any $S \subseteq V$,*

$$\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{H}}(\pi(S)) \leq \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1} \cdot \mathsf{Inf}_{\mathcal{G}}(S). \tag{16}$$

In summary, $\mathsf{Inf}_{\mathcal{H}}(\pi(\cdot))$ well approximates $\mathsf{Inf}_{\mathcal{G}}(\cdot)$ when $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])^{-1}$ is small, i.e., $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ is large. The factor $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ takes a value from zero to one, and it approaches to one if for every $j \in [\ell]$, the random graph sampled from the distribution $\mathcal{D}_{\mathcal{G}[C_j]}$ is strongly connected with a high probability.

Monotonicity. Apart from the above results, we here show the monotonicity of the size and the influence function of coarsened graphs with respect to the refinement relation. Let $\mathcal{G} = (V, E, p)$ be an influence graph and $\mathcal{P}_1$ and $\mathcal{P}_2$ be two partitions of V. Then, for $i = 1, 2$, let $\mathcal{H}_i = (W_i, F_i, q_i, w_i)$ and $\pi_i : V \to W_i$ be a vertex-weighted influence graph and the correspondence mapping obtained from $\mathcal{G}$ by coarsening P_i, respectively. We obtain the following as a consequence of Definition 4.1 and Lemmas 4.3 and 4.4, whose proof is deferred to Appendix A.

THEOREM 4.7. *If $\mathcal{P}_1$ is a refinement of $\mathcal{P}_2$, then, $|W_1| \geq |W_2|$ and $|F_1| \geq |F_2|$.*

In particular, coarsening $\mathcal{P} = \bigcup_{v \in V} \{\{v\}\}$, which is a refinement of any partition, yields the original $\mathcal{G}$; therefore, coarsening definitely does not increase the graph size.

THEOREM 4.8. *If $\mathcal{P}_1$ is a refinement of $\mathcal{P}_2$, then, $\mathsf{Inf}_{\mathcal{H}_1}(\pi_1(S)) \leq \mathsf{Inf}_{\mathcal{H}_2}(\pi_2(S))$ for any $S \subseteq V$.*

To sum up, it is desirable if the fineness of a vertex partition is tunable.

4.3 Creating a partition to be coarsened

Definition of r-robust SCCs. We consider how to create a vertex partition which is desired to be coarsened. In the previous section, we demonstrated that

- coarsening $\{C_j\}_{j \in [\ell]}$ with large $\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j])$ preserves the influence on $\mathcal{G}$ (Theorem 4.6),
- the coarser a vertex partition is, the smaller a coarsened graph is (Theorem 4.7), and
- the finer a vertex partition is, the closer an influence function is to that of $\mathcal{G}$ (Theorem 4.8).

Unfortunately, computing the strongly connected reliability exactly is proven to be ♯P-hard [2,47], and even approximate computation through sampling requires a number of random graph generations. Thus, we rely on the following intuition:

> if a vertex set is strongly connected in a small number of random graphs, then it is likely to be strongly connected in other random graphs.

Now, we formally define the notion of r-robust SCCs.

DEFINITION 4.9 (r-ROBUST SCC). *Let $\mathcal{G} = (V, E, p)$ be an influence graph and $G_1 = (V, E_1), \ldots, G_r = (V, E_r)$ be r (fixed) random graphs sampled from $\mathcal{D}_{\mathcal{G}}$. Then, a vertex set $C \subseteq V$ (or an induced subgraph $\mathcal{G}[C]$) is identified as an* r-robust SCC *with regard to $G_1, \ldots, G_r$ if*

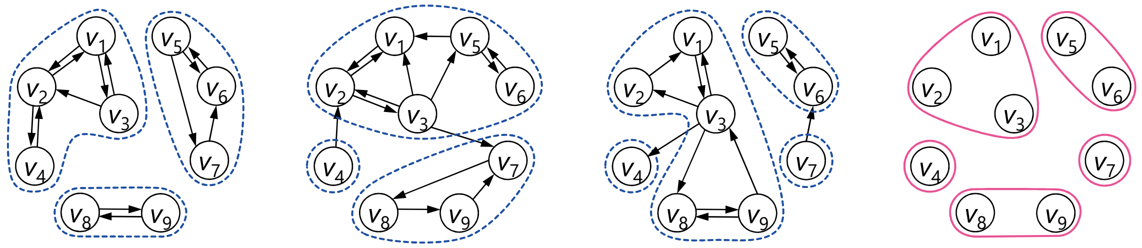
1. *for all $i \in [r]$, C is strongly connected in G_i, i.e., vertices in C are mutually reachable in G_i,*
2. *C is maximal.*

EXAMPLE 4.10. *Figure 3 illustrates three subgraphs G_1, G_2, and G_3 with the same vertex set and r-robust SCCs with regard to G_1, G_2, and G_3. Each r-robust SCC is strongly connected in all the three subgraphs and maximal, e.g., adding a vertex v_4 to an r-robust SCC $\{v_1, v_2, v_3\}$ violates the former condition.*

r-robust SCCs have the following convenient characterization, whose proof is straightforward from Definition 4.9.

THEOREM 4.11. *Let $\mathcal{C}_i$ be a partition consisting of all SCCs in G_i for each $i \in [r]$, and let $\mathcal{P}_r$ be a collection of all r-robust SCCs with regard to $G_1, \ldots, G_r$. Then, $\mathcal{P}_r = \bigwedge_{i \in [r]} \mathcal{C}_i$.*

Thus, a collection of r-robust SCCs forms a vertex partition, and it is easy to find, i.e., a partition consisting of all i-robust SCCs with regard to $G_1, \ldots, G_i$ can be computed incrementally as the meet of a partition consisting of



(a) Subgraph G_1. (b) Subgraph G_2. (c) Subgraph G_3. (d) r-robust SCCs.

Figure 3: Example of r-robust SCCs with regard to three subgraphs. Each blue dotted curve corresponds an SCC and each red solid curve corresponds an r-robust SCC.

all $(i-1)$-robust SCCs with regard to $G_1, \ldots, G_{i-1}$ and a partition consisting of SCCs in G_i. Note that an isolated vertex forms an r-robust SCC for any $r \geq 1$ since it is an SCC by itself. Furthermore, the value of r controls the fineness; a partition consisting of r-robust SCCs becomes finer as r increases.

Robustness. Here, we give a justification of r-robust SCCs and discuss the choice of r from a theoretical point of view. We first claim that coarsening r-robust SCCs will not significantly affect the influence function. Rather than aiming to bound the strongly connected reliability of r-robust SCCs, we show the following:

THEOREM 4.12. *For an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, let $\mathcal{P}_r$ be a (random) vertex partition consisting of r-robust SCCs with regard to r random graphs $G_1, \ldots, G_r$ sampled from $\mathcal{D}_\mathcal{G}$. Then, for any vertex set $V' \subseteq V$, the probability that V' is contained inside some element in $\mathcal{P}_r$ is at least $\mathsf{Rel}(\mathcal{G}[V'])^r$.*

The proof is presented in Appendix A. The above fact implies that the "most part" of an r-robust SCC has a large strongly connected reliability. In our experiments, we set $r = 16$ so that r-robust SCCs contain vertex sets with a large strongly connected reliability. For example, in Figure 1, a simple calculation according to Eq. 14 yields $\mathsf{Rel}(\mathcal{G}[C_1]) = 0.88848$; thus, some 16-robust SCC contains C_1 with probability at least $0.88848^{16} \approx 0.15$. Our experimental results in Section 7.4 verify this implication.

Density. We then claim that coarsening r-robust SCCs leads to a powerful edge reduction in the sense that r-robust SCCs are dense in practice. Our spotlight for this purpose is on a well-known structural property of *complex networks*, such as social networks and web graphs, called the *core-fringe structure* [30, 32], i.e., complex networks can be decomposed into two parts, a well-connected dense subgraph called *core* and a tree-like subgraph called *fringe*. It is also known that the core part contains a large k-edge-connected subgraph with high k, e.g., $k = 100$ [1].[2] The following theorem suggests that r-robust SCCs contain k-edge-connected subgraphs with high k.

THEOREM 4.13. *Assume that $\mathcal{G} = (V, E, p)$ is undirected*[3] *and the influence probability function p is a constant α, i.e., $p_e = \alpha$ for every edge e. Let $\mathcal{P}_r$ be a (random) vertex partition consisting of r-robust SCCs with regard to r random graphs sampled from $\mathcal{D}_\mathcal{G}$. If there exists a vertex set $V' \subseteq V$ such that the subgraph of (V, E) induced by V' is k-edge-connected and it holds that $\alpha^k = |V'|^{-(2+\delta)}$ for some $\delta > 0$, then, the probability that V' is contained inside some element in $\mathcal{P}_r$ is at least $(1 - |V'|^{-\delta}(1 + 2/\delta))^r$.*

The proof is presented in Appendix A. Note that every vertex in a k-edge-connected subgraph is of degree at least k, and thus, r-robust SCCs are expected to contain such a dense vertex set. For example, suppose that there exists a 100-edge-connected subgraph of size $|V'| = 10^6$, $\alpha = 0.1$, and $r = 16$. Then, $\delta = 44/3$ satisfies $\alpha^k = |V'|^{-(2+\delta)}$, and thus, the desired probability is at least $(1 - 10^{-88}(1 + 3/22))^{16}$, which is close to one. In Section 7.4, we will experimentally verify that the subgraph induced by the largest r-robust SCC is much denser than the whole graph.

Monotonicity. We finally show that the value of r controls the trade-off between size reduction and estimation accuracy. For an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, let $\mathcal{H}_r = (W_r, F_r, q_r, w_r)$ and $\pi_r : V \to W_r$ be random variables representing a vertex-weighted influence graph and the correspondence mapping obtained from $\mathcal{G}$ by coarsening $\mathcal{P}_r$, respectively. As a consequence of Theorems 4.7, 4.8, and 4.11, we have the following:

THEOREM 4.14. *The expected size of $\mathcal{H}_r$ increases as r increases, i.e.,*

$$\mathbf{E}[|W_1|] \leq \mathbf{E}[|W_2|] \leq \cdots \leq |V|, \tag{17}$$

$$\mathbf{E}[|F_1|] \leq \mathbf{E}[|F_2|] \leq \cdots \leq |E|, \tag{18}$$

where the expectation is taken over the choice of random graphs.

THEOREM 4.15. *The expected influence on $\mathcal{H}_r$ decreases as r increases, i.e., for any $S \subseteq V$,*

$$\mathbf{E}[\mathsf{Inf}_{\mathcal{H}_1}(\pi_1(S))] \geq \mathbf{E}[\mathsf{Inf}_{\mathcal{H}_2}(\pi_2(S))] \geq \cdots \geq \mathsf{Inf}_\mathcal{G}(S), \tag{19}$$

where the expectation is taken over the choice of random graphs.

5. OUR ALGORITHM

In this section, we present a scalable algorithm for coarsening an influence graph. We provide an overview of our algorithm followed by its implementations according to the available space and analyze their time, space, and I/O complexities.

[2] An undirected graph is *k-edge-connected* if it remains connected after removing fewer than k edges.

[3] For undirected graphs, "connectivity" is substituted for "strongly connectivity."

5.1 Overview

Here, we give an overview of the proposed method. Given an influence graph $\mathcal{G} = (V, E, p)$ and an integer r, the proposed method produces a (smaller) vertex-weighted influence graph $\mathcal{H} = (W, F, q, w)$. At a high level, the proposed method executes the following two stages.

- **First stage:** Create a partition $\mathcal{P}_r$ of V consisting of all r-robust SCCs with respect to r random graphs sampled from $\mathcal{D}_{\mathcal{G}}$.
- **Second stage:** Construct an influence graph $\mathcal{H}$ obtained from $\mathcal{G}$ by coarsening $\mathcal{P}_r$.

In the following, we provide detailed implementations according to the available space, i.e., a speed-oriented implementation with linear space and a scalability-oriented implementation with sublinear space.

5.2 Linear-space implementation

We first consider a situation wherein the entire input $\mathcal{G}$ can be stored in memory. Assume that $\mathcal{G}$ is already stored in memory, which actually requires $\mathcal{O}(|V| + |E|)$ I/O cost. Our implementation with linear space is presented in Algorithm 1, and the two stages are performed as follows.

First stage. In the first stage (Algorithm 1, lines 1–5), we create a partition consisting of all r-robust SCCs with regard to r random graphs. Since generating r random graphs at once consumes $\mathcal{O}(r(|V|+|E|))$ space, we reduce the memory consumption to $\mathcal{O}(|V| + |E|)$ by sequential generation of random graphs and incremental updates.

More precisely, beginning with $\mathcal{P}_0 = \{V\}$, which is actually the partition consisting of the (single) 0-robust SCC, we repeat the following process to obtain $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$, r times. In the i^{th} process, we sample the i^{th} random graph G_i from $\mathcal{D}_{\mathcal{G}}$, compute all of its SCCs $\mathcal{C}_i$, and identify a partition $\mathcal{P}_i$ consisting of all i-robust SCCs by computing the meet $\mathcal{P}_{i-1} \wedge \mathcal{C}_i$ (see Appendix B for the computation procedure of the meet of two partitions). Eventually, we obtain a partition $\mathcal{P}_r$ consisting of all r-robust SCCs with regard to $G_1, \ldots, G_r$.

Second stage. In the second stage (Algorithm 1, lines 6–12), we construct a coarsened influence graph $\mathcal{H}$.

Given an influence graph $\mathcal{G}$ and a partition $\mathcal{P}_r = \{C_j\}_{j\in[\ell]}$ of V, we naively construct the vertex set W, the edge set F, the vertex weights w, and the correspondence mapping π according to Definition 4.1. Here, the only non-trivial procedure is the calculation of the influence probability q_e for each e in F according to Eq. 5. We perform this with a single scan of (u, v)'s and p_{uv}'s using a hash table. Let q be a hash table storing influence probabilities for all edges in F with initial values of one. For each scanned edge (u, v) in E, we translate it onto $W \times W$, i.e., we compute $(\pi(u), \pi(v))$. If $(\pi(u), \pi(v))$ is not a self-loop, we update the entry of key $(\pi(u), \pi(v))$ in q as $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p_{uv})$. After scanning all edges, $q_e = 1 - \mathsf{q}[e]$ holds for every edge e in F, and we finally output an influence graph $\mathcal{H} = (W, F, q, w)$.

Efficiency analysis. The proof is presented in Appendix A.

THEOREM 5.1. *Algorithm 1 requires $\mathcal{O}(r(|V|+|E|))$ time, $\mathcal{O}(|V| + |E|)$ space, and $\mathcal{O}(|V| + |E|)$ I/O cost.*

Algorithm 1 Proposed algorithm with linear space.

```
Input: an influence graph $\mathcal{G} = (V, E, p)$ and an integer $r$.
Output: a vertex-weighted influence graph $\mathcal{H} = (W, F, q, w)$ and
    the correspondence mapping $\pi : V \to W$.
 1: $\mathcal{P}_0 \leftarrow \{V\}$.
 2: for $i = 1$ to $r$ do
 3:     $G_i \leftarrow$ a random graph sampled from $\mathcal{D}_{\mathcal{G}}$.
 4:     compute a partition $\mathcal{C}_i$ consisting of all SCCs in $G_i$.
 5:     $\mathcal{P}_i \leftarrow \mathcal{P}_{i-1} \wedge \mathcal{C}_i$.
 6: build $W, F, \pi, w$ from $\mathcal{G}$ and $\mathcal{P}_r$ according to Definition 4.1.
 7: $\mathsf{q}[c_x, c_y] \leftarrow 1$ for all $(c_x, c_y) \in F$.
 8: for all $(u, v) \in E$ do
 9:     if $(\pi(u), \pi(v)) \in F$ then
10:         $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p_{uv})$.
11: $q_e \leftarrow 1 - \mathsf{q}[e]$ for all $e \in F$.
12: return $\mathcal{H} = (W, F, q, w)$ and $\pi$.
```

Algorithm 2 Proposed algorithm with sublinear space.

```
Input: disk $\mathbb{D}_{\mathcal{G}}$ storing $\mathcal{G} = (V, E, p)$ and an integer $r$.
Output: disk $\mathbb{D}_{\mathcal{H}}$ storing $\mathcal{H} = (W, F, q, w)$ and $\pi : V \to W$.
 1: $\mathcal{P}_0 \leftarrow \{V\}$.
 2: for $i = 1$ to $r$ do
 3:     for all $\langle u, v, p_{uv} \rangle$ read from disk $\mathbb{D}_{\mathcal{G}}$ do
 4:         write $(u, v)$ to disk $\mathbb{D}_{G_i}$ with probability $p_{uv}$.
 5:     run a disk-based SCC algorithm on $G_i$ stored in disk $\mathbb{D}_{G_i}$.
 6:     $\mathcal{C}_i \leftarrow$ a partition of $V$ consisting of all SCCs in $G_i$.
 7:     $\mathcal{P}_i \leftarrow \mathcal{P}_{i-1} \wedge \mathcal{C}_i$.
 8: build $W, w, \pi$ from $\mathcal{G}$ and $\mathcal{P}_r$ according to Definition 4.1.
 9: write $W, w, \pi$ to disk $\mathbb{D}_{\mathcal{H}}$.
10: $F' \leftarrow \{(c_x, c_y) \in F \mid w(c_x) > 1 \vee w(c_y) > 1\}$.
11: $\mathsf{q}[c_x, c_y] \leftarrow 1$ for all $(c_x, c_y) \in F'$.
12: for all $\langle u, v, p_{uv} \rangle$ read from disk $\mathbb{D}_{\mathcal{G}}$ do
13:     if $(\pi(u), \pi(v))$ is not a self-loop then
14:         if $(\pi(u), \pi(v)) \in F'$ then
15:             $\mathsf{q}[\pi(u), \pi(v)] \leftarrow \mathsf{q}[\pi(u), \pi(v)] \cdot (1 - p_{uv})$.
16:         else                                  $\triangleright$ $(\pi(u), \pi(v)) \in F \setminus F'$.
17:             write $\langle \pi(u), \pi(v), p_{uv} \rangle$ to disk $\mathbb{D}_{\mathcal{H}}$.
18: for all $(c_x, c_y) \in F'$ do
19:     $q_{c_x c_y} \leftarrow 1 - \mathsf{q}[c_x, c_y]$.
20:     write $\langle c_x, c_y, q_{c_x c_y} \rangle$ to disk $\mathbb{D}_{\mathcal{H}}$.
21: return disk $\mathbb{D}_{\mathcal{H}}$.
```

5.3 Sublinear-space implementation

Next, we consider a situation wherein we cannot store the input graph $\mathcal{G}$ in memory. This situation happens frequently because simply storing all edges into memory consumes $8|E|$ bytes when a single edge is represented by a pair of 4-byte integers. To address this, we can reasonably assume that $\mathcal{G}$ is stored on a disk $\mathbb{D}_{\mathcal{G}}$ in the form of a sequence of triplets $\langle u, v, p_{uv} \rangle$, i.e., we can read $\langle u, v, p_{uv} \rangle$ sequentially in a certain order. Hereafter, $\mathbb{D}_I$ denotes a disk storing some information I (e.g., a graph).

Our implementation with sublinear space, presented in Algorithm 2, reduces its space complexity from $\mathcal{O}(|V|+|E|)$ to $\mathcal{O}(|V| + |F'|)$, where F' is defined as $F' = \{(c_x, c_y) \in F \mid w(c_x) > 1 \vee w(c_y) > 1\}$ and $|F'| \ll |F|$ in practice.

First stage. In the first stage (Algorithm 2, lines 1–7), we construct a partition consisting of all r-robust SCCs using only $\mathcal{O}(|V|)$ space, rather than $\mathcal{O}(|V| + |E|)$ space. To achieve this space requirement, we write each random graph to a disk and run a disk-based SCC algorithm with $\mathcal{O}(|V|)$ space, e.g., [27].

Specifically, beginning with $P_0 = \{V\}$, we repeat the following process r times, similar to the linear-space implementation. In the i^{th} process, consuming constant space, we read each triplet $\langle u, v, p_{uv} \rangle$ from $\mathbb{D}_{\mathcal{G}}$ and write (u, v) on $\mathbb{D}_{G_i}$ with probability p_{uv} one by one. Eventually, $\mathbb{D}_{G_i}$ stores

Algorithm 3 Proposed influence estimation framework.

```
Input: an influence graph $G = (V, E, p)$, a vertex set $S$, a coarsened graph $H = (W, F, q, w)$, the correspondence mapping $\pi : V \to W$, an influence estimation algorithm $\mathcal{A}$.
1: $T \leftarrow \pi(S)$.
2: $\mathsf{Inf}_{\text{out}} \leftarrow$ (approximately) compute $\mathsf{Inf}_{\mathcal{H}}(T)$ using $\mathcal{A}$.
3: return $\mathsf{Inf}_{\text{out}}$.
```

Algorithm 4 Proposed influence maximization framework.

```
Input: an influence graph $G = (V, E, p)$, a seed size $k$, a coarsened graph $H = (W, F, q, w)$, the correspondence mapping $\pi : V \to W$, an influence maximization algorithm $\mathcal{A}$.
1: compute a solution $T$ of size $k$ that (approximately) maximizes $\mathsf{Inf}_{\mathcal{H}}(\cdot)$ using $\mathcal{A}$.
2: $S_{\text{out}} \leftarrow$ select random $S$ such that $\pi(S) = T$.
3: return $S_{\text{out}}$.
```

all edges in G_i. Next, we apply a disk-based SCC algorithm to $\mathbb{D}_{G_i}$ and obtain a partition $\mathcal{C}_i$ consisting of all SCCs in G_i. We then compute $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$ and $\mathcal{C}_i$.

Second stage. In the second stage (Algorithm 2, lines 8–21), we construct a coarsened influence graph $\mathcal{H}$ given $\mathcal{G}$ and $\mathcal{P}_r$ using $\mathcal{O}(|V|+|F'|)$ space, rather than $\mathcal{O}(|V|+|E|)$ space. Recall that $F' = \{(c_x, c_y) \in F \mid w(c_x) > 1 \vee w(c_y) > 1\}$. A key factor of this space reduction is that $q_{\pi(u)\pi(v)} = p_{uv}$ holds if $(\pi(u), \pi(v)) \in F \setminus F'$. Thus, we do not need to store an entry with key $(\pi(u), \pi(v))$ in $F \setminus F'$ in a hash table.

More precisely, sequentially scanning each triplet $\langle u, v, p_{uv}\rangle$ from $\mathbb{D}_{\mathcal{G}}$, we update the entry $\mathsf{q}[\pi(u), \pi(v)]$ in the same manner as the linear-space implementation if $(\pi(u), \pi(v)) \in F'$; otherwise, we immediately write $\langle \pi(u), \pi(v), p_{uv}\rangle$ to $\mathbb{D}_{\mathcal{H}}$. After the scan is complete, we write $\langle c_x, c_y, q_{c_x c_y}\rangle$ to $\mathbb{D}_{\mathcal{H}}$ for each edge (c_x, c_y) in F'.

Our space reduction technique is effective if $w(c_j) = 1$ holds for most $c_j \in W$. This is the case in reality because vertices in the *tree-like* fringe part are rarely strongly connected. In fact, our experimental results demonstrate 90% reduction of memory usage compared to the linear-space implementation.

Efficiency analysis. The proof is presented in Appendix A.

THEOREM 5.2. *Algorithm 2 requires $\mathcal{O}(r(|V|+|E|+\mathrm{t}_{\mathcal{A}}(V, E)))$ time, $\mathcal{O}(|V| + |F'|)$ space, and $\mathcal{O}(r(|V| + |E| + \mathrm{i/o}_{\mathcal{A}}(V, E)))$ I/O cost, where $\mathrm{t}_{\mathcal{A}}(V, E)$ and $\mathrm{i/o}_{\mathcal{A}}(V, E)$ are the time and I/O complexities of a disk-based SCC algorithm $\mathcal{A}$ given a graph (V, E), respectively.*

6. SCALING-UP INFLUENCE ANALYSIS

Here, we present general frameworks that exploit our coarsened graphs for accelerating existing algorithms for influence estimation and influence maximization. Throughout this section, $\mathcal{G} = (V, E, p)$ is an input influence graph, $\{C_j\}_{j\in[\ell]}$ is a partition of V, $\mathcal{H} = (W, F, q, w)$ is a vertex-weighted influence graph obtained from $\mathcal{G}$ by coarsening $\{C_j\}_{j\in[\ell]}$, and $\pi : V \to W$ is a correspondence mapping. Note that $\mathcal{H}$ is not limited to be an output of our algorithm but a coarsened graph obtained by coarsening any partition.

6.1 Framework for influence estimation

Recall that the influence estimation problem requires the computation of $\mathsf{Inf}_{\mathcal{G}}(S)$ given a seed set $S \subseteq V$. To improve efficiency, our influence estimation framework runs an existing algorithm on $\mathcal{H}$ rather than directly running on $\mathcal{G}$. Specifically, given $\mathcal{H}$, π, S, and an influence estimation algorithm $\mathcal{A}$, our framework shown in Algorithm 3 translates S onto W via π, i.e., it computes $T = \pi(S)$, (approximately) computes $\mathsf{Inf}_{\mathcal{H}}(T)$ using $\mathcal{A}$, and returns an obtained estimation $\mathsf{Inf}_{\text{out}}$. Since $\mathcal{H}$ is smaller than $\mathcal{G}$, processing on $\mathcal{H}$ is expected to be more efficient than on $\mathcal{G}$. Moreover, we bound the relative error of our framework's estimation as follows, whose proof is deferred to Appendix A.

THEOREM 6.1. *For a non-empty seed set $S \subseteq V$, let $\mathsf{Inf}_{\text{out}}$ be an output of Algorithm 3, i.e., an estimation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$. If $\mathsf{Inf}_{\text{out}}$ is a $(1 \pm \epsilon)$-approximation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$, then, the relative error between $\mathsf{Inf}_{\mathcal{G}}(S)$ and $\mathsf{Inf}_{\text{out}}$ is bounded by*

$$-\epsilon \le \frac{\mathsf{Inf}_{\text{out}} - \mathsf{Inf}_{\mathcal{G}}(S)}{\mathsf{Inf}_{\mathcal{G}}(S)} \le \frac{1+\epsilon}{\prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j])} - 1. \qquad (20)$$

For example, consider applying our framework to a naive simulation method (Section 3.2). In our framework, the naive method simulates the diffusion process from $\pi(S)$ on $\mathcal{H}$ and takes the average weight of active vertices. Since $\mathcal{H}$ is smaller than $\mathcal{G}$, the diffusion process in $\mathcal{H}$ terminates earlier than in $\mathcal{G}$. Furthermore, $\mathsf{Inf}_{\text{out}}$ is a $(1 \pm \epsilon)$-approximation of $\mathsf{Inf}_{\mathcal{H}}(\pi(S))$ with high probability for a sufficiently large number of simulations; thus, Theorem 6.1 can be applied. Remark that when an output of our algorithm with r is used as $\mathcal{H}$, we can expect that the estimation accuracy improves as r increases due to Theorem 4.15.

6.2 Framework for influence maximization

Recall that the influence maximization problem seeks to select a seed set S of size k with the maximum influence on $\mathcal{G}$. Similar to the influence estimation framework described above, our influence maximization framework applies an existing method to $\mathcal{H}$. More precisely, given $\mathcal{H}$, π, a seed size k, and an influence maximization algorithm $\mathcal{A}$, our framework shown in Algorithm 4 first (approximately) solves the influence maximization problem for $\mathcal{H}$ and k using $\mathcal{A}$. Let $T \subseteq W$ be an obtained solution of size k. Then, in contrast to influence estimation, we translate T onto $\mathcal{G}$. To this end, we simply convert each vertex w in T to a random vertex v such that $\pi(v) = w$ and return an obtained set S_{out}. Processing on $\mathcal{H}$ is faster than on $\mathcal{G}$, and the following theorem gives an approximation ratio of Algorithm 4, whose proof is deferred to Appendix A.

THEOREM 6.2. *For an integer k, let S_{out} be an output of Algorithm 4. If T is an α-approximate solution of size k for $\mathcal{H}$, then, S_{out} is an $(\alpha \cdot \prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j]))$-approximate solution of size k for $\mathcal{G}$.*

For example, consider applying our framework to sketching algorithms (Section 3.3). In our framework, to build sketches, sketching algorithms repeatedly perform reverse simulations on $\mathcal{H}$ starting from a vertex selected from W with probability proportional to its weight. Then, they greedily select a vertex that intersects the maximum number of sketches and construct a seed set T of size k for $\mathcal{H}$. They finally return a random S such that $\pi(S) = T$. Here, T is a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution to the optimal solution of size k for $\mathcal{H}$ with high probability for some parameter ϵ; thus, S_{out} is a $((1 - \mathrm{e}^{-1} - \epsilon) \cdot \prod_{j\in[\ell]} \mathsf{Rel}(\mathcal{G}[C_j]))$-approximate solution of size k for $\mathcal{G}$.

Table 1: Datasets. (d) and (u) denote "directed" and "undirected," respectively.

name	type	$\|V\|$	$\|E\|$
ca-HepPh	collab.(u)	12,008	236,978
soc-Slashdot0922	social(d)	82,168	870,161
web-NotreDame	web(d)	325,729	1,469,679
wiki-Talk	commu.(d)	2,394,385	5,021,410
com-Youtube	social(u)	1,134,890	5,975,248
higgs-twitter	social(d)	456,626	14,855,819
soc-Pokec	social(d)	1,632,803	30,622,564
soc-LiveJournal1	social(d)	4,847,571	68,475,391
com-Orkut	social(u)	3,072,441	234,370,166
twitter-2010	social(d)	41,652,230	1,468,364,884
com-Friendster	social(u)	65,608,366	3,612,134,270
uk-2007-05	web(d)	105,218,569	3,717,169,969
ameblo	web(d)	272,687,914	6,910,266,107

7. EXPERIMENTAL EVALUATIONS

We conducted experiments using real-world networks to demonstrate the effectiveness, efficiency, and scalability of our algorithm and frameworks. The experiments were conducted on a Linux server with an Intel Xeon E5-2690 2.90GHz CPU and 256GB memory. The proposed algorithm and frameworks were implemented in C++ and compiled using g++v4.6.3 with the -O2 option.

7.1 Setup

Datasets. We used real-world social networks and web graphs. Table 1 summarizes the basic statistics of each network.[4] twitter-2010 and uk-2007-05 were downloaded from Laboratory for Web Algorithmics [4, 5]. ameblo is a crawled web graph of the ameblo.jp domain provided by Yahoo Japan Corporation, which was used to demonstrate the scalability of our algorithm.[5] The other networks were downloaded from Stanford Network Analysis Project [29]. For undirected graphs, we replaced each undirected edge with a pair of bidirected edges. All edges in the web graphs were reversed because it is natural to assume that information follows hyperlinks in the opposite direction.

Influence probability settings. To investigate the behavior of the algorithms under different probability settings, we employed the following four settings: *(1) Exponential* (EXP) setting under which each probability is chosen from the exponential distribution with mean 0.1. which is motivated by empirical evidence [3, 13], *(2) Trivalency* (TRI) setting [9] under which each probability is chosen randomly from a set $\{0.1, 0.01, 0.001\}$. *(3) Uniform cascade* (UC) setting [22] under which each edge has a constant probability 0.1. *(4) Weighted cascade* (WC) setting [22] under which the influence probability of an edge is set as the reciprocal of the in-degree of its head. The results for UC and WC are deferred to Appendix D.

Parameter settings. Our algorithm has a parameter r that controls the trade-off between size reduction and estimation accuracy. Unless otherwise specified, we set $r = 16$. This choice will be justified in Section 7.5 in terms of accuracy. Note that the sublinear-space implementation (Algorithm 2) uses an existing disk-based SCC algorithm proposed by Laura and Santaroni [27].

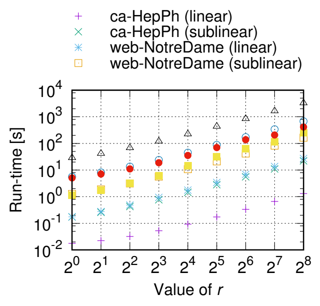


Figure 4: Run time with varying r (EXP).

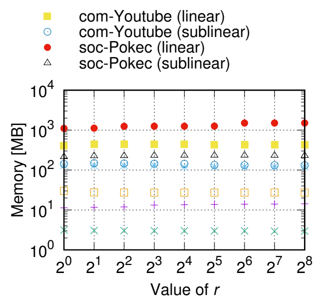


Figure 5: Memory usage with varying r (EXP).

7.2 Scalability evaluation

First, we examined the scalability of the proposed algorithm (Algorithms 1 and 2). Table 2 reports the run times and memory usages of the two implementations for each setting. Note that the run time of the linear-space implementation does not include the time required to read the input graph from and write the coarsened graph to secondary storage. Obviously, both the run time and memory usage scale linearly to the graph size and are robust against the probability setting. The linear-space implementation took approximately one hour and required roughly one hundred gigabytes for billion-edge graphs, but it ran out-of-memory for ameblo because the input and output graphs cannot fit in memory at the same time. Compared to the linear-space implementation, the sublinear-space implementation reduced the memory usage by 90% and ran on ameblo, while it is ten times slower.

Figures 4 and 5 plot the run time and memory usage of the two implementations against the value of r, respectively. As expected from the efficiency analyses (Theorems 5.1 and 5.2), the run time of both implementations scales linearly to r and the memory usage of linear-space implementation is not affected by r. Note that the memory usage of the sublinear-space implementation is also robust against the value of r.

7.3 Graph size reduction

Next, we investigated the effect of the proposed algorithm on graph size. Table 3 shows the numbers of vertices and edges and the corresponding reduction ratio of each coarsened graph. The proposed method reduced the number of edges to 3.6%–72.4% (EXP) and 15.4%–95.6% (TRI). Note that the edge reduction ratio is higher than the vertex reduction ratio because we extracted dense r-robust SCCs, as will be discussed in Section 7.4. Figure 6 shows the edge reduction ratio with various values of r. We can observe that the number of edges in the coarsened graphs logarithmically increases with r.

7.4 Analyzing extracted r-robust SCCs

Size distribution and density. Third, we analyzed the structural properties of the extracted r-robust SCCs. Figure 7 illustrates the size distribution of the extracted r-robust SCCs (EXP). As can be seen, a giant r-robust SCC exists, e.g., for soc-LiveJournal1 and com-Friendster, the largest r-robust SCCs are of size 3,370,90 and 18,897,527, respectively, while the second largest is of size 281 and 80, respec-

[4] Since we discarded self-loops, multi-edges, and isolated vertices, the numbers of vertices and edges for some datasets differ from those provided by the source websites.

[5] Experiments for ameblo were conducted on a Linux server with an Intel Xeon E5-2630L 2.00GHz CPU and 256GB memory because this dataset is only available on this machine at the moment.

Table 2: **Run time and memory usage of the proposed algorithm.**

	exponential				trivalency			
	linear space (Alg.1)		sublinear space (Alg.2)		linear space (Alg.1)		sublinear space (Alg.2)	
dataset	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**
ca-HepPh	0.09 s	14 MB	1.46 s	3 MB	0.07 s	16 MB	1.34 s	3 MB
soc-Slashdot0922	0.54 s	57 MB	5.57 s	13 MB	0.44 s	61 MB	5.28 s	11 MB
web-NotreDame	1.77 s	122 MB	10.86 s	28 MB	1.67 s	125 MB	10.57 s	26 MB
wiki-Talk	42.31 s	603 MB	57.37 s	270 MB	25.67 s	588 MB	55.47 s	249 MB
com-Youtube	14.59 s	452 MB	44.89 s	147 MB	12.22 s	472 MB	44.25 s	121 MB
higgs-twitter	7.64 s	530 MB	100.35 s	85 MB	5.00 s	686 MB	94.96 s	63 MB
soc-Pokec	35.20 s	1,280 MB	224.19 s	237 MB	30.49 s	1,489 MB	216.44 s	164 MB
soc-LiveJournal1	94.58 s	2,966 MB	507.54 s	677 MB	83.45 s	3,329 MB	473.83 s	480 MB
com-Orkut	122.34 s	6,288 MB	1,527.21 s	523 MB	103.55 s	7,183 MB	1,452.29 s	576 MB
twitter-2010	1,762.93 s	50,801 MB	11,522.38 s	5,635 MB	1,209.89 s	58,467 MB	10,354.66 s	5,325 MB
com-Friendster	3,964.05 s	101,398 MB	26,423.50 s	7,683 MB	3,120.62 s	113,337 MB	23,929.75 s	8,315 MB
uk-2007-05	3,105.88 s	136,659 MB	29,540.45 s	10,841 MB	2,619.00 s	166,670 MB	25,345.31 s	8,087 MB
ameblo	OOM	OOM	35,761.37 s	27,502 MB	OOM	OOM	16,926.93 s	20,684 MB

Table 3: **Effect of the proposed algorithm on graph size. V and E denote vertex and edge sets of an input graph, and W and F denote vertex and edge sets of a coarsened graph, respectively.**

	exponential				trivalency			
dataset	$\lvert W\rvert$	$\lvert W\rvert/\lvert V\rvert$	$\lvert F\rvert$	$\lvert F\rvert/\lvert E\rvert$	$\lvert W\rvert$	$\lvert W\rvert/\lvert V\rvert$	$\lvert F\rvert$	$\lvert F\rvert/\lvert E\rvert$
ca-HepPh	10,656	88.7%	74,036	31.2%	11,592	96.5%	127,618	53.9%
soc-Slashdot0922	78,202	95.2%	313,526	36.0%	81,450	99.1%	609,007	70.0%
web-NotreDame	321,196	98.6%	1,064,305	72.4%	324,525	99.6%	1,262,801	85.9%
wiki-Talk	2,389,034	99.8%	3,084,856	61.4%	2,392,612	99.9%	3,677,722	73.2%
com-Youtube	1,120,463	98.7%	3,437,134	57.5%	1,132,305	99.8%	4,471,436	74.8%
higgs-twitter	406,481	89.0%	4,077,314	27.4%	446,601	97.8%	9,891,098	66.6%
soc-Pokec	1,453,548	89.0%	13,292,116	43.4%	1,626,472	99.6%	29,277,656	95.6%
soc-LiveJournal1	4,498,127	92.8%	28,925,742	42.2%	4,800,660	99.0%	53,523,063	78.2%
com-Orkut	1,330,379	43.3%	8,446,726	3.6%	2,474,681	80.5%	64,035,002	27.3%
twitter-2010	38,805,787	93.2%	345,073,261	23.5%	40,746,064	97.8%	592,434,240	40.3%
com-Friendster	46,709,541	71.2%	171,230,294	4.7%	56,750,334	86.5%	557,214,040	15.4%
uk-2007-05	102,402,599	97.3%	1,553,084,998	41.8%	104,335,244	99.2%	2,578,741,487	69.4%
ameblo	271,042,159	99.4%	5,478,111,790	79.3%	272,624,016	99.9%	6,836,460,512	98.9%

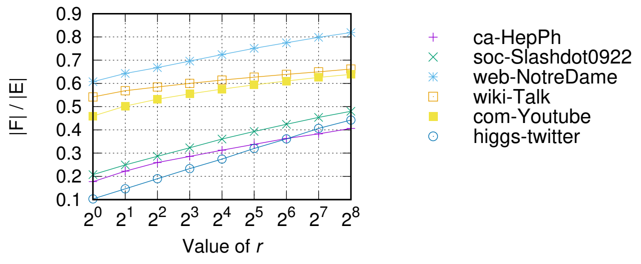
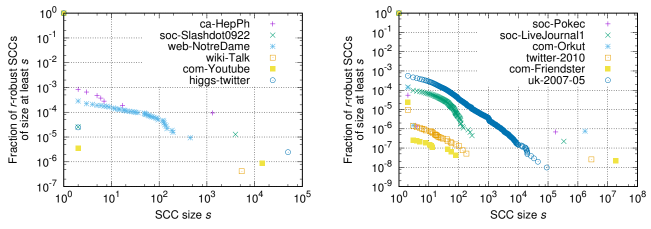


Figure 6: Edge reduction ratio with varying r (EXP).



(a) Smaller graphs (b) Larger graphs

Figure 7: Size distribution of r-robust SCCs (EXP).

tively. Furthermore, as discussed in Section 4.3, these components are dense; for soc-LiveJournal1 and com-Friendster, the average degree of the subgraph induced by the largest r-robust SCC is 57.9 and 154.8, while the average degree of the whole graph is 14.1 and 55.1, respectively. Thus, r-robust SCCs provide a vertex set whose coarsening leads to powerful edge reduction. We also observe that 99.9% of r-robust SCCs are of size one, which indicates that $|F'| \ll |F|$.

Robustness. Here, we investigated the robustness of the extracted r-robust SCCs. For a deterministic graph G, we define the *maximum SCC rate* as the size of the largest SCC in G divided by the number of vertices in G. This rate takes one if G is strongly connected in whole. Then, we consider the distribution of the maximum SCC rate of $\mathcal{G}[C_{\max}]$, where $C_{\max}$ is the largest r-robust SCC. To this end, we evaluate the probability that "the maximum SCC rate of the random graph sampled from $\mathcal{D}_{\mathcal{G}[C_{\max}]}$ is more than a threshold θ." We estimated this value by sampling 10,000 random graphs from $\mathcal{D}_{\mathcal{G}[C_{\max}]}$.

Figure 8 shows the cumulative distribution of the maximum SCC rate. We can see that the most part of $C_{\max}$ is strongly connected with a high probability. For example, 93% of the vertices in $C_{\max}$ of soc-Slashdot0922 are strongly connected in the random graph with probability 0.9. This coincides with the discussion in Section 4.3.

7.5 Evaluating influence estimation framework

Fourth, we evaluated our influence estimation framework (Algorithm 3). We applied our framework to the naive Monte-Carlo simulation method *MC* (Section 3.2), which repeatedly simulates the diffusion process and takes the average number (weight) of active vertices.

Efficiency improvement. We first observe the improvement of computational efficiency. To this end, we randomly sampled 10,000 vertices from the input graph and ran plain *MC* and our framework with *MC* to estimate the influence of each sampled vertex. Table 4 shows the total run time of

Table 4: Average estimation time of the influence of a single vertex for plain *MC* and our framework with *MC*. MARE and RCC stand for "mean absolute relative error" and "rank correlation coefficient," respectively.

	exponential					trivalency				
	run time			accuracy		run time			accuracy	
dataset	*MC*	Alg.3(*MC*)	$\frac{\text{Alg.3}(MC)}{MC}$	MARE	RCC	*MC*	Alg.3(*MC*)	$\frac{\text{Alg.3}(MC)}{MC}$	MARE	RCC
ca-HepPh	9.0 s	1.8 s	20.1%	0.0163	0.9991	2.8 s	926.5 ms	32.9%	0.0284	0.9995
soc-Slashdot0922	31.6 s	8.0 s	25.4%	0.0156	0.9998	9.1 s	5.2 s	57.1%	0.0370	0.9996
web-NotreDame	229.8 ms	46.5 ms	20.2%	0.0090	0.9989	60.9 ms	16.6 ms	27.3%	0.0021	0.9942
wiki-Talk	10.5 s	6.7 s	63.7%	0.0010	1.0000	4.4 s	2.0 s	46.5%	0.0030	1.0000
com-Youtube	135.2 s	51.9 s	38.4%	0.0497	0.9999	28.0 s	14.1 s	50.4%	0.1025	0.9978
higgs-twitter	1,213.4 s	280.7 s	23.1%	0.0177	0.9942	443.0 s	248.8 s	56.2%	0.0246	0.9998
soc-Pokec	2,442.3 s	897.1 s	36.7%	0.0143	0.9969	606.0 s	481.7 s	79.5%	0.0422	1.0000
soc-LiveJournal1	5,348.8 s	1,782.6 s	33.3%	0.0240	0.9995	859.2 s	668.9 s	77.9%	0.0913	0.9999
com-Orkut	33,122.7 s	1,239.0 s	3.7%	0.0071	0.8834	26,080.0 s	6,696.2 s	25.7%	0.0114	0.9913
twitter-2010	106,428.0 s	24,212.4 s	22.8%	–	–	31,234.2 s	11,312.4 s	36.2%	–	–
com-Friendster	540,483.0 s	18,967.8 s	3.5%	–	–	279,137.0 s	32,467.3 s	11.6%	–	–
uk-2007-05	5,718.6 s	1,900.2 s	33.2%	–	–	20.2 s	1.6 s	8.0%	–	–

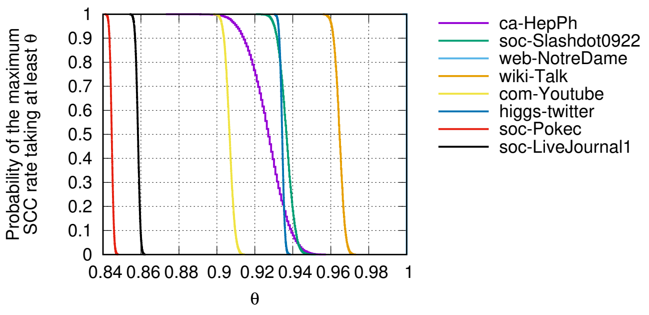
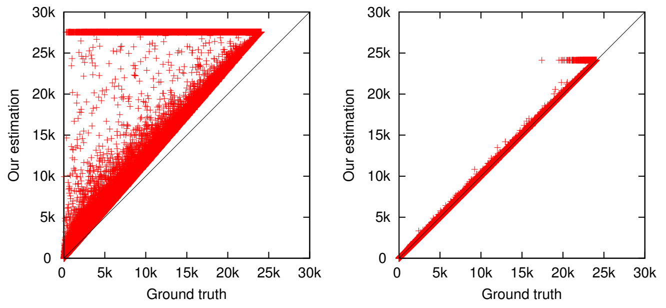


Figure 8: Cumulative distribution of the maximum SCC rate of the subgraph induced by the largest r-robust SCC (EXP).



soc-Slashdot0922 (EXP, $r = 1$) soc-Slashdot0922 (EXP, $r = 16$)

Figure 9: Influence correlation between the ground truth and our framework's estimation.

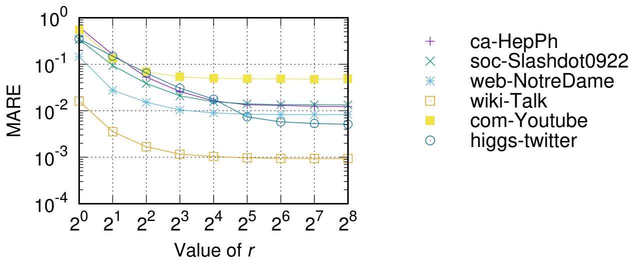


Figure 10: Estimation accuracy with varying r (EXP).

each method. Our framework drastically reduced the computation time to up to 3.5%. As the simulation cost is dominated by edge traversal costs, the time reduction ratio is roughly equal to the edge reduction ratio.

Estimation accuracy. We now examine the estimation accuracy of our framework. For each vertex v in the input graph, our framework ran *MC* on the coarsened graph, i.e., it simulated the diffusion process 100,000 times on the *coarsened graph*. We approximated the ground truth of the influence of each vertex v by simulating the diffusion process 100,000 times on the *input graph*.[6] We denote the ground truth and our framework's estimation for v's influence as $\mathsf{Inf}_{\mathrm{gt}}(v)$ and $\mathsf{Inf}_{\mathrm{out}}(v)$, respectively. First, we quantitatively evaluated our estimation accuracy. Table 4 reports the mean absolute relative error (MARE) defined as $\frac{1}{|V|}\sum_{v\in V}\left|\frac{\mathsf{Inf}_{\mathrm{gt}}(v)-\mathsf{Inf}_{\mathrm{out}}(v)}{\mathsf{Inf}_{\mathrm{gt}}(v)}\right|$ and Spearman's rank correlation coefficient (RCC) between the ground truth and our estimation. The MAREs are at most 0.1025, i.e., our estimation errors are within 10% on average, and the RCCs are always greater than 0.8834. These results demonstrate both the accuracy and stability of our framework.

We then qualitatively analyzed the estimation accuracy. Figure 9 illustrates the relationship between the two estimations, where each point corresponds to a single vertex v, and the x and y coordinate $\mathsf{Inf}_{\mathrm{gt}}(v)$ and $\mathsf{Inf}_{\mathrm{out}}(v)$, respectively. The estimation with $r = 1$ was heavily biased to higher values because we accidentally merged a 1-robust SCC, which has a small strongly connected reliability. For $r = 16$, we avoided merging such "fragile" components. As a result, most of the estimations are close to the diagonal line, i.e., the ground truth.

We here justify our choice of parameter r in terms of accuracy. Figure 10 presents MAREs with various values of r. Higher values of r yield more accurate estimations, but the improvements are limited when $r \geq 16$. Because our algorithm with lower r produces smaller coarsened graphs, we adopted $r = 16$ as a sweet spot between accuracy and size reduction.

7.6 Evaluating influence maximization framework

Fifth, we examined the effectiveness of our influence maximization framework (Algorithm 4). To this end, we applied our framework to *D-SSA* [36], a state-of-the-art algorithm.

[6] Since twitter-2010, com-Friendster, and uk-2007-05 are too large to compute the ground truth for every vertex, we were unable to obtain the results for these datasets.

Table 5: Run time for selecting a seed set of size 100 and solution quality for plain *D-SSA* and our framework with *D-SSA*. OOM denotes "out of memory."

	exponential					trivalency				
	run time			influence ($\mathsf{Inf}_\mathcal{G}/\lvert V\rvert$)		run time			influence ($\mathsf{Inf}_\mathcal{G}/\lvert V\rvert$)	
dataset	*D-SSA*	Alg.4(*D-SSA*)	$\frac{\text{Alg.4}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.4(*D-SSA*)	*D-SSA*	Alg.4(*D-SSA*)	$\frac{\text{Alg.4}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.4(*D-SSA*)
ca-HepPh	26.1 s	14.3 s	54.8%	0.3574	0.3586	28.8 s	10.6 s	36.8%	0.1668	0.1671
soc-Slashdot0922	140.6 s	78.9 s	56.1%	0.2951	0.2955	271.4 s	158.2 s	58.3%	0.1359	0.1358
web-NotreDame	3.0 s	1.1 s	34.9%	0.0828	0.0830	1.8 s	1.7 s	93.9%	0.0329	0.0329
wiki-Talk	521.6 s	154.7 s	29.7%	0.1396	0.1397	221.8 s	84.4 s	38.1%	0.0521	0.0521
com-Youtube	2,808.7 s	772.8 s	27.5%	0.1511	0.1509	927.7 s	636.6 s	68.6%	0.0547	0.0548
higgs-twitter	3,873.8 s	1,228.1 s	31.7%	0.3510	0.3510	5,251.6 s	3,275.6 s	62.4%	0.1776	0.1776
soc-Pokec	18,349.5 s	6,215.7 s	33.9%	0.4739	0.4739	5,538.9 s	5,129.3 s	92.6%	0.1972	0.1971
soc-LiveJournal1	OOM	OOM	–%	–	–	7,806.3 s	11,771.8 s	150.8%	0.1307	0.1310
com-Orkut	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
twitter-2010	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
com-Friendster	OOM	OOM	–%	–	–	OOM	OOM	–%	–	–
uk-2007-05	OOM	OOM	–%	–	–	595.4 s	208.5 s	35.0%	0.0792	0.0790

D-SSA produces a $(1 - \mathrm{e}^{-1} - \epsilon)$-approximate solution with probability at least $1 - \delta$ for parameters $\epsilon \in [0, 1 - 2 \cdot \mathrm{e}^{-1}]$ and $\delta \in [0, 1]$. We set $\epsilon = 0.1$ and $\delta = 0.01$.

Efficiency improvement. Table 5 summarizes the run time of plain *D-SSA* and our framework with *D-SSA* required to compute a solution of size 100. Our framework exhibited remarkable computation time reduction (up to 27.5%), which is approximately equal to the edge reduction ratio. This coincides with the *D-SSA* mechanism that iteratively performs reverse simulations, whose computational cost is dominated by edge traversal costs.

Solution quality. Table 5 reports the influence of a seed set extracted by plain *D-SSA* and our framework with *D-SSA*. To obtain reasonably accurate estimations, we conducted Monte-Carlo simulations 10,000 times. As expected from our framework's estimation accuracy, it provided solutions that were nearly the same quality as plain *D-SSA*.

7.7 Comparison with existing algorithms

Finally, we compared the scalability of the proposed algorithm to the following existing algorithms for influence graph reduction: COARSENET [40] and SPINE [33].

Settings. For COARSENET, we used a C++ implementation[7] provided by the authors of [40] with GNU Octave[8] version 3.2 for eigenvalue calculation. We ran COARSENET with the same edge reduction rate as that shown in Table 2.

For SPINE, we used a Java implementation[9] provided by the authors of [33]. Since SPINE requires a log of cascades to produce a sparsified graph, we conducted Monte-Carlo simulations from $|V|$ randomly selected vertices for each graph. We ran SPINE with the input graph and generated cascade logs so that the edge reduction ratio was the same as that of our algorithm's output.

Results. Table 6 compares the run time of our linear-space implementation, COARSENET, and SPINE for the exponential setting. Note that we observed the same tendency for the trivalency setting. The proposed method is several orders of magnitude faster than both existing algorithms for larger graphs. Moreover, due to out-of-memory error, COARSENET could handle only small graphs with tens of millions of edges and SPINE could process only the smallest graph. These results demonstrate the superiority of the proposed algorithm over existing algorithms in terms of both computation time and memory consumption.

[7] http://people.cs.vt.edu/~badityap/CODE/coarsenet.tgz

[8] https://www.gnu.org/software/octave/

[9] http://queens.db.toronto.edu/~mathiou/spine/

Table 6: Comparison of the run time of each algorithm (EXP). OOM denotes "out of memory."

	exponential		
dataset	This work (Alg.1)	COARSENET [40]	SPINE [33]
ca-HepPh	0.09 s	2.11 s	3,800.05 s
soc-Slashdot0922	0.54 s	116.75 s	OOM
web-NotreDame	1.77 s	55.26 s	OOM
wiki-Talk	42.31 s	4,760.76 s	OOM
com-Youtube	14.59 s	570.05 s	OOM
higgs-twitter	7.64 s	9,574.53 s	OOM
soc-Pokec	35.20 s	28,158.00 s	OOM
soc-LiveJournal1	94.58 s	OOM	OOM
com-Orkut	122.34 s	OOM	OOM
twitter-2010	1,762.93 s	OOM	OOM
com-Friendster	3,964.05 s	OOM	OOM
uk-2007-05	3,105.88 s	OOM	OOM

8. CONCLUSION

In this paper, we proposed a scalable algorithm for reducing influence graphs in order to scale-up influence analysis. The central approach of our graph reduction is coarsening, which has the potential to greatly reduce the number of edges by merging a vertex set into a single vertex. Based on its theoretical property, the notion of r-robust strongly connected components was introduced to extract a vertex partition which is desirable to be coarsened. According to the available space, we designed two implementations of our algorithm. In addition, we presented general frameworks that exploit coarsened graphs to accelerate existing algorithms for two influence analysis problems, i.e., influence estimation and influence maximization. Experiments with real-world networks demonstrated the effectiveness, efficiency, and scalability of the proposed algorithm and frameworks.

The flexibility of our method leads to several extensions. In Appendix C, we discuss parallelization of our method and dynamic updates of our coarsened graphs without recomputation. We will study further optimization in future work.

Acknowledgments. N. O. is supported by JSPS Grant-in-Aid for JSPS Research Fellow (No. 16J09440). Part of this work was done when N. O. is in research internship at Yahoo! JAPAN Research. K. K. is partially supported by NEDO, Internet of Things Promotion Department.

9. REFERENCES

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *CIKM*, pages 909–918, 2013.

[2] M. O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980.

[3] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM*, pages 81–90, 2012.

[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.

[5] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–602, 2004.

[6] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014.

[7] C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *WWW*, pages 665–674, 2011.

[8] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.

[9] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[10] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[11] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: Graph processing at Facebook-scale. *PVLDB*, 8(12):1804–1815, 2015.

[12] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014.

[13] L. Dickens, I. Molloy, J. Lobo, P.-C. Cheng, and A. Russo. Learning stochastic models of information flow. In *ICDE*, pages 570–581, 2012.

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

[15] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[16] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Market. Lett.*, 12(3):211–223, 2001.

[17] M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, pages 561–568, 2011.

[18] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.

[19] R. Jin, N. Ruan, S. Dey, and J. X. Yu. SCARAB: scaling reachability computation on large graphs. In *SIGMOD*, pages 169–180, 2012.

[20] K. Jung, W. Heo, and W. Chen. IRIE: Scalable and robust influence maximization in social networks. In *ICDM*, pages 918–923, 2012.

[21] D. R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514, 1999.

[22] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[23] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

[24] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *PKDD*, pages 259–271, 2006.

[25] M. Kimura, K. Saito, and R. Nakano. Extracting influential nodes for information diffusion on a social network. In *AAAI*, pages 1371–1376, 2007.

[26] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J. Cui, and A. G. Percus. Reducing large internet topologies for faster simulations. In *NETWORKING*, pages 328–341, 2005.

[27] L. Laura and F. Santaroni. Computing strongly connected components in the streaming model. In *TAPAS*, pages 193–205, 2011.

[28] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[29] J. Leskovec and A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, June 2014.

[30] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704, 2008.

[31] B. Lucier, J. Oren, and Y. Singer. Influence at scale: Distributed computation of complex contagion in networks. In *KDD*, pages 735–744, 2015.

[32] T. Maehara, T. Akiba, Y. Iwata, and K. Kawarabayashi. Computing personalized PageRank quickly by exploiting graph structures. *PVLDB*, 7(12):1023–1034, 2014.

[33] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537, 2011.

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

[35] H. T. Nguyen, T. N. Dinh, and M. T. Thai. Cost-aware targeted viral marketing in billion-scale networks. In *INFOCOM*, pages 1–9, 2016.

[36] 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*, pages 695–710, 2016.

[37] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Fast and accurate influence maximization on large networks with pruned Monte-Carlo simulations. In *AAAI*, pages 138–144, 2014.

[38] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Dynamic influence analysis in evolving networks. *PVLDB*, 9(12):1077–1088, 2016.

[39] N. Ohsaka, Y. Yamaguchi, N. Kakimura, and K. Kawarabayashi. Maximizing time-decaying influence in social networks. In *ECML-PKDD*, pages 132–147, 2016.

[40] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. Subrahmanian. Fast influence-based coarsening for large networks. In *KDD*, pages 1296–1305, 2014.

[41] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[42] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD*, pages 721–732, 2011.

[43] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD*, pages 1539–1554, 2015.

[44] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*, pages 75–86, 2014.

[45] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[46] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.

[47] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[48] Y. Wang, G. Cong, G. Song, and K. Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*, pages 1039–1048, 2010.

APPENDIX

A. PROOFS FROM SECTIONS 4–6

Proof of Lemma 4.4. We can construct a sequence of $|E|+1$ influence probability functions $p = p^0, p^1, \ldots, p^{|E|-1}, p^{|E|} = p'$ such that p^{i-1} and p^i differ in only at most one element, say e^*, and ${p^{i-1}}_{e^*} \leq {p^i}_{e^*}$ for $i \in [|E|]$. Then, $\mathsf{Inf}_{(V,E,p^{i-1})}(S) \leq \mathsf{Inf}_{(V,E,p^i)}(S)$ for each $i \in [|E|]$ follows from Eq. 2, and thus we obtain that $\mathsf{Inf}_{\mathcal{G}}(S) \leq \mathsf{Inf}_{\mathcal{G}'}(S)$. $\square$

Proof of Lemma 4.5. For each $j \in [\ell]$, we define $E_j = \{(u,v) \in E \mid u,v \in C_j\}$ and $E_{\mathrm{o}} = E \setminus (\bigcup_{j\in[\ell]} E_j)$. For $X_j \subseteq E_j$ ($j \in [\ell]$) and $X_{\mathrm{o}} \subseteq E_{\mathrm{o}}$, $\mathsf{R}_{(V,X_{\mathrm{o}}\cup\bigcup_j X_j)}(S) = \mathsf{R}_{(V,X_{\mathrm{o}}\cup\bigcup_j E_j)}(S)$ holds if every subgraph (C_j, X_j) for $j \in [\ell]$

is strongly connected. Thus,

$$\begin{aligned}
\mathsf{Inf}_{\mathcal{G}}(S) &= \sum_{X_1 \subseteq E_1} p_{X_1|E_1} \cdots \sum_{X_\ell \subseteq E_\ell} p_{X_\ell|E_\ell} \\
&\qquad \sum_{X_o \subseteq E_o} p_{X_o|E_o} \cdot \mathsf{R}_{(V, X_o \cup \bigcup_j X_j)}(S) && (21) \\
&\geq \sum_{\substack{X_1 \subseteq E_1 \\ (C_1, X_1) \text{ is SC}}} p_{X_1|E_1} \cdots \sum_{\substack{X_\ell \subseteq E_\ell \\ (C_\ell, X_\ell) \text{ is SC}}} p_{X_\ell|E_\ell} \\
&\qquad \sum_{X_o \subseteq E_o} p_{X_o|E_o} \cdot \mathsf{R}_{(V, X_o \cup \bigcup_j X_j)}(S) && (22) \\
&= \Bigg(\sum_{\substack{X_1 \subseteq E_1 \\ (C_1, X_1) \text{ is SC}}} p_{X_1|E_1} \Bigg) \cdots \Bigg(\sum_{\substack{X_\ell \subseteq E_\ell \\ (C_\ell, X_\ell) \text{ is SC}}} p_{X_\ell|E_\ell} \Bigg) \\
&\qquad \cdot \Bigg(\sum_{X_o \subseteq E_o} p_{X_o|E_o} \cdot \mathsf{R}_{(V, X_o \cup \bigcup_j E_j)}(S) \Bigg) && (23) \\
&= \prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j]) \cdot \mathsf{Inf}_{\mathcal{I}}(S). \quad \square && (24)
\end{aligned}$$

Proof of Theorem 4.7. The proof is a direct consequence of Definition 4.1. $\square$

Proof of Theorem 4.8. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two intermediate influence graphs for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. It is easy to see that every influence probability of $\mathcal{I}_1$ is at least that of $\mathcal{I}_2$. Hence, the proof is a direct consequence of Lemmas 4.3 and 4.4. $\square$

Proof of Theorem 4.12. This event is equivalent to the event that V' is contained inside some SCC in G_i for all $i \in [r]$, which occurs with probability at least $\mathsf{Rel}(\mathcal{G}[V'])^r$. $\square$

Proof of Theorem 4.13. By [21, Theorem 2.9], the random graph sampled from $\mathcal{D}_{\mathcal{G}[V']}$ is connected with probability at least $1 - |V'|^{-\delta}(1 + 2/\delta)$. Thus, by combining with Theorem 4.12, we obtain the desired bound. $\square$

Proof of Theorem 5.1. In the first stage, $\mathcal{O}(|V| + |E|)$ time is sufficient to generate a single random graph, find all of its SCCs [45], and compute the meet of two partitions as described in Appendix B. Thus, the whole process completes in $\mathcal{O}(r(|V|+|E|))$ time. During the i^{th} process, we maintain only the i^{th} random graph G_i, all of its SCCs $\mathcal{C}_i$, the $(i-1)^{\text{th}}$ partition $\mathcal{P}_{i-1}$, and the i^{th} partition $\mathcal{P}_i$; thus, $\mathcal{O}(|V| + |E|)$ space is sufficient.

In the second stage, $\mathcal{O}(|V| + |E|)$ time is required to construct $\mathcal{H}$ because each edge and influence probability are scanned once, and $\mathcal{H}$ consumes $\mathcal{O}(|W| + |F|)$ space, which is dominated by $\mathcal{O}(|V| + |E|)$.

Note that reading $\mathcal{G}$ from and writing $\mathcal{H}$ to secondary storage can be completed with $\mathcal{O}(|V| + |E|)$ I/O cost. $\square$

Proof of Theorem 5.2. In the first stage, the whole process obviously completes in $\mathcal{O}(r(|V| + |E| + \mathrm{t}_{\mathcal{A}}(V, E)))$ time and $\mathcal{O}(r(|V| + |E| + \mathrm{i/o}_{\mathcal{A}}(V, E)))$ I/O cost. During the i^{th} process, we maintain only $\mathcal{C}_i$, $\mathcal{P}_{i-1}$, and $\mathcal{P}_i$, which require $\mathcal{O}(|V|)$ space, and a disk-based SCC algorithm $\mathcal{A}$ is assumed to consume $\mathcal{O}(|V|)$ space. Thus, $\mathcal{O}(|V|)$ space is sufficient.

In the second stage, $\mathcal{O}(|V| + |E|)$ time is required to construct $\mathcal{H}$ because $\mathbb{D}_{\mathcal{G}}$ is scanned once. W, w, and π consume $\mathcal{O}(|W|) = \mathcal{O}(|V|)$ space, F' and $\mathtt{q}$ consume $\mathcal{O}(|F'|)$ space, and writing all information involving $\mathcal{H}$ requires $\mathcal{O}(|W| + |F|)$ I/O cost, which is dominated by $\mathcal{O}(|V| + |E|)$. $\square$

Algorithm 5 Computing the meet of two partitions.

```
1: procedure MEET(P, Q)
2:     map ← an empty hash table, M ← an empty array, and $\ell \leftarrow 1$.
3:     for all $v \in V$ do
4:         if map does not contain $\langle P[v], Q[v] \rangle$ then
5:             map[$\langle P[v], Q[v] \rangle$] ← $\ell$, and $\ell \leftarrow \ell + 1$.
6:         M[$v$] ← map[$\langle P[v], Q[v] \rangle$].
7:     return M.
```

Proof of Theorem 6.1. Remark that $\mathsf{Inf}_{\mathcal{G}}(S) > 0$ since $S \neq \emptyset$. Hence, The proof is a direct consequence of Lemmas 4.3, 4.4, and 4.5. $\square$

Proof of Theorem 6.2. Let S^* and T^* be the optimal solutions of size k for $\mathcal{G}$ and $\mathcal{H}$, respectively. Since Lemmas 4.3 and 4.4 ensure $\mathsf{Inf}_{\mathcal{H}}(T^*) \geq \mathsf{Inf}_{\mathcal{G}}(S^*)$, we have

$$\mathsf{Inf}_{\mathcal{H}}(\pi(S_{\mathrm{out}})) = \mathsf{Inf}_{\mathcal{H}}(T) \geq \alpha \cdot \mathsf{Inf}_{\mathcal{H}}(T^*) \geq \alpha \cdot \mathsf{Inf}_{\mathcal{G}}(S^*).$$

Then, applying Lemmas 4.3 and 4.5 yields the following:

$$\mathsf{Inf}_{\mathcal{G}}(S_{\mathrm{out}}) \geq \Bigg(\prod_{j \in [\ell]} \mathsf{Rel}(\mathcal{G}[C_j]) \Bigg) \cdot \alpha \cdot \mathsf{Inf}_{\mathcal{G}}(S^*). \quad \square \qquad (25)$$

B. COMPUTATION OF THE MEET

We here describe how to compute the meet of two partitions. Assume that a partition $\mathcal{P} = \{C_j\}_{j \in [\ell]}$ of a set V is represented as an array $\mathtt{P}$ of size $|V|$ where $\mathtt{P}[v] \in [\ell]$ and $v \in C_{\mathtt{P}[v]}$ for each v in V. Let $\mathtt{P}$ and $\mathtt{Q}$ be arrays representing two partitions $\mathcal{P}$ and $\mathcal{Q}$ of the same set V. Algorithm 5 describes a procedure for constructing an array $\mathtt{M}$ representing the meet $\mathcal{M} = \mathcal{P} \wedge \mathcal{Q}$ in time $\mathcal{O}(|V|)$ using a hash table.

C. EXTENSIONS

In this section, we consider two variants of our proposed algorithm. Thanks to its flexibility, we can easily extend it so as to support *parallel processing* and *dynamic updates*.

C.1 Parallel implementations

Overview. Let us begin with an overview of our parallel algorithm, which is applicable to both *shared-memory* and *distributed-memory* systems. Recall that our algorithm consists of the following two stages: *(1)* creates a partition $\mathcal{P}_r$ consisting of all r-robust SCCs, and *(2)* constructs a coarsened influence graph $\mathcal{H}$ from $\mathcal{G}$ and $\mathcal{P}_r$. Essentially, we simply parallelize the first stage. This significantly gains the scalability of the sublinear-space implementation (Algorithm 2), since the first stage runs a disk-based SCC algorithm many times, which is quite expensive (though depends on which method is employed).

Our parallel algorithm is presented in Algorithm 6. In addition to the ordinary input $\mathcal{G}$ and r, we are given the number T of threads to be created. For each t in $[T]$, we assign the number r_t of random graphs that the t^{th} thread will create in a balanced way so that $\sum_{i \in [T]} r_t = r$ and $|r_{t_1} - r_{t_2}| \leq 1$ for any $t_1, t_2 \in [T]$. Then, we launch the t^{th} thread and execute CREATEPARTITION(t, r_t), which creates a partition $\mathcal{P}(t)$ that consists of all r_t-robust SCCs with regard to r_t random graphs. When all the T threads have been completed, we have T partitions $\mathcal{P}(1), \ldots, \mathcal{P}(T)$. By computing the meet of them, we obtain a partition $\mathcal{P}_r$ consisting of all r-robust SCCs. We finally construct $\mathcal{H}$ and π according to Definition 4.1.

Implementations. We here demonstrate the effectiveness of our parallelization which works for two systems below.

Algorithm 6 Parallel implementation.

```
Input: $\mathcal{G}$, $r$, and the number of threads $T$.
Output: $\mathcal{H}$ and $\pi$.
1: for $t = 1$ to $T$ do
2:     $r_t \leftarrow \lfloor \frac{r+t-1}{T} \rfloor$.                ▷ $\sum_{1 \leq t \leq T} r_t = r$
3:     launch the $t^{\text{th}}$ thread to execute CreatePartition$(t, r_t)$.
4: wait until all the threads have completed.
5: $\mathcal{P}_r \leftarrow \bigwedge_{t \in [T]} \mathcal{P}(t)$.
6: construct $\mathcal{H}$ and $\pi$ for $\mathcal{G}$ and $\mathcal{P}_r$ according to Def. 4.1.
7: return $\mathcal{H}$ and $\pi$.

8: procedure CreatePartition$(t, r_t)$
9:     $\mathcal{P}(t) \leftarrow$ a partition consisting of all $r_t$-robust SCCs.
```

Algorithm 7 Dynamic updates of coarsened graphs.

```
1: procedure InsertEdge$((u,v), p_{uv})$
2:     for $i \in [r]$ do
3:         continue with probability $p_{uv}$.
4:         add $(u,v)$ to $G_i$, and compute all of its SCCs $\mathcal{C}_i$.
5:     if none of $\mathcal{C}_i$ has changed then
6:         if $\mathcal{H}$ does not contain $(\pi(u), \pi(v))$ then
7:             add $(\pi(u), \pi(v))$ to $\mathcal{H}$, and $q_{\pi(u)\pi(v)} \leftarrow 0$.
8:         $q_{\pi(u)\pi(v)} \leftarrow 1 - (1 - q_{\pi(u)\pi(v)}) \cdot (1 - p_{uv})$.
9:     else
10:        $\mathcal{P}_r \leftarrow \bigwedge_{i \in [r]} \mathcal{C}_i$.
11:        construct $\mathcal{H}$ and $\pi$ for $\mathcal{G}$ and $\mathcal{P}_r$ according to Def. 4.1.

12: procedure DeleteEdge$((u,v))$
13:     for $i \in [r]$ s.t. $G_i$ contains $(u,v)$ do
14:         remove $(u,v)$ from $G_i$, and compute all of its SCCs $\mathcal{C}_i$.
15:     if none of $\mathcal{C}_i$ has changed then
16:         $q_{\pi(u)\pi(v)} \leftarrow 1 - (1 - q_{\pi(u)\pi(v)})/(1 - p_{uv})$.
17:     else
18:         $\mathcal{P}_r \leftarrow \bigwedge_{i \in [r]} \mathcal{C}_i$.
19:         construct $\mathcal{H}$ and $\pi$ for $\mathcal{G}$ and $\mathcal{P}_r$ according to Def. 4.1.
```

We first adopt a *shared-memory* system, i.e., there exists a global address space, and parallel threads read from it and write to it concurrently. We used OpenMP for parallelization and ran the OpenMP implementation on the same environment as that described in Section 7. Table 7 shows the run times of the shared-memory parallel implementation, with linear and sublinear space, and with different number of threads (1, 4, and 16). Table 7 indicates that our parallel algorithm exhibited good scalability on both linear- and sublinear-space implementations, around 3–4 times speed-up with 16 threads.

We then consider a *distributed-memory* system, i.e., multiple processes that run in multiple machines send and receive data through Message Passing Interface (MPI). One master process constructs r random graphs and sends them to slave processes through MPI. Each slave, given a number r_t, creates a partition that consists of all r_t-robust SCCs and sends it back to the master. Finally, the master process computes $\mathcal{P}_r$ and constructs a coarsened influence graph. We used OpenMPI for message passing and ran the OpenMPI implementation on three Linux machines with the following specifications: (1) Intel Core i7-6850K 3.6GHz and 128GB RAM, (2) Intel Core i7-5960X 3.0GHz and 64GB RAM, and (3) Intel Xeon E5-1603 2.8GHz and 64GB RAM. All the machines were placed in the same LAN over 1000BASE-T Ethernet. Table 7 shows the run time of the distributed-memory parallel algorithm with 16 slaves. Note that we were not able to use this distributed environment for ameblo dataset due to its availability.[5] For the linear-space implementation, communication overheads by MPI dominate the processing time, resulting in the same or even worse performance than the sequential algorithm (Algorithm 1). However, thanks to the decentralization of disk access to multiple computing nodes, the distributed-memory algorithm with sublinear space achieved better performance than the shared-memory one, up to six times speed-up.

C.2 Dynamic updates without recomputation

We next discuss how to dynamically update our coarsened graph when the underlying influence graph changes, rather than rerunning our algorithm from scratch. For simplicity, we allow only *edge insertions* and *edge deletions*.

Our dynamic algorithm is presented in Algorithm 7. It maintains $\mathcal{H}$ and π for the *latest snapshot* of an influence graph, and further keeps and reuses G_i and $\mathcal{C}_i$ for $i \in [r]$ and $\mathcal{P}_r$. When an edge has been inserted to, or deleted from, the current influence graph $\mathcal{G}$, InsertEdge or DeleteEdge updates G_i's, $\mathcal{C}_i$'s, $\mathcal{P}_r$, $\mathcal{H}$ and π according to the graph change, in this order listed. To this end, we first update each G_i as follows.

- **Edge insertion:** Suppose that an edge (u,v) with edge probability p_{uv} has been inserted to $\mathcal{G}$. Then, we independently add (u,v) to each G_i with probability p_{uv}.
- **Edge deletion:** Suppose that an edge (u,v) has been deleted from $\mathcal{G}$. Then, we remove (u,v) from each G_i.

Next, we update each $\mathcal{C}_i$. If G_i has not been changed, then it is obvious that $\mathcal{C}_i$ would be the same; thus, we skip the SCC computation for G_i. Otherwise, $\mathcal{C}_i$ for the latest G_i may be different from that for the previous G_i; thus, we run an in-memory or a disk-based SCC algorithm on G_i to obtain the latest $\mathcal{C}_i$. Finally, we check whether none of $\mathcal{C}_i$ has been changed for all $i \in [r]$. If this is the case, then we can safely conclude that $\mathcal{P}_r$ would be the same, and we efficiently construct $\mathcal{H}$ for the latest snapshot of an influence graph (see Algorithm 7 for more details). Otherwise, we compute $\mathcal{P}_r$ and naively construct $\mathcal{H}$ and π, according to Definition 4.1, which can be implemented with either linear space (lines 6–12 in Algorithm 1) or sublinear space (lines 8–21 in Algorithm 2).

Remark that the insertion or deletion of an edge (u,v) affects only a p_{uv} fraction of random graphs in expectation. This implies that we run an SCC algorithm only for the same fraction of random graphs in expectation. As the value of p_{uv} is small in practice [3, 13], almost all of SCC computations will be pruned, which is a strong advantage of our dynamic algorithm in terms of computation time. More precisely, both InsertEdge and DeleteEdge with linear space can be done in time $\mathcal{O}(r|V| + (1 + rp_{uv})|E|)$ time, and both InsertEdge and DeleteEdge with sublinear space can be done in time $\mathcal{O}(r|V| + |E| + rp_{uv}\mathrm{t}_{\mathcal{A}}(V,E))$ time, where $\mathrm{t}_{\mathcal{A}}(V,E)$ is the time complexity of an SCC algorithm $\mathcal{A}$.

D. ADDITIONAL EXPERIMENTAL RESULTS

This section provides additional experimental results under UC and WC. Table 8 shows the run times and memory usages of both Algorithms 1 and 2. Table 9 shows the numbers of vertices and edges, and the corresponding reduction ratio. Table 10 shows the total run time and accuracy of each influence estimation method. Table 11 shows the run time and solution quality of each influence maximization method. Results for some datasets are omitted due to space limitations. For the UC setting, we observe similar trends to those for the EXP setting. On the other hand, our coarsening for WC is not much effective. It should be noted, however, that both influence estimation and influence maximization under WC can be solved quickly as shown in Tables 10 and 11.

Table 7: Run time of our parallel implementations for EXP.

	linear space (Alg.6 with impl. of Alg.1)				sublinear space (Alg.6 with impl. of Alg.2)			
	shared			**distributed**	**shared**			**distributed**
dataset	1 thread	4 threads	16 threads	16 threads	1 thread	4 threads	16 threads	16 threads
ca-HepPh	0.09 s	0.19 s	0.11 s	0.21 s	1.46 s	1.20 s	0.77 s	0.23 s
soc-Slashdot0922	0.54 s	1.01 s	0.63 s	1.28 s	5.57 s	3.88 s	1.79 s	1.01 s
web-NotreDame	1.77 s	3.43 s	1.73 s	2.27 s	10.86 s	8.24 s	3.85 s	2.31 s
wiki-Talk	42.31 s	25.70 s	12.39 s	24.26 s	57.37 s	42.09 s	22.08 s	18.35 s
com-Youtube	14.59 s	11.99 s	6.95 s	14.60 s	44.89 s	29.03 s	13.72 s	13.37 s
higgs-twitter	7.64 s	7.76 s	3.58 s	18.38 s	100.35 s	47.15 s	23.43 s	15.26 s
soc-Pokec	35.20 s	21.74 s	10.75 s	37.10 s	224.19 s	102.73 s	52.09 s	40.85 s
soc-LiveJournal1	94.58 s	60.97 s	33.45 s	102.64 s	507.54 s	244.71 s	127.36 s	106.69 s
com-Orkut	122.34 s	59.82 s	38.29 s	209.90 s	1,527.21 s	657.50 s	349.71 s	217.32 s
twitter-2010	1,762.93 s	1,017.31 s	611.98 s	1,751.02 s	11,522.38 s	4,923.69 s	2,619.67 s	1,928.31 s
com-Friendster	3,964.05 s	2,171.24 s	1,142.70 s	4,381.44 s	26,423.50 s	11,333.60 s	5,843.53 s	4,083.38 s
uk-2007-05	3,105.88 s	2,164.21 s	1,035.69 s	4,425.94 s	29,540.45 s	12,856.15 s	6,328.57 s	4,494.19 s
ameblo	OOM	OOM	OOM	–	35,761.37 s	23,074.51 s	7,612.53 s	–

Table 8: Run time and memory usage of the proposed algorithm under UC and WC.

	uniform cascade				**weighted cascade**			
	linear space (Alg.1)		sublinear space (Alg.2)		linear space (Alg.1)		sublinear space (Alg.2)	
dataset	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**	**run time**	**mem usage**
soc-Slashdot0922	0.53 s	58 MB	4.86 s	13 MB	0.48 s	68 MB	5.27 s	9 MB
wiki-Talk	37.89 s	603 MB	60.78 s	270 MB	38.76 s	545 MB	56.35 s	217 MB
higgs-twitter	7.68 s	562 MB	87.83 s	85 MB	4.49 s	666 MB	87.11 s	41 MB
soc-LiveJournal1	89.33 s	2,965 MB	452.94 s	676 MB	84.60 s	4,265 MB	492.49 s	430 MB
twitter-2010	1618.74 s	50,789 MB	10,419.08 s	5,629 MB	1,076.50 s	72,567 MB	9,955.61 s	3,403 MB
com-Friendster	3537.57 s	101,429 MB	22,659.64 s	7,711 MB	2,293.77 s	158,596 MB	28,065.90 s	5,203 MB
uk-2007-05	3103.58 s	136,572 MB	27,029.84 s	10,778 MB	2,491.07 s	165,428 MB	27,232.59 s	8,168 MB
ameblo	OOM	OOM	27,017.30 s	27,479 MB	OOM	OOM	6,860.45 s	20,488 MB

Table 9: Effect of the proposed algorithm on graph size under UC and WC. V and E denote vertex and edge sets of an input graph, and W and F denote vertex and edge sets of a coarsened graph, respectively.

	uniform cascade				**weighted cascade**			
dataset	$\|W\|$	$\|W\|/\|V\|$	$\|F\|$	$\|F\|/\|E\|$	$\|W\|$	$\|W\|/\|V\|$	$\|F\|$	$\|F\|/\|E\|$
soc-Slashdot0922	78,360	95.4%	317,110	36.4%	82,168	100.0%	870,161	100.0%
wiki-Talk	2,389,283	99.8%	3,102,167	61.8%	2,394,284	100.0%	5,021,203	100.0%
higgs-twitter	409,250	89.6%	4,367,703	29.4%	453,623	99.3%	14,846,862	99.9%
soc-LiveJournal1	4,514,953	93.1%	29,562,747	43.2%	4,838,026	99.8%	68,453,373	100.0%
twitter-2010	38,952,357	93.5%	359,675,952	24.5%	41,597,759	99.9%	1,468,218,916	100.0%
com-Friendster	47,022,949	71.7%	175,222,222	4.9%	65,608,362	100.0%	3,612,134,262	100.0%
uk-2007-05	102,506,082	97.4%	1,582,197,218	42.6%	105,197,183	100.0%	3,717,097,320	100.0%
ameblo	271,134,723	99.4%	5,489,262,364	79.4%	269,812,560	98.9%	6,837,663,021	98.9%

Table 10: Average influence estimation time for plain *MC* and our framework with *MC* under UC and WC. MARE and RCC stand for "mean absolute relative error" and "rank correlation coefficient," respectively.

	uniform cascade					**weighted cascade**				
	run time			**accuracy**		**run time**			**accuracy**	
dataset	*MC*	Alg.3(*MC*)	$\frac{\text{Alg.3}(MC)}{MC}$	MARE	RCC	*MC*	Alg.3(*MC*)	$\frac{\text{Alg.3}(MC)}{MC}$	MARE	RCC
soc-Slashdot0922	35.2 s	8.7 s	24.6%	0.0106	0.9953	27.4 ms	25.9 ms	94.6%	0.0000	1.0000
wiki-Talk	10.1 s	6.1 s	60.5%	0.0006	0.9999	5.2 ms	4.5 ms	87.9%	0.0088	1.0000
higgs-twitter	1,124.0 s	252.9 s	22.5%	0.0103	0.9951	104.3 ms	90.2 ms	86.5%	0.0302	0.9991
soc-LiveJournal1	4,293.8 s	1,627.2 s	37.9%	0.0125	0.9993	45.6 ms	52.5 ms	115.0%	0.0271	0.9988
twitter-2010	105,743.0 s	21,889.5 s	20.7%	–	–	42.9 ms	44.7 ms	104.2%	–	–
com-Friendster	450,452.0 s	18,145.4 s	4.0%	–	–	583.2 ms	583.7 ms	100.1%	–	–
uk-2007-05	4,555.2 s	1,813.6 s	39.8%	–	–	56.4 ms	57.7 ms	102.2%	–	–

Table 11: Run time for selecting a seed set of size 100 and solution quality for plain *D-SSA* and our framework with *D-SSA* under UC and WC. OOM denotes "out of memory."

	uniform cascade					**weighted cascade**				
	run time			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\|V\|$)		**run time**			**influence** ($\mathsf{Inf}_{\mathcal{G}}/\|V\|$)	
dataset	*D-SSA*	Alg.4(*D-SSA*)	$\frac{\text{Alg.4}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.4(*D-SSA*)	*D-SSA*	Alg.4(*D-SSA*)	$\frac{\text{Alg.4}(D\text{-}SSA)}{D\text{-}SSA}$	*D-SSA*	Alg.4(*D-SSA*)
soc-Slashdot0922	291.9 s	84.3 s	28.9%	0.2958	0.2959	1.2 s	1.2 s	97.7%	0.2653	0.2653
wiki-Talk	266.1 s	94.9 s	35.6%	0.1394	0.1394	2.3 s	2.3 s	98.5%	0.4038	0.4041
higgs-twitter	3,837.2 s	1,317.1 s	34.3%	0.3511	0.3511	6.7 s	6.9 s	102.8%	0.0654	0.0654
soc-LiveJournal1	OOM	OOM	–%	–	–	25.0 s	26.0 s	104.2%	0.0222	0.0222
twitter-2010	OOM	OOM	–%	–	–	50.5 s	80.2 s	158.8%	0.2582	0.2586
com-Friendster	OOM	OOM	–%	–	–	5,297.2 s	5,220.4 s	98.5%	0.0040	0.0040
uk-2007-05	OOM	OOM	–%	–	–	81.9 s	92.7 s	113.2%	0.0350	0.0351