Scalable and Parallelizable Processing of Influence Maximization for Large-Scale Social Networks*

Jinha Kim, Seung-Keol Kim, Hwanjo Yu#

Department of Computer Science and Engineering,
Pohang University of Science and Technology(POSTECH)
San 31, Hyoja-dong, Pohang, South Korea

{goldbar,azzibobj,hwanjoyu}@postech.ac.kr

***Abstract*—As social network services connect people across the world, influence maximization, i.e., finding the most influential nodes (or individuals) in the network, is being actively researched with applications to viral marketing. One crucial challenge in scalable influence maximization processing is evaluating influence, which is #P-hard and thus hard to solve in polynomial time. We propose a scalable influence approximation algorithm, *Independent Path Algorithm (IPA)* for Independent Cascade (IC) diffusion model. IPA efficiently approximates influence by considering an independent influence path as an influence evaluation unit. IPA are also easily parallelized by simply adding a few lines of OpenMP meta-programming expressions. Also, overhead of maintaining influence paths in memory is relieved by safely throwing away insignificant influence paths. Extensive experiments conducted on large-scale real social networks show that IPA is an order of magnitude faster and uses less memory than the state of the art algorithms. Our experimental results also show that parallel versions of IPA speeds up further as the number of CPU cores increases, and more speed-up is achieved for larger datasets. The algorithms have been implemented in our demo application for influence maximization (available at http://dm.postech.ac.kr/ipa_demo), which efficiently finds the most influential nodes in a social network.**

***Index Terms*—Influence maximization, social networks, parallel processing**

I. Introduction

Nowadays, the influence of acquaintances' opinions is boosted by explosive exposure to social networks. Social network services such as blogs, Facebook and Twitter connect individuals across the world. As a consequence, people in social networks influence each other in both direct and indirect ways [1]. Such influence exchange in social networks is called the *"word-of-mouth" effect* [2].

To fully exploit the "word-of-mouth" effect in marketing, influence maximization addresses the most influential entities – seed nodes – in social networks. While the influence maximization problem was introduced by Domingos and Richardson in a probabilistic perspective [3, 4], Kempe et al. established it as a discrete optimization [5] and their formulation is now actively studied in the database and data mining community [1, 5–13].

In a discrete perspective, a social network is abstracted to a weighted directed graph $G(V, E, W)$ where nodes (V) are individuals, edges (E) are connections between individuals, and edges' weights (W) are the probabilities that a node influences another node. The weights (W) can be pre-defined or learned from the past action log [11–13].

Given a graph, influence is propagated from seed nodes to all the other nodes by following an influence diffusion model such as independent cascade (IC) model. The solution to the influence maximization problem is the finding of the top-k seed nodes such that the total influence spread propagated from the seed nodes is maximized. (k represents the budget for persuading seed nodes in viral marketing.)

However, making influence maximization processing effective and scalable is proving quite challenging. At the macro level, finding the optimal seed nodes is NP-hard [5], and thus the greedy algorithm [5] approximates $1 - 1/e$ ratio of the optimal solution. Nevertheless, at the micro level, evaluating influence has two challenges to scalability: (1) enormous influence evaluations occupy most of the processing time, and (2) a single influence evaluation itself is #P-hard [6], which is also hard to solve in polynomial time. The CELF greedy algorithm [9] reduces the number of influence evaluations. To boost a single influence evaluation, Chen et al. [7] use the property of the independent cascade model; Wang et al. [8] break the whole social network into several communities; and Chen et al. [6] use local arborescences of the most probable influence paths between two nodes.

This paper proposes the scalable influence approximation algorithms called *Independent Path Algorithms (IPAs)* for IC model. IPA treats an influence path from a seed node to a non-seed node as an influence evaluation unit and assumes that influence paths are independent of each other. Then, influence approximation only requires a series of simple arithmetic of influence paths. Consequently, after collecting influence paths, IPA and IPA-N require almost no additional time to approximate influence spread.

In addition, IPA is highly parallelizable. Embedding multiple cores in a CPU is considered a breakthrough that improves CPU performance, which was limited by excessive power

*This research was partially supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2012M3C4A7033344). This work was also supported by IT Consilience Creative Program of MKE and NIPA (C1515-1121-0003).

#corresponding author

consumption and a heating problem [14]. As a consequence, to exploit the performance improvement of modern CPUs, parallelizable algorithms are now not optional. By considering an influence path as a fine-grained influence evaluation unit, most sections of IPA are parallelizable by organizing influence paths appropriately and adding a few lines of OpenMP expressions [15].

IPA is also memory efficient. A naive integration of IPA and the CELF greedy algorithm requires the maintenance of an excessive number of influence paths in memory. However, the isolated influence path usage of IPA and the lazy evaluation characteristic of the CELF greedy algorithm make it possible to discard insignificant influence paths.

Our extensive experiments on five real datasets of social network graphs comprising millions of nodes indicate the following:

- IPA is an order of magnitude faster than the state of the art algorithm PMIA [6]. In particular, once IPA-integrated CELF greedy finds the first seed node, it finds the remaining seed nodes in less than one second regardless of datasets, while PMIA-integrated CELF greedy takes a lot of time to find the second through the last seed nodes.
- IPA uses much less memory than PMIA; IPA successfully produces results on large-scale datasets comprising millions of nodes using less than $4GB$ of memory, whereas PMIA fails due to its excessive memory usage (over $24GB$).
- IPA accurately approximates influence spread; influence spread of the IPA solution is close to that of the Greedy solution obtained by $20,000$ times Monte-Carlo simulations, and is larger than the influence spread of the PMIA solution overall.
- The parallel version of IPA speeds up further as the number of CPU cores increases, and more speed-up is achieved for larger datasets.

IPA is implemented in our demo application for influence maximization, which efficiently finds the most influential nodes in a social network. The application is available at http://dm.postech.ac.kr/ipa_demo.

The rest of this paper is organized as follows. Section II presents the influence maximization problem, IC model, and existing algorithms related to influence maximization. Section III introduces our influence approximation algorithm IPA as equations. Section IV describes IPA algorithmically and its parallelization. Section V presents a memory efficient processing method for IPA. Section VI reports our experimental results on real datasets. Section VII concludes this paper.

II. Preliminary and Related Work

A. Independent Cascade(IC) Model

We first introduce the independent cascade (IC) model. Each influence diffusion model focuses on specific aspects of real influence propagation, because we do not know the exact dynamics of the influence diffusion, and embedding all known aspects is too complicated. As one of the representative influence diffusion models, IC model assumes that influence propagates by means of one-to-one persuasion. The simple and clear dynamics of IC model attracts many researchers, and it is presently the most actively studied influence diffusion model [5–9].

To explain how an influence diffusion model works, a social network is first abstracted as a weighted directed graph as defined by Definition 1. Nodes represent entities such as authors, customers, and products. Edges are connections among the nodes that influence each other. The probability that a node influences another is embedded in an edge's weight.

Definition 1: A social network graph is defined as $G(V, E, W)$. V is a set of nodes. $E = \{(v_i, v_j)|v_i, v_j \in V\}$ is a set of edges. For $(v, u) \in E$,$w_{(v,u)} \in W$ is the influence propagation probability from v to u which ranges over $(0, 1]$.

In the IC model, a node has one of two states – *active* or *inactive*. The active state signifies that a node is either a seed node or has been activated by its neighbors. The inactive state signifies that a node has not yet had a chance to be influenced by its neighbors.

The IC model in a weighted directed graph G works in an inductive way. At an initial stage 0, $S \subseteq V$ is selected as a seed set. A_t is the set of nodes activated at stage t and $A_0 = S$. At stage $t+1$, each node $v \in A_t$ has a single chance to propagate influence to its neighbors $u \notin \bigcup_{i=0}^{t} A_i$ with probability $w_{(v,u)}$. When $A_{t+1} = \phi$, influence propagation is finished.

B. Assigning propagation probability

In most of the literature, an edge's weight, propagation probability, is defined by assumption; it is either globally constant (e.g., 0.01), one of the pre-defined constants (e.g., one of $\{0.1(\text{high}), 0.01(\text{medium}), 0.001(\text{low})\}$), or the reverse of the in-degree.

Learning the propagation probability of an edge is also an actively researched topic. Saito et al. [13] learn the propagation probabilities in the IC model from the past propagation log. Goyal et al. [12] deal with the same problem in a temporal manner – when a user perform an action. Goyal et al. [11] directly generate influence spread from the action log, without learning the propagation probabilities by simulating/approximating the influence diffusion process.

As our goal is to efficiently approximate influence spread, learning propagation probability is out of scope in this paper.

C. Problem Definition

While the influence maximization problem informally addresses the most influential entities, it should be formalized so that it can be dealt with in an algorithmic way. Kempe et al. [5] dealt with the influence maximization problem in a discrete perspective and their problem setting has been actively studied [5–9, 16, 17]. This paper also follows the setting of [5].

With a weighted directed graph $G(V, E, W)$, the influence maximization problem is formalized as follows.

Definition 2: The influence maximization problem addresses the top-k seed node set S from a graph G that satisfies

$$S = \arg \max_{T \subseteq V, |T|=k} \sigma(T, G) \quad (1)$$

where $\sigma(T, G) : V \times G \rightarrow \mathbb{R}$ is a function of a node set that provides the expected number of the influenced nodes from a node set T in a graph G.

With Definition 2, the influence maximization problem becomes an instance of a combinatorial optimization problem. (From here onwards, we call the expected number of the influenced nodes *influence spread* and omit the obvious parameter G from $\sigma(T, G)$.)

Even for the same graph, the seed set of the influence maximization varies according to the underlying influence diffusion model. Different influence diffusion models result in different influence spread, because it determines how a node is activated (influenced). The influence spread of a seed set S in the IC model – $\sigma_I(S)$ – is the expected number of active state nodes. *positive* state nodes.

D. Existing Solutions

Obtaining the optimal solution to the influence maximization problem intuitively requires an exponential search space. For a graph of N nodes, to get top-k seed nodes (usually $k \ll N$), we need to check ${}_N C_k = \frac{N!}{k!(N-k)!} \approx N^k$ cases. Theorem 1 shows that it is impossible to get the optimal solution in polynomial time under the prevalent assumption of P≠NP. The greedy algorithm [5] alleviates NP-hardness and our algorithm starts from the scalability challenge of the greedy algorithm.

Theorem 1: The influence maximization problem of Definition 2 with IC model is NP-hard.

Proof: Theorem 2.4 of [5]. ■

Greedy algorithm: To overcome the NP-hardness of the influence maximization problem, the greedy algorithm [5] was proposed. The greedy algorithm repeatedly selects the node that gives the largest marginal increase of influence spread, so that it approximates the optimal solution with the lower bound ratio $1 - 1/e \approx 0.63123$. To exploit the greedy hill-climbing approach of the greedy algorithm, an object function $f(\cdot)$ should satisfy three conditions: (1) non-negativity, i.e., $\forall(S \subseteq V).f(S) \geq 0$; (2) monotonicity, i.e., for $S \subseteq T \subseteq V$, $f(S) \leq f(T)$; and (3) sub-modularity, i.e., the marginal gain diminishes as the set size increases, which is formalized as the following equation.

$$\begin{array}{l} \text{For } S \subseteq T \subseteq V, v \in V, \text{ and } v \notin T \\ f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T) \end{array} \quad (2)$$

Kempe et al. [5] showed that $\sigma_I(S)$ satisfies the three conditions.

CELF greedy algorithm: Although the greedy algorithm detours the NP-hardness of the influence maximization problem via approximation, it still suffers from the scalability problem. On the naive application of the greedy algorithm, $\sum_{i=1}^{k}(N - i + 1) = k(2N - k + 1)/2$ influence evaluations are required to find the top-k seed nodes, where $N = |V|$. Although the number of influence evaluations is linear to N, the naive application of the greedy algorithm requires too many influence evaluations in that real social networks have over millions of nodes. To rectify this, Leskovec et al. [9] proposed the CELF greedy algorithm, which drastically reduces the number of influence evaluations by exploiting the sub-modularity of $\sigma(T)$.

Algorithm 1 $CELFgreedy(G, \sigma, k)$

Require: G : graph, k : the number of seed nodes
Ensure: the seed set of size k

```
1:  S ← φ
2:  for v ∈ V do evaluate σ({v})
3:  for v ∈ V do pq.push(σ({v}), v)
4:  for i = 1 to k do
5:    while do
6:      (inc_u, u) ← pq.pop()
7:      ninc_u ← σ(S ∪ {u}) − σ(S)
8:      (inc_w, w) ← pq.top()
9:      if ninc_u ≥ inc_w then
10:       S ← S ∪ {u}
11:       break
12:     else
13:       pq.push(ninc_u, u)
14:     end if
15:   end while
16: end for
17: return S
```

The CELF greedy algorithm is outlined in Algorithm 1. In the CELF greedy algorithm, a priority queue pq maintains the upper bound of marginal influence spread increase of each node. When a node u is popped (line 6), its upper bound is updated into $ninc_u$ by considering the current seed set S (line 7). If $ninc_u$ is larger than inc_w, i.e., the upper bound of the top node w, u is guaranteed to be a seed node without seeing the remaining nodes in the priority queue (lines 9-11).

Scalable influence spread evaluation: An even worse challenge to scalable influence maximization processing is exact evaluation of $\sigma(S)$. Chen et al. [6] proved that evaluating $\sigma(S)$ is #P-hard, and thus cannot be completed in polynomial time. Moreover, although the CELF greedy algorithm reduces the number of influence evaluations, most of the influence maximization processing time is taken up by a series of $\sigma(S)$ evaluations (lines 2 and 7 of Algorithm 1).

In early studies of the influence maximization problem [5], [9], the result of Monte-Carlo simulations of influence propagation was averaged to get influence spread. However, to get stable $\sigma(S)$, a large number of Monte-Carlo simulations (e.g. $20,000$) were required, which took an intolerably long time. Chen et al. [7] reduced the processing time of line 2 of Algorithm 1 using simultaneous simulations of influence propagation. Wang et al. [8] reduced the influence propagation space by breaking down a graph into several communities.

However, [7] and [8] still depend on Monte-Carlo simulations and thus cannot be scaled to graphs comprising millions of nodes. Chen et al. [6] proposed the state of the art PMIA that approximates $\sigma(S)$. PMIA constructs local influence arborescences based on the most probable influence path between two nodes. Based on local arborescences, PMIA boosts $\sigma(S)$ evaluation time up-to several orders of magnitudes. However, it still takes a very long time on graphs comprising millions of nodes, has inter-dependency between local arborescences, and requires cascading updates of local arborescences when getting marginal influence spread increase (line 7 of Algorithm 1).

E. OpenMP Parallelization Environment

OpenMP [15] is a well-known shared-memory parallel programming environment. In typical programming environments such as POSIX pthread, to fully utilize multi-core CPU resources, heavy code modification of serial programming is required, such as thread forking, thread management, and synchronization. OpenMP transforms serial programs into parallel programs that exploit multi-core CPUs by simply adding a few meta-programming expressions. As a simple example, the following C fragment

```
#pragma omp parallel for
for( i = 0; i < N; i++ )
  f(d[i]);
```

enables parallel execution of `f(d[i])` in the for-loop using only one line of pragma expression. However, to fully utilize OpenMP, serial programs should be designed with sophistication – synchronization blocks should be avoided and each serial execution unit should as much as possible take even processing time. Our algorithms are carefully designed to make use of OpenMP parallelization capability.

III. Independent Path Algorithm

In this section, we propose our influence spread approximation algorithms IPA under the IC model. The main idea underlying IPA is that each influence path is considered an influence spread evaluation unit and influence paths are independent of each other. Our fine-grained influence evaluation unit makes influence spread approximation more scalable.

IPA first decomposes $\sigma_I(S)$ evaluation into an inductive procedure to incorporate with the CELF greedy algorithm. As a base case, for all $v \in V$, a single node influence spread $\sigma_I(\{v\})$ is approximated by gathering influence paths starting from v (Section III-A). As an inductive case, marginal influence spread increase $\sigma_I(S \cup \{v\}) - \sigma_I(S)$ is approximated by selecting valid influence paths (Section III-B).

A. Influence Spread of a Node

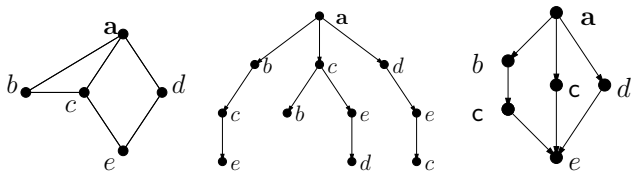
We efficiently approximate the influence spread of a single node v ($\sigma_I(\{v\})$) by controlling the number of influence paths from v. Informally, the essence of the #P-hardness of the influence spread evaluation is that we cannot find all influence paths between two nodes in tractable time. Therefore, we boost the processing time for influence spread evaluation by controlling the number of influence paths.

Before describing our approach, an influence path and its influence propagation probability are defined as follows. An influence path p from a node v to a node $u(\neq v)$ is expressed as $p = \langle v_1 = v, v_2, \cdots, v_m = u \rangle (m \geq 2)$ and the influence propagation probability of p in the IC model is

$$ipp(p) = \prod_{i=1}^{m-1} w_{(v_i, v_{i+1})} \tag{3}$$

To control the number of influence paths, influence paths are collected whose influence propagation probabilities are no less than a pre-defined threshold θ. To illustrate, consider the following example.

Example 1: [Influence path collection starting from **a**] Suppose (1) a graph is Figure 1a, (2) the influence diffusion model is IC, (3) a propagation probability of 0.1 is uniformly assigned to every edge, and (4) the threshold $\theta = 0.001$.



(a) Graph (b) Path expansion tree of **a** (c) Paths from **a** to e

Fig. 1. Path collection starting from **a**

Influence paths are gathered by repeatedly expanding the outgoing edge of each existing path's last node. Path expansion is continued until (1) the influence propagation probability of a path is less than a pre-defined threshold θ, or (2) a cycle is detected on a path. θ controls the maximum length of an influence path. Cycles are not allowed in the IC model because cycles mean that active nodes has been reactivated. Figure 1b shows the resultant path expansion tree from **a**. All ten paths from the root **a** to the other nodes in the tree are influence paths starting from **a** and are used to approximate $\sigma_I(\{\mathbf{a}\})$.

Using the collected influence paths starting from node v, the approximated influence spread of v is calculated by summing the influence spread of v to each node influenced by v. Let us denote the influence path from v to u as $P_{v \to u} = \{p | p = \langle v, \cdots, u \rangle\}$ and that from a node set $T \subseteq V$ to u as $P_{T \to u} = \{p | p = \langle v, \cdots, u \rangle, v \in T\}$. $P_{v \to T}$ and $P_{S \to T}$ are trivial. The influenced area of node v is $O_v = \{u | \langle \cdots, u \rangle \in P_{v \to V}\}$ and that of node set $T \subseteq V$ is $O_T = \{u | \langle \cdots, u \rangle \in P_{T \to V}\}$. The approximated influence spread of v is

$$\hat{\sigma_I}(\{v\}) = 1 + \sum_{u \in O_v} \hat{\sigma_I}^u(\{v\}) \tag{4}$$

in the IC model. The term 1 in Equation 4 is the influence of v to itself and $\hat{\sigma_I}^u(\{v\})$ is the approximated influence spread

of v to a specific node $u \in O_v$. Given $P_{v\to u}$, $\hat{\sigma_I}^u(\{v\})$ is the complement of the probability that no path in $P_{v\to u}$ activates u, which is expressed as follows.

$$\hat{\sigma_I}^u(\{v\}) = 1 - \prod_{p \in P_{v\to u}} (1 - ipp(p)). \quad (5)$$

In Equation 5, for fast evaluation we ignore dependencies between influence paths. In Example 1, although $\langle a, c, e\rangle$ and $\langle a, b, c, e\rangle$ have intermediate node c in common, they are considered as independent influence paths, as shown in Figure 1c and thus the influence spread of $\mathbf{a}$ to e becomes $\hat{\sigma_I}^e(\{\mathbf{a}\}) = 1-\{(1-0.0001)(1-0.001)(1-0.001)\} = 0.0021$ by Equation 5.

B. Marginal Influence Spread Increase

Although computing marginal influence spread increase $\Delta_I(S, v) = \sigma_I(S \cup \{v\}) - \sigma_I(S)$ is complicated due to a characteristic of the IC model, getting $\Delta_I(S, v)$ is easily approximated in our approach. The difficulty in getting marginal influence spread increase is that $\sigma_I(S \cup \{v\}) \neq \sigma_I(S)+\sigma_I(\{v\})$ due to the influence path blocking of different seed nodes. Nevertheless, with the influence path collection described in Section III-A, approximated marginal influence spread increase $\hat{\Delta}_I(S, v) = \hat{\sigma_I}(S \cup \{v\}) - \hat{\sigma_I}(S)$ is obtained by filtering out invalid influence paths.

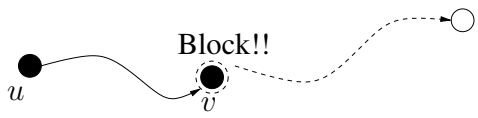


Fig. 2. A situation in which a seed node v blocks the influence of the other seed node u. (black circle : seed node, white circle : non-seed node)

In the IC model, $\sigma_I(\{v\})$ is not equal to $\Delta_I(S, v)$ because of the influence path blocking. For the influence paths $\langle \cdots, v, \cdots\rangle \in P_{S\to V}$, they are no longer valid when v is added as a new seed because v blocks the influence of other seed nodes $u \in S$. Any influence path to a node $u \in S$ cannot change u's active status because in the IC model once a node is activated it keeps its active status until the end of influence propagation. Figure 2 shows an influence path from existing seed node $u \in S$ becoming invalid because of new seed node v. In addition, influence paths $\forall u \in S.(\langle v, \cdots, u, \cdots\rangle \in P_{v\to V})$ are also invalid for the same reason. Accordingly, such influence paths should not be included in the evaluation of $\hat{\Delta}_I(S, v)$.

Thus, the valid influence path set from S to $u \in O_S$ is defined as follows:

$$P^{valid}_{S\to u} = \{p | p \in P_{S\to u}, |p \cap S| = 1\}. \quad (6)$$

The $|p \cap T| = 1$ requirement tells us that only one seed node in p is allowed as a starting node to be a valid influence path.

With the influence path collection, marginal influence spread increase is efficiently approximated. For each node $v \in V$, we have all influence paths starting from v, $P_{v\to V}$, in the base case of getting $\hat{\sigma_I}(\{v\})$. Thus, filtering out invalid(blocked) influence paths under the new seed set $S \cup \{v\}$ is executed by simple set membership operations.

Moreover, computing $\hat{\Delta}_I(S, v)$ does not require all influence paths of $\bigcup_{u\in S\cup\{v\}} P_{u\to V}$. When computing differences, common factor does not have to be considered. In evaluating $\hat{\Delta}(S, v)$, influence paths $\{p | p = \langle u, \cdots, w\rangle, u \in S, w \notin O_v \cup \{v\}\}$ are commonly used. Thus, for all $w \notin O_v \cup \{v\}$, $\hat{\sigma_I}^w(S)$ and $\hat{\sigma_I}^w(S \cup \{v\})$ have the same value. All paths in $P_{S\to v}$ are blocked by the newly added seed node v. Accordingly, considering influence paths $P_{S\cup\{v\}\to O_v\cup\{v\}}$ is enough to evaluate marginal influence spread increase.

Thus, $\hat{\Delta}_I(S, v)$ which excludes the common influence paths of $\hat{\sigma_I}(S \cup \{v\})$ and $\hat{\sigma_I}(S)$ is derived as follows:

$$\begin{aligned}
& \hat{\Delta}_I(S, v) \\
= \; & \hat{\sigma_I}(S \cup \{v\}) - \hat{\sigma_I}(S) \\
= \; & |S \cup \{v\}| + \sum_{u\in O_{S\cup\{v\}}} \hat{\sigma_I}^u(S \cup \{v\}) - |S| - \sum_{u\in O_S} \hat{\sigma_I}^u(S) \\
= \; & 1 + \sum_{u\in O_{S\cup\{v\}}} \hat{\Delta}^u_I(S, v) \quad (\because \sum_{u\in O_{S\cup\{v\}}\setminus O_S} \hat{\sigma}^u_I(S) = 0) \\
= \; & 1 + \cancel{\sum_{u\in O_S\setminus(O_v\cup\{v\})} \hat{\Delta}^u_I(S, v)}(= 0) + \sum_{u\in O_v\cup\{v\}} \hat{\Delta}^u_I(S, v) \\
= \; & 1 + \sum_{u\in O_v\cup\{v\}} \hat{\Delta}^u_I(S, v).
\end{aligned}$$

The term 1 in Equation 7 is the influence of new seed v to itself. $\hat{\sigma_I}^u(S)$ is the influence spread from seed node set S to a single node u.

$$\hat{\sigma_I}^u(S) = 1 - \prod_{p \in P^{valid}_{S\to u}} (1 - ipp(p)). \quad (7)$$

$\hat{\Delta}^u_I(S, v)$ is marginal influence spread increase that affects u.

$$\hat{\Delta}^u_I(S, v) = \hat{\sigma_I}^u(S \cup \{v\}) - \hat{\sigma_I}^u(S). \quad (8)$$

IV. Parallelizable IPA Processing

This section describes the parallel versions of IPA. Parallel IPA processing consists of three phases – traversing, re-organizing, and updating. With the idea of decomposing influence spread into combinations of independent influence paths, traversing and updating phases are easily parallelized.

A. Traversing Phase

In the traversing phase, $\hat{\sigma}_I(\{v\})$ evaluations are executed in parallel. To evaluate $\hat{\sigma}_I(\{v\})$ for a single node v, all we need are influence paths starting from v, $P_{v\to V}$, as shown in Equations 4 and 5. $P_{v\to V}$ is obtained by traversing graph from v until two stopping conditions – threshold θ and a cycle – are met. For $v \neq u, u, v \in V$, $P_{v\to V}$ and $P_{u\to V}$ have no common influence paths because $p \in P_{v\to V}$ and $p' \in P_{u\to V}$ have different starting points. Accordingly, no interaction occurs between evaluating $\hat{\sigma}_I(\{v\})$ and evaluating $\hat{\sigma}_I(\{u\})$. Each serial execution unit of the parallel architecture (e.g., each physical core in the multi-core CPU) is assigned a $\hat{\sigma}_I(\{v\})$ evaluation task which is illustrated as dashed line box in

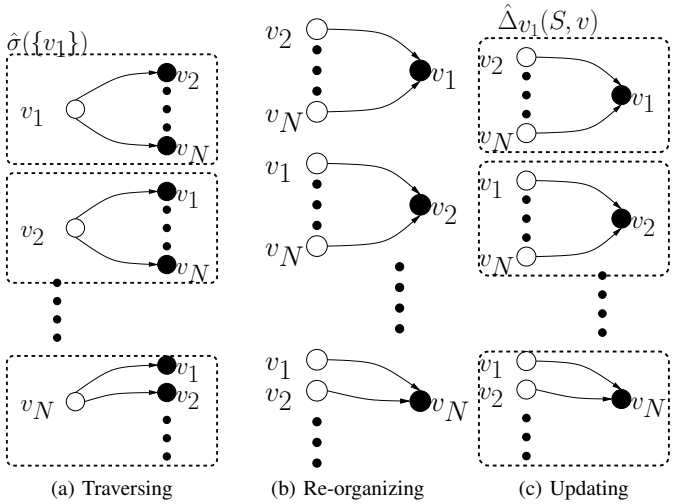


Fig. 3. Three phases of parallel processing scheme for influence maximization. (Dashed line box : independently executable block)

Figure 3a. Such tasks can be executed in parallel without any interaction with the other serial execution units. Algorithm 2 outlines the traversing phase and traversing(·) replaces line 2 of Algorithm 1.

Algorithm 2 traversing(G,θ)

Require: G : graph, θ : threshold
Ensure: R : set of $\hat{\sigma}_I(\{v\})$
1: $R \leftarrow \phi$
2: **parallel for** $v \in V$ **do**
3: $\quad \hat{\sigma}_I(\{v\}) \leftarrow 1$
4: $\quad$ generate $P_{v\to V}$ by traversing G with θ
5: $\quad$ **for** $u \in O_v$ **do**
6: $\quad\quad \hat{\sigma}_I(\{v\}) \leftarrow \hat{\sigma}_I(\{v\}) + \hat{\sigma}^u_I(\{v\})$
7: $\quad$ **end for**
8: $\quad R \leftarrow R \cup \{\hat{\sigma}_I(\{v\})\}$
9: **end for**
10: **return** R

The time complexity of traversing(·) is O($\frac{|V|}{c} \cdot |O_v| \cdot n_{vu}$) where n_{vu} is the average number of influence paths between two nodes. For each parallel for-loop iteration (lines 3–7), $O(|O_v| \cdot n_{vu})$ time is required. Getting $P_{v\to V}$ requires O($|O_v| \cdot n_{vu}$), which amounts to graph traversing time and $\sigma(\{v\})$ evaluation also requires O($|O_v| \cdot n_{vu}$). When c serial execution units are available, the time complexity of getting $P_{v\to V}$ and $\hat{\sigma}_I(\{v\})$ for all $v \in V$ becomes O($\frac{\lceil V \rceil}{c} \cdot |O_v| \cdot n_{vu}$). Even though synchronization is needed in line 8 of Algorithm 2, it is ignorable because in most cases the finishing time for each serial execution unit is different.

B. Re-organizing Phase

In the re-organizing phase, influence paths generated in the traversing phase are re-organized. In the traversing phase, influence paths obtained by traversing a graph are grouped by their starting nodes. In the re-organizing phase, influence paths are re-grouped by their ending nodes. The re-organization of the influence paths is more clearly presented as the following equation:

$$\begin{aligned} P \quad &= \quad \{P_{v\to V} | P_{v\to V}, v \in V\} \\ &\quad\quad \text{in traversing phase} \\ &\underset{re-organize}{\to} \quad \{P_{V\to v} | P_{V\to v}, v \in V\} \end{aligned} \tag{9}$$

By organizing influence paths in this way, parallel execution in the updating phase is ready. Figure 3b shows the result of the re-organizing phase from the traversing phase of Figure 3a. This phase is a kind of bridge from parallel evaluations of $\forall v \in V.\hat{\sigma}_I(\{v\})$ to parallel evaluation of $\hat{\Delta}_I(S,v)$ for a $v \in V$.

The time complexity of the re-organizing phase is O($|V| \cdot |O_v| \cdot n_{vu}$) because one scan of influence paths is sufficient for re-organizing influence paths. However, parallelization cannot be achieved due to the global re-ordering of the influence paths.

C. Updating Phase

In the updating phase, a marginal influence spread increase $\hat{\Delta}_I(S,v)$ is evaluated in parallel. In the CELF greedy algorithm processing, simultaneous evaluations of marginal influence spread increase are impossible. This is due to the fact that marginal influence spread increase of the popped node should be checked to determine whether the popped node is a seed node or not (lines 9–14 of Algorithm 1). Thus, the parallelization of the updating phase should be applied to a single evaluation of $\hat{\Delta}_I(S,v)$.

Algorithm 3 updating(S,v)

Require: S : current seed nodes, v : new seed node
Ensure: $\hat{\Delta}(S,v)$
1: $ret \leftarrow 1$
2: **parallel for** $u \in O_v \cup \{v\}$ **do**
3: $\quad new \leftarrow 1; old \leftarrow 1$
4: $\quad$ **for** $p \in P_{v\to u}$ and $p \cap S = \phi$ **do**
5: $\quad\quad new \leftarrow new \times (1 - ipp(p))$
6: $\quad$ **end for**
7: $\quad$ **for** $s \in S$ **do**
8: $\quad\quad$ **for** $p \in P_{s\to u}$ and $p \cap S \subseteq \{s,v\}$ **do**
9: $\quad\quad\quad old \leftarrow old \times (1 - ipp(p))$
10: $\quad\quad\quad$ **if** $p \cap S = \{s\}$ **then**
11: $\quad\quad\quad\quad new \leftarrow new \times (1 - ipp(p))$
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: $\quad ret \leftarrow ret + (1 - new) - (1 - old)$
16: **end for**
17: **return** ret

The parallelization of the $\hat{\Delta}_I(S,v)$ evaluation is done by evaluating $\hat{\Delta}^u_I(S,v)$ of Equation 8 in parallel. $\hat{\Delta}^u_I(S,v)$ can be executed in an independent serial execution unit because (1) influence paths are grouped by their ending nodes after the

re-organizing phase and (2) $\hat{\Delta}^u_I(S, v)$ evaluation only exploits influence paths that end at u. Each dashed line box in Figure 3c shows an independent evaluation block of $\hat{\Delta}^u_I(S, v)$ in the updating phase.

Algorithm 3 outlines the updating phase and replaces line 7 of Algorithm 1. updating($\cdot$) initializes ret as 1, which represents the influence of v to itself. For $u \in O_v \cup \{v\}$, as described in Section III-B, the parallelizable loop is executed (lines 2–16). In the parallelizable loop, invalid influence paths from $S \cup \{v\}$ are filtered out by checking their validity for two circumstances of evaluating $\hat{\sigma}^u_I(S \cup \{v\})$ and $\hat{\sigma}^u_I(S)$.

When c serial execution units are available, the time complexity of updating($\cdot$) is O($\frac{|O_v \cup \{v\}|}{c} \cdot n_{vu} + (c-1)$) in the worst case, where n_{vu} is the average number of influence paths between two nodes. For each parallelizable loop, one scan of $P_{v \to u}$ is required for the first inner loop (lines 4–6) and $|S|$ scans of $P_{s \to u}$ are required for the second inner loop (lines 7–14). Thus, O(n_{vu}) is required considering that $|S|$ is a small number. When all c execution units finish simultaneously, an additional $c-1$ synchronization operations are required for line 15. Finally, the complexity of updating($\cdot$) becomes O($\frac{|O_v \cup \{v\}|}{c} \cdot n_{vu} + (c-1)$).

V. Efficient Memory Handling

In this section, we propose a memory usage reduction scheme for the IPA-integrated CELF greedy algorithm. When the IPA algorithm is naively used in the CELF greedy algorithm to approximate influence spread, storing all influence paths P of Equation 9 is required. However, the lazy evaluation characteristic of the CELF greedy algorithm and the isolated influence path usage of IPA enable safe discarding of most influence paths generated in the traversing phase.

A problem of IPA is excessive memory usage. When the IPA integrated CELF algorithm is implemented naively, for all $v \in V$, $(\hat{\sigma}_I(\{v\}), v)$ tuple is inserted into priority queue (line 5 of Algorithm 1), and the corresponding influence paths of $\hat{\sigma}_I(\{v\})$, $P_{v \to V}$, should be maintained for $\hat{\Delta}(S, v)$ evaluation. Accordingly, all generated infuence path $\bigcup_{v \in V} P_{v \to V}$ take memory space until the end of the CELF greedy algorithm processing.

In the CELF greedy algorithm, not all entries are popped and evaluated to get marginal influence spread increase. To illustrate, consider the following example.

Example 2: An example of the CELF greedy algorithm processing which finds the top-2 seed nodes.

Top in priority queue				Bottom	S	operation
(10,a)	(8,b)	(6,c)	(3,d)	$\cdots$	ϕ	
(8,b)	(6,c)	(3,d)	$\cdots$		$\{\mathbf{a}\}$	$\Delta(\phi, a) = 10 > 8$
(6,c)	(3,d)	$\cdots$			$\{a\}$	$\Delta(\{a\}, b) = 4 < 6$
(6,c)	(4,b)	(3,d)	$\cdots$		$\{a\}$	push (4,b)
(4,b)	(3,d)	$\cdots$			$\{\mathbf{a}, \mathbf{c}\}$	$\Delta(\{a\}, c) = 5 > 4$

In Example 2, only three nodes – $a, b,$ and c – are used to get the top-2 seed nodes. This phenomenon does not change even when a graph has millions of nodes. Accordingly, maintaining entries for a subset of nodes in the priority queue does not have a negative impact on the influence spread of the CELF greedy solution.

By maintaining entries for only a subset of nodes in the priority queue, the IPA-integrated CELF greedy algorithm can discard the majority of influence paths. When an entry for a node v is in the priority queue, influence paths from v, $P_{v \to V}$, should be stored in memory. For $v \neq u$, $P_{v \to V}$ and $P_{u \to V}$ do not have common influence paths. Thus, when an entry for a node v is discarded from the priority queue, its corresponding influence paths $P_{v \to V}$ can also be freed from memory. Note that PMIA, which uses also another kind of local structure in influence spread approximation, cannot discard local structures of insignificant nodes because in PMIA local structures of different nodes have inter-dependency for $\Delta(S, v)$ evaluation.

One open problem for partially maintained priority queue in the CELF greedy algorithm is how many entries should be maintained in order not to lose influence spread. In the experiment in Section VI-B4, to get the top-k seed nodes, it is empirically shown that $3k$ is a sufficient number of priority queue entries for various real datasets.

VI. Experiment

In this section, to compare IPA with the other existing algorithms, we look at extensive experiments that were conducted on five real-world datasets.

A. Experiment setup

Datasets: Five real-world datasets were used in the experiments. Epinion [18] is a customer trust network in which each customer is a node and a customer's trust in another customer is an edge. Stanford [19] is a traditional Web graph in which documents are nodes and hyperlinks are edges. DBLP [6] is an academic collaboration network of co-authorships. Patent [20] is a citation network of US patents. LiveJournal [21] is a blogging network in which bloggers are nodes and friendships between them are edges. Table I gives the detailed statistics for the five datasets.

TABLE I
Detailed statistics for the datasets

Dataset	Epinion	Stanford	DBLP	Patent	LiveJournal
# of Nodes	75.8K	281K	655K	3.77M	4.85M
# of Edges	509K	2.31M	3.98M	16.5M	69.0M
Average Degree	6.71	8.20	6.10	4.38	14.2
Max In Degree	3032	38606	588	779	13906
Max Out Degree	1798	255	588	770	20293
Direction	directed	directed	undirected	directed	directed

Assigning propagation probability: When $idegree(v)$ is the number of incoming edges of v, a propagation probability of $1/idegree(v)$ is assigned to each incoming edge of v.

Algorithms: Our method IPA was compared with other existing solutions.

- Greedy : the CELF greedy algorithm [9]. To measure stable influence spread, $20,000$ Monte-Carlo simulations were repeated.

- PMIA : the state of the art algorithm for scalable influence evaluation in the IC model [6], which approximates influence spread using local structure based on the most probable influence path.
- IPA : the influence spread approximation algorithm described in Sections III and IV.
- SD : the single discount heuristic in which, when a node is selected as a seed node, the degree of its neighbors decreases by 1. It only uses structural property (e.g., the degree of a node) to search seed nodes.
- Random : randomly selecting k nodes as a baseline method.

Because IPA and PMIA are influence spread approximation algorithms, at the macro level of the influence maximization problem they are integrated with the CELF greedy algorithm [9].

Threshold: IPA and PMIA require a threshold θ, which controls the size of their own influence evaluation units. For each dataset, IPA and PMIA select the best threshold among $1/5, 1/10, 1/20, \cdots, 1/2560$. Section VI-B1 describes how a threshold is selected.

Influence spread of seed nodes: After finding the top-k seed set for each algorithm, Monte-Carlo simulations of influence propagation were repeated $20,000$ times. Then, the averaged influenced nodes were treated as influence spread.

Running environment: All algorithms were implemented in C++ . All experiments were conducted on a Linux machine with two Intel Xeon CPUs (a total of 8 physical cores) and $24GB$ of memory.

B. Experiment results

After searching for the best thresholds, we measured (1) the processing time for each algorithm to evaluate scalability, (2) the processing time trends as the number of seed nodes increased to evaluate progressiveness, (3) the influence spread of the IPA solution according to changes in priority queue size to evaluate memory efficiency, (4) the influence spread of each algorithm's solution to evaluate effectiveness, and (5) the speed up of parallel IPA to assess parallelization effect.

1) Selecting Threshold: Before reporting on the experimental results, such as processing time and influence spread of seed nodes, we will first describe how the thresholds of IPA and PMIA were set. IPA ignores influence paths p whose $ipp(p)$ is less than a threshold θ. θ controls the trade-off between effectiveness and efficiency. Intuitively, a low θ value provides a more accurate influence approximation, but requires more processing time. To set a proper threshold for IPA, for each threshold which ranges over $1/5, 1/10, 1/20, 1/40, \cdots, 1/2560$, we recorded the processing time and the influence spread of the resultant seed nodes. We then chose the lowest threshold that reaches stable influence spread. The threshold of PMIA has the same role as that of IPA. Therefore, we set the threshold for PMIA the same as we did for IPA.

Figure 4 illustrates an influence spread trend with increasing processing time by decreasing the threshold for the DBLP dataset. From the figure, we chose the elbow point $\theta = 1/320$ which is the first point having stable influence spread.

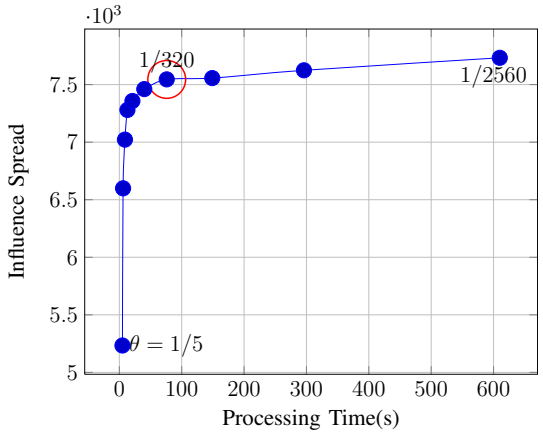


Fig. 4. Influence increase with respect to processing time induced by different thresholds for DBLP

TABLE II
SELECTED THRESHOLDS FOR IPA AND PMIA

	Epinion	Stanford	DBLP	Patent	LiveJournal
IPA	1/320	1/160	1/320	1/40	1/80
PMIA	1/160	1/160	1/640	1/80	n/a

Table II shows the selected thresholds for IPA and PMIA for five datasets. The selected thresholds were used throughout the ensuing experiments. We could not find the PMIA threshold in the LiveJournal dataset, as PMIA failed to finish for the threshold after $1/20$ due to excessive memory usage over $24GB$ before reaching a stable influence spread.

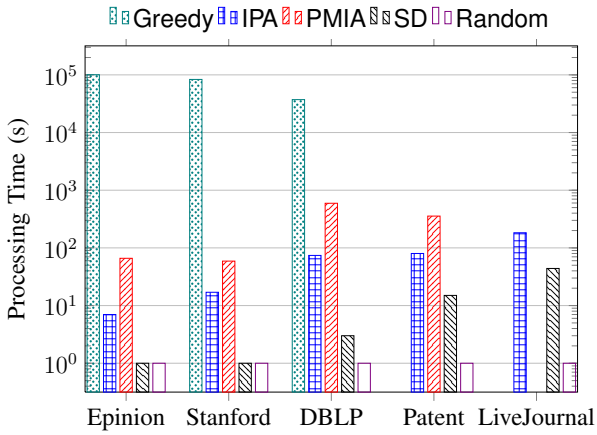
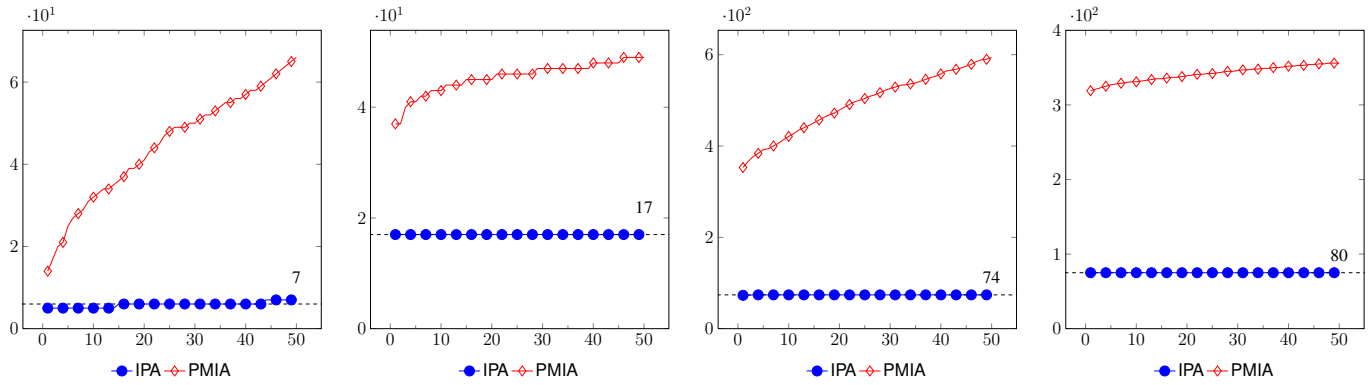


Fig. 5. Processing Time (X axis : Datasets, Y axis : Processing Time)

2) Processing time: Let us now look at the processing time for getting the top-50 seed nodes for all combinations of datasets and algorithms. The results are illustrated in Figure 5, where y-axis is log-scaled. For all datasets, it can be seen thatIPA always took less time than PMIA – the state of the art algorithm. Specifically, IPA was $9.4, 3.5, 8.0$ and 4.45 times



(a) Epinion (b) Stanford (c) DBLP (d) Patent

Fig. 6. Processing time as the number of seed nodes increases (X axis : number of seed nodes, Y axis : processing time in seconds)

faster than PMIA in Epinion, Stanford, DBLP, and Patent datasets, respectively. In LiveJournal, PMIA did not finish due to excessive memory usage over $24GB$. The processing time for IPA monotonically increased as the number of nodes increased while the processing time for PMIA fluctuated; PMIA sometimes took more time in a smaller dataset (e.g., Epinion over Stanford, and Patent over DBLP) even when the same threshold is used.

Faster processing of IPA was achieved by considering a single influence path as an influence evaluation unit, while PMIA sets local influence arborescence as an influence evaluation unit. When getting influence evaluation units, both algorithms did not have much difference because influence paths and local arborescences are obtained by simple graph traversal such as breath first search. However, in the influence path blocking detection, IPA only needs set membership operations between influence paths, whereas PMIA requires cascading reconstruction of local arborescences.

Note that in all datasets Greedy took too much time to get seed nodes. We ran all algorithms up-to $100,000s$ and Greedy did not provide seed nodes in Patent and LiveJournal. $20,000$ simulations to get stable influence spread is a big obstacle to making Greedy scalable. SD takes less than $100s$, even in LiveJournal, because it only exploits structural properties and does not consider influence diffusion. However, SD solutions showed poor influence spread. This is detailed in Section VI-B5.

3) Progressiveness in providing seed nodes: We also compared the progressiveness of IPA and PMIA in providing seed nodes. When finding seed nodes, IPA and PMIA , which were integrated with the greedy algorithm, provided seed nodes one by one. Thus, before obtaining the kth seed node, IPA and PMIA could produce earlier seed nodes, which is a desirable feature.

To see the progressiveness of IPA and PMIA, we recorded the trends in processing time to get each ith seed node ($1 \leq i \leq 50$) taken by both algorithms. Figures 6a through 6d show such trends in the first four datasets. In all the datasets, IPA did not need additional processing time after finding the first seed node, whereas the processing time for PMIA increased linearly. In LiveJournal, IPA took $183s$ to get the first seed node and only took additional one second to get the remaining 49 seed nodes. We cannot report on the LiveJournal dataset because PMIA failed to provide a result due to the memory problem. Moreover, IPA found the first seed node faster than PMIA did. Accordingly, the gap in processing time between IPA and PMIA increased as the number of seed nodes increased.

4) Priority Queue Size vs. Influence Spread: To find the appropriate number for priority queue size, influence spread of IPA solution was analyzed according to the change in priority queue size from k to $4k$. Table III shows the influence spread trends. In all datasets, after $2k$, the influence spread showed a difference less than 0.1% and such a difference is ignorable because influence spread obtained by Monte-Carlo simulation has some variation inherently. Empirically, from the table, $3k$ is a conservatively safe number for priority queue entries.

TABLE III
PRIORITY QUEUE SIZE VS. INFLUENCE SPREAD ($k = 50$)

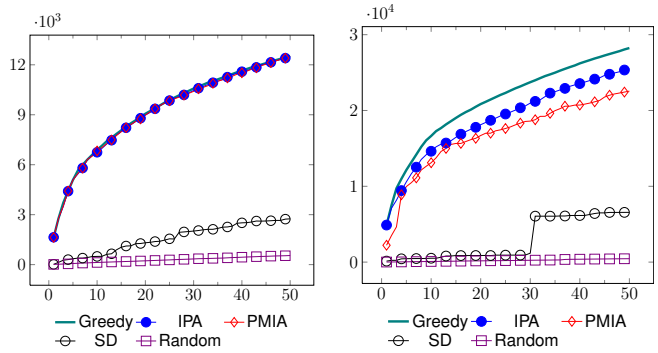
priority queue size	Epinion	Stanford	DBLP	Patent	LiveJournal
k	12496	21172	7533	10551	106745
$2k$	12486	25505	7549	10899	107469
$3k$	12476	25509	7550	10902	107482
$4k$	12482	25509	7553	10900	107451

TABLE IV
MEMORY USAGE OF IPA ($k = 50$)

priority queue size	Epinion	Stanford	DBLP	Patent	LiveJournal
IPA($\|V\|$)	$1.6GB$	$9.2GB$	$17GB$	$\times(> 24GB)$	$\times(> 24GB)$
IPA($3k$)	**207MB**	**597MB**	**419MB**	**1.6GB**	**3.6GB**
PMIA($\|V\|$)	$418MB$	$1.6GB$	$5.5GB$	$16GB$	$\times(> 24GB)$

Table IV shows the memory usage of IPA and PMIA for different sizes of the priority queue. When all nodes were inserted in priority queue, IPA used more memory than PMIA because IPA stores all influence paths between two nodes but PMIA stores only one path between them. IPA failed

to provide a solution for Patent and LiveJournal and PMIA failed in LiveJournal. However, by storing a subset of nodes in the priority queue (e.g., $3k$), IPA dramatically saved memory without losing the influence of seed solution. Especially in LiveJournal, IPA provided seed nodes using less then $4GB$ memory.



(a) Epinion (b) Stanford

Fig. 7. Influence Spread in Epinion and Stanford (X axis : number of seed nodes, Y axis : influence spread)

5) Influence spread: To see the effectiveness of algorithms, we compared influence spread of each algorithm's top-k solution. The target performance of IPA and PMIA solutions is the influence spread of the Greedy solution because IPA and PMIA approximate influence spread, which replaces the $20,000$ Monte-Carlo simulations of Greedy to get stable influence spread. The influence spread of the SD solution represents how much an influence maximization algorithm, which does not consider influence flow, catches up with other algorithms that exploit influence flow. The influence spread of Random solution is a baseline method where any sophisticated algorithm's solution should provide better influence spread. Figures 7 through 9 illustrate the trends of influence spread as each seed node is added to seed set up-to 50. Influence trends for experiment scenarios that failed to finish due to exhaustive time consumption or memory usage, are not reported.

Figure 7a shows the influence spread trends in the Epinion trust-network dataset. Up to the 50th seed node, IPA, PMIA, and Greedy solutions show similar influence spread and their trend lines are not distinguishable in Figure 7a. In fact, the influence spread loss of the IPA and PMIA solutions is less than the 1% of influence spread of the Greedy solution. This means that IPA and PMIA find seed nodes by not sacrificing effectiveness for scalability. The SD solution shows much slower increase in influence spread than algorithms that exploit influence flow.

Figure 7b shows the influence spread trends in the Stanford web-graph dataset. The seed nodes of Greedy always shows the best influence spread at any number of seed nodes. IPA provides a better solution than PMIA from the first to the last seed node. At the 50th seed node, the IPA solution shows 9.9% less influence spread while the PMIA solution shows over 20% less influence spread than the Greedy solution. One interesting result is the influence spread of the SD solution. At the 31th seed node, the influence spread of the SD solution suddenly jumps from 1178.33 to 6061.14. This means that without considering influence flow better seed nodes cannot be obtained in the early stage.

Figure 8 shows the influence spread trends in the DBLP co-authorship network dataset. In this dataset, the SD solution shows the closest influence spread to the Greedy solution. This phenomenon is observed in undirected graph datasets. In [7], the influence spread of the SD solutions for NetHEPT and NetPHY datasets is also close to that of Greedy solutions. An important common point of those three datasets is that they are undirected graphs. One reason for the good influence spread of the SD solution is the fact that in-degree and out-degree of nodes are the same in an undirected graph. IPA and PMIA solutions exhibit similar trends and their influence spread is 8.8% and 6.1% less than that of the Greedy solution. Even though IPA provides a less effective solution, it is still useful in that the influence spread of the IPA solution does not fluctuate by datasets.

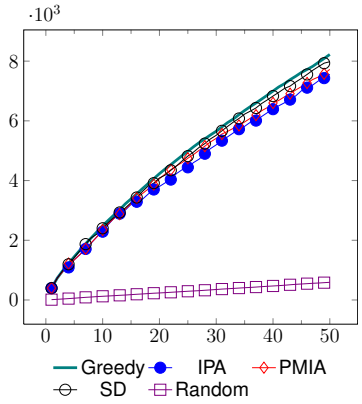
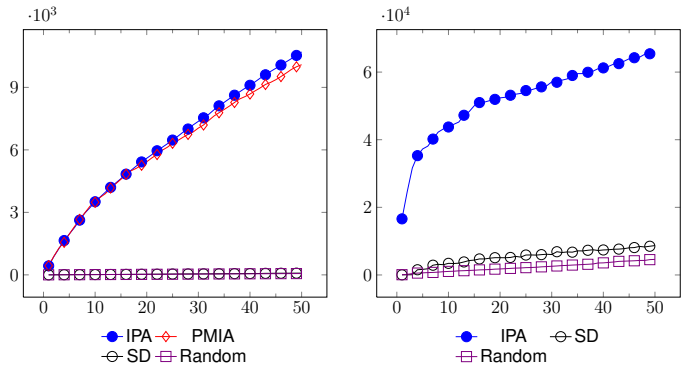


Fig. 8. Influence Spread in DBLP (X axis : number of seed nodes, Y axis : influence spread)

Figure 9a shows influence trends in the Patent citation-network dataset. In a comparison between the IPA and the PMIA solutions, there is no winner until the first 8 seed nodes. After the 8th seed node, the IPA solution constantly shows larger influence spread than the PMIA solution. Moreover, the difference in the influence spread gets larger as the number of seed nodes increases, and the IPA solution shows a 6.7% larger influence spread than the PMIA solution at the 50th seed node. Although we conjecture that the influence spread of the Greedy solution would be larger than that of the IPA and the PMIA solutions like other datasets, Greedy solution was excluded because of exhaustive processing time over $100,000s$. The influence spread of the SD solution does not show much difference from that of the Random solution, which shows the unstable effectiveness of algorithms that do not consider influence flow.

Figure 9b shows the influence spread trends for the Live-Journal friendship social-network dataset. Due to its huge size,



(a) patent (b) LiveJournal

Fig. 9. Influence Spread in Patent and LiveJournal (X axis : number of seed nodes, Y axis : influence spread)

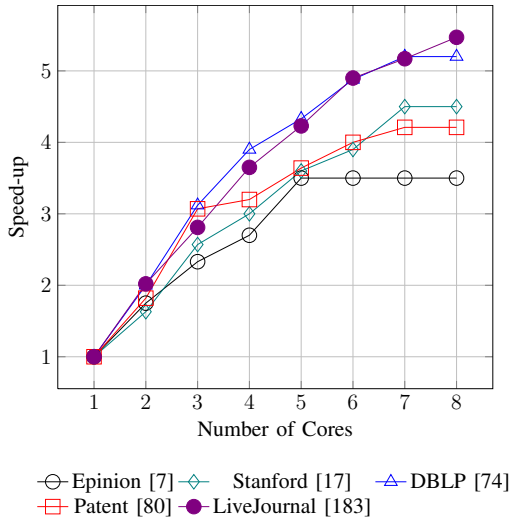


Fig. 10. Speed-up of parallel IPA (X axis : number of cores, Y axis : speed up, legend : dataset [single core processing time in seconds])

Greedy and PMIA failed to provide seed nodes solution. Greedy took over $100,000s$ and PMIA required more than $24GB$ of memory. IPA also suffered from memory problems when maintaining all influence paths. However, the memory reduction scheme of Section V enabled IPA to succeed in providing a solution.

In summary, the IPA solution showed stable influence spread in tractable time while not sacrificing much influence spread. Among Greedy, PMIA, and IPA all of which consider influence flow, only IPA succeeded in providing seed nodes for all datasets. Also, the influence spread loss of the IPA solution compared with the Greedy solution is less than 10%, while that of the PMIA solution is over 20% in Stanford.

6) Parallelization Effect: Finally, let us look at the parallelization effect of IPA. IPA is easily parallelized using OpenMP [15] meta-programming expressions, as described in Section IV. Experiments were conducted on five datasets by changing the number of cores from one to eight. The parallelization effect measure is the relative speed-up of the following equation

$$\text{speed-up} = \frac{\text{processing time of serial algorithm}}{\text{processing time of parallel algorithm}}. \quad (10)$$

In the ideal case, for c cores, c speed-up is expected. However, in general, real speed-up of shared memory parallelization is below c because (1) certain parts of the parallel algorithm, such as the synchronization section, should be run serially, (2) the overhead of spawning multiple threads cannot be ignored, and (3) memory allocation and free-up are conducted synchronously.

Figure 10 shows the speed-up trends for parallel IPA. The processing time for serial IPA is reported in the bracket beside each dataset legend. In all the datasets, as the number of cores increased, the speed-up also increased. However, the marginal speed-up increase diminished. The sub-modular property of speed-up – diminishing increase – is naturally supported by Amdahl's law [22]. However, the elbow points of the speed-up curves, where speed-up does not increase, are different for datasets. In Epinion, which is relatively small dataset, the elbow points was reached at 5 cores, and speed-up never exceeded 4. This is because the processing time for serial IPA is short enough and parallelization overhead gives negative effect on processing time. In the Stanford and DBLP dataset, the elbow point was reached at 7 cores. In Patent, although the dataset is large, the speed-up was lower than that of Stanford and DBLP, because the threshold in Patent is low $(1/40)$ and the fraction of traversing phase is consequently small. In LiveJournal, a millions of nodes dataset, room for speed-up increase still exists and more speed-up is expected after 8 cores. In summary, parallel IPA speeds up further as the number of CPU cores increases, and more speed-up is achieved for larger datasets.

VII. Conclusion

To make influence spread evaluation more scalable, we proposed an influence spread approximation algorithm IPA for the IC model. The main idea underlying IPA is that each influence path between two nodes is considered a fine-grained influence spread evaluation unit. Using the concept of influence paths, IPA drastically simplifies influence spread approximation and also makes algorithm parallelization straightforward. Accordingly, a seed node set of influence maximization problems was acquired by combining IPA and the CELF greedy algorithm. In addition, the memory usage reduction scheme of the IPA-integrated CELF algorithm reduces memory requirement significantly. Extensive experiments conducted by us on various large-scale graphs indicate that IPA is the fastest influence spread approximation algorithm, which is an order of magnitude faster than the state of the art PMIA algorithm, and also show that IPA solutions sufficiently preserves influence spread compared to time-consuming Monte-Carlo simulation-based influence spread evaluation. In addition, parallel IPA shows more scale-up as the number of CPU cores increases and is also expected scale-up more for larger datasets.

REFERENCES

[1] D. Kempe, J. M. Kleinberg, and É. Tardos, "Influential nodes in a diffusion model for social networks," in *ICALP*, ser. Lecture Notes in Computer Science, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds., vol. 3580. Springer, 2005, pp. 1127–1138.

[2] H. Ma, H. Yang, M. R. Lyu, and I. King, "Mining social networks using heat diffusion processes for marketing candidates selection," in *Proceeding of the 17th ACM conference on Information and knowledge management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 233–242.

[3] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 57–66.

[4] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 61–70.

[5] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2003, pp. 137–146.

[6] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *KDD '10: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2010, pp. 1029–1038.

[7] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2009, pp. 199–208.

[8] Y. Wang, G. Cong, G. Song, and K. Xie, "Community-based greedy algorithm for mining top-k influential nodes in mobile social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '10. New York, NY, USA: ACM, 2010, pp. 1039–1048.

[9] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 420–429.

[10] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks," in *Knowledge Discovery in Databases: PKDD 2006*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer Berlin / Heidelberg, 2006, vol. 4213, pp. 259–271.

[11] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 73–84, Sep. 2011.

[12] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *Proceedings of the third ACM international conference on Web search and data mining*, ser. WSDM '10. New York, NY, USA: ACM, 2010, pp. 241–250.

[13] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part III*, ser. KES '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 67–75.

[14] H. Sutter and J. Larus, "Software and the concurrency revolution," *Queue*, vol. 3, pp. 54–62, September 2005.

[15] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, pp. 46–55, January 1998.

[16] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ser. ICDM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 88–97.

[17] S. Datta, A. Majumder, and N. Shrivastava, "Viral marketing for multiple products," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, dec. 2010, pp. 118 –127.

[18] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *The Semantic Web - ISWC 2003*, ser. Lecture Notes in Computer Science, D. Fensel, K. Sycara, and J. Mylopoulos, Eds. Springer Berlin / Heidelberg, 2003, vol. 2870, pp. 351–368.

[19] J. Leskovec, K. J. Land, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[20] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 177–187.

[21] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 44–54.

[22] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485.