MONSTOR: An Inductive Approach for Estimating and Maximizing Influence over Unseen Networks

Jihoon Ko
KAIST AI
jihoonko@kaist.ac.kr

Kyuhan Lee
KAIST AI
kyuhan.lee@kaist.ac.kr

Kijung Shin
KAIST AI & EE
kijungs@kaist.ac.kr

Noseong Park
Department of AI, Yonsei University
noseong@yonsei.ac.kr

Abstract—Influence maximization (IM) is one of the most important problems in social network analysis. Its objective is to find a given number of seed nodes that maximize the spread of information through a social network. Since it is an NP-hard problem, many approximate/heuristic methods have been developed, and a number of them repeat Monte Carlo (MC) simulations over and over to reliably estimate the influence (i.e., the number of infected nodes) of a seed set. In this work, we present an inductive machine learning method, called <u>Mon</u>te <u>C</u>arlo <u>S</u>imula<u>tor</u> (MONSTOR), for estimating the influence of given seed nodes in social networks unseen during training. To the best of our knowledge, MONSTOR is the first inductive method for this purpose. MONSTOR can greatly accelerate existing IM algorithms by replacing repeated MC simulations. In our experiments, MONSTOR provided highly accurate estimates, achieving 0.998 or higher Pearson and Spearman correlation coefficients in unseen real-world social networks. Moreover, IM algorithms equipped with MONSTOR are more accurate than state-of-the-art competitors in 63% of IM use cases.

Index Terms—social network analysis, influence maximization, graph neural networks

I. INTRODUCTION

Influence maximization (IM) [1] is to find a certain number of seed nodes who maximize the spread of information through a social network. It is NP-hard, and many IM algorithms (e.g., Greedy [1], CELF [2], and UBLF [3]) rely on repeated Monte Carlo (MC) simulations of information cascade processes. An MC simulation takes $\mathcal{O}(|\mathcal{E}|)$ time, where $|\mathcal{E}|$ is the number of edges, and estimating the influence of a seed set via d simulations takes $\mathcal{O}(d|\mathcal{E}|)$ time, which is one of the main performance bottlenecks. In [1], [3], d is set to $10,000$.

In this work, we propose a neural network-based method, called <u>Mon</u>te <u>C</u>arlo <u>S</u>imula<u>tor</u> (MONSTOR), for estimating MC simulation results under the independent cascade (IC) model. MONSTOR is inductive. That is, it is capable of estimating MC simulation results in social networks unseen during training. To the best of our knowledge, no existing IM method is inductive. That is, existing methods cannot be used for estimating influences in unseen social networks [4], [5]. After being trained, MONSTOR significantly speeds up existing IM methods by replacing repeated MC simulations.

We conduct experiments with three real-world social networks. One strong point in our experiments is that we use real activation probabilities of edges, which are calculated from retweet logs. That is, we weight each directed edge (u, v)
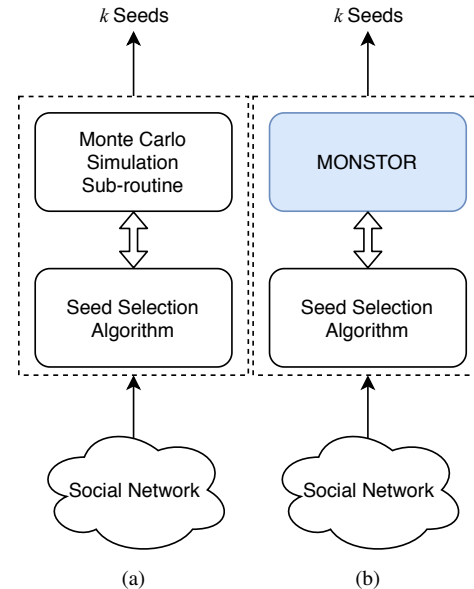
Fig. 1. Comparison of Influence Maximization Approaches: (a) MC simulated-based approaches, and (b) our approach equipped with the proposed <u>Mon</u>te <u>C</u>arlo <u>S</u>imula<u>tor</u> (MONSTOR).

with the real probability that user u influences v. Note that most previous studies on influence maximization simply used random, uniform, and degree-based probabilities [2], [3], [6]–[8], which are different from real ones.

In our experiments, MONSTOR yielded near-perfect estimations in terms of the Pearson's correlation coefficients and the Spearman's Rank correlation coefficients. In addition, simulation-based IM algorithms [1]–[3] equipped with MONSTOR yielded almost the same influence maximization results as those of the original algorithms based on MC simulations. Moreover, they were more accurate than state-of-the-art non-simulation-based IM algorithms [6]–[8] in 17 out of the 27 cases in our experiments.

The rest of this paper is organized as follows. In Section II, we introduce some concepts and notions related to our problem. In Section III, we describe the overall workflow and the detailed design of MONSTOR. In Section IV, we present experimental results. After reviewing previous related work in Section V, we briefly discuss the application of our approach to the linear threshold model in Section VI. We offer conclusions in Section VII.

II. CONCEPTS AND PROBLEM DEFINITION

In this work, we focus on the *independent cascade* (IC) model [1], where an infected node u attempts **once** to activate (i.e., directly infect) each neighbor v and the probability of success is $p_{(u,v)}$. This process is repeated for each newly infected node until there are no newly infected nodes.

Definition 1 (Activation Probability): The *activation probability* $p_{(u,v)}$ from u to v is the success probability that the node u activates its neighbor v when u is infected.

The adjacency matrix when weighting each directional edge (u, v) by $p_{(u,v)}$ is called the *activation probability matrix* $\mathbf{P} \in [0, 1]^{|\mathcal{V}| \times |\mathcal{V}|}$. That is, each (u, v)-th entry of $\mathbf{P}$ is $p_{(u,v)}$.

We use interaction logs such as retweets among users to construct a social network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and we measure activation probabilities as follows [1], [9]:

1) Bernoulli Trial (BT):
$$p_{(u,v)} = \frac{|actions(u, *) \cap actions(v, *)|}{|actions(u, *)|},$$

2) Jaccard Index (JI):
$$p_{(u,v)} = \frac{|actions(u, *) \cap actions(*, v)|}{|actions(u, *) \cup actions(*, v)|},$$

3) Linear Probability (LP):
$$p_{(u,v)} = \frac{|actions(u, *) \cap actions(*, v)|}{|actions(*, v)|},$$

where $actions(x, *)$ denotes the set of actions (e.g., retweets and replies) done by node x, and $actions(*, x)$ denotes the set of actions whose object (e.g., author of retweeted tweets and recipient of replies) is x. We consider all these probabilities, and thus we define three different activation probability matrices, $\mathbf{P}_{BT}$, $\mathbf{P}_{JI}$, and $\mathbf{P}_{LP}$, from a social network.

Based on $\mathbf{P}$, we define the infection probability of each node given a set of *seed nodes* (i.e., initially infected nodes).

Definition 2 (Infection Probability): Given a seed set $S \subseteq \mathcal{V}$, for each node $x \in \mathcal{V}$, the *infection probability* $\rho(x)$ is the probability that x is infected under the IC model with S.

In this work, we consider the following two problems:

- **Influence Estimation (IE):**
  - **Given** a seed set S,
  - **Estimate** its *influence* $\sum_{x \in \mathcal{V}} \rho(x)$ (i.e., the expected number of infected nodes).

- **Influence Maximization (IM)** [1]:
  - **Given** the number of seed nodes k,
  - **Find** the set S of k seed nodes
  - to **Maximize** the influence $\sum_{x \in \mathcal{V}} \rho(x)$.

III. PROPOSED METHOD

In this section, we propose a general inductive model MONSTOR for estimating the infection probability $\rho(x)$ of every node $x \in \mathcal{V}$ in a network that is not necessarily a part of training data (i.e., in a network unseen during training). Note that we can answer the IE problem by summing the estimates and answer the IM problem by replacing MC simulations in simulation-based algorithms (e.g., [1]–[3]) with MONSTOR.
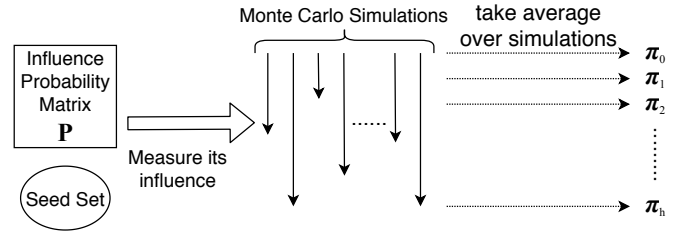

Fig. 2. How to build training data. Given a network $\mathcal{G}$ (whose activation probability matrix is $\mathbf{P}$) and a seed set, we perform multiple simulations and collect $\boldsymbol{\pi}_0, \cdots, \boldsymbol{\pi}_h$, where $\boldsymbol{\pi}_h = \boldsymbol{\pi}_{h+1}$, for the seed set. Note that the inner product $\langle \mathbf{1}, \boldsymbol{\pi}_h \rangle$ is the influence, i.e., the number of infected nodes under the IC model. We repeat these steps with many different seed sets.

Note that seed nodes are infected in the 0-th step of the IC model, and those infected directly by seed nodes are infected in the 1-st step. That is, a step is one-hop cascade from newly infected nodes. From Definitions 3-4, Proposition 1 follows.

Definition 3 (Infection Probability within Limited Steps): Given a seed set $S \subseteq \mathcal{V}$, $\rho_i(x)$ denotes the infection probability during the first i steps of the IC model. Note that $\rho_i(x) \approx \rho(x)$ if i is sufficiently large, and $\rho_i(x) = \rho(x)$ if i is greater than or equal to the longest path length in the input network.

Definition 4 (Infection Probability Vector): Let $\boldsymbol{\pi} := [\rho(x)] \in [0, 1]^{|\mathcal{V}|}$ be the vector of $\rho(x)$, $\forall x \in \mathcal{V}$. We also let $\boldsymbol{\pi}_i := [\rho_i(x)]$.

Proposition 1: The infection probability monotonically increases w.r.t. i. That is, $\boldsymbol{\pi}_i \leq \boldsymbol{\pi}_{i+1}$, or equivalently, $\rho_i(x) \leq \rho_{i+1}(x)$ for all $i \geq 0$ and $x \in \mathcal{V}$.

A. Overall Workflow

The overall workflow in MONSTOR is as follows:

1) We collect one or more social networks $\{\mathcal{G}_1, \mathcal{G}_2, \cdots \}$.
2) From each $\mathcal{G}_j$, we collect the tuples $\{(\boldsymbol{\pi}_i, \boldsymbol{\pi}_{i-1}, \cdots, \boldsymbol{\pi}_{i-e}, \mathbf{P}_j) : i \geq e\}$, where $e > 1$ is a hyperparameter, after choosing a seed set S randomly so that $1 \leq |S| \leq \frac{|\mathcal{V}|}{50}$. $\mathbf{P}_j$ can be in BT, JI, or LP. We repeat this multiple times with different seed sets, as shown in Fig. 2.
3) We train our graph convolutional network (GCN)-based model M with the training data. It has l graph convolutional layers, and it estimates $\boldsymbol{\pi}_i$ given $\boldsymbol{\pi}_{i-1}, \cdots, \boldsymbol{\pi}_{i-e}$, That is, M estimates a single step of the IC model.
4) We stack s times the pre-trained model M, and this stacked model estimates $\boldsymbol{\pi}_s$ from $\boldsymbol{\pi}_0$. That is, it estimates end-to-end simulations under the IC model. Hereinafter, MONSTOR means the stacked GCN, described in Fig. 3, unless otherwise stated.
5) For the IE problem, we compute $\langle 1, \boldsymbol{\pi}_s \rangle$ using the estimated $\boldsymbol{\pi}_s$. For the IM problem, we replace the MC simulation subroutine of existing IM algorithms (e.g., [1]–[3]) with MONSTOR.

Note that, in the training phase, we use MC simulations to obtain $\boldsymbol{\pi}_i, \boldsymbol{\pi}_{i-1}, \cdots, \boldsymbol{\pi}_{i-e}$. In the test phase, MC simulations in (potentially unseen) target networks are not needed.

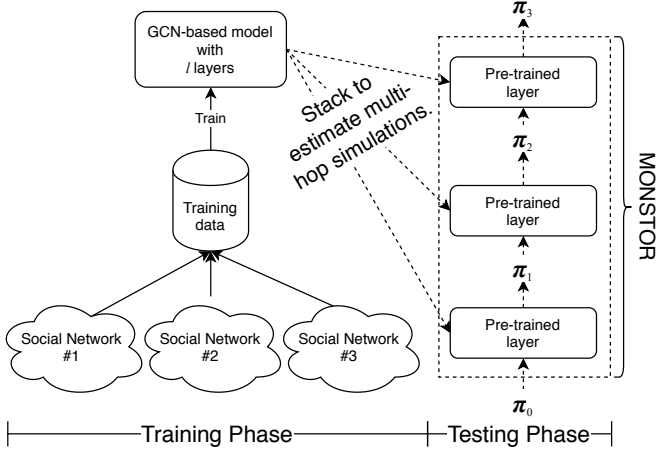This trained GCN-based model estimates one-hop simulations.

Fig. 3. The overall workflow in our approach. We train a GCN and stack it s (e.g., 3 in this figure) times for testing. MONSTOR estimates $\boldsymbol{\pi}_3$ from $\boldsymbol{\pi}_0$.

B. Detailed Design

We describe our GCN-based model M and the training method for it. As stated earlier, M estimates $\boldsymbol{\pi}_i$ given $\boldsymbol{\pi}_{i-1}$, $\cdots$, $\boldsymbol{\pi}_{i-e}$. Specifically, M initializes the feature vector of each node v as $\mathbf{h}_v^0 := (\rho_{i-e+1}(v) - \rho_{i-e}(v), \cdots, \rho_{i-1}(v) - \rho_{i-2}(v), \rho_{i-1}(v))$, and it repeatedly computes new feature vectors of each node as follows for $1 \leq i \leq l$:

$$\mathbf{a}_v^i := \mathsf{MAX}(\{p_{(u,v)} \cdot (\mathbf{h}_u^{i-1} \mathbf{W}_1^i + \mathbf{b}_1^i) : u \in \mathsf{NEI}(v)\}), \; \forall v \in V,$$

$$\mathbf{h}_v^i := \mathsf{ReLU}(\mathsf{CONCAT}(\mathbf{h}_v^{i-1}, \mathbf{a}_v^i)\mathbf{W}_2^i + \mathbf{b}_2^i), \; \forall v \in V,$$

where $\mathbf{h}_u^i \in \mathbb{R}^{d_i}$ is the feature vector of the node u at the i-th layer; $\mathsf{NEI}(v)$ is the set of neighbors of v; MAX is the element-wise max function; CONCAT is the concatenation function; and $\mathbf{W}_1^i \in \mathbb{R}^{d_{i-1} \times d_{i-1}}$, $\mathbf{W}_2^i \in \mathbb{R}^{2d_{i-1} \times d_i}$, $\mathbf{b}_1^i \in \mathbb{R}^{d_{i-1}}$ and $\mathbf{b}_2^i \in \mathbb{R}^{d_i}$ are learnable parameters.

The idea of multiplying $p_{(u,v)}$ and $\mathbf{h}_u^{i-1}$ is inspired by the fact that activation probabilities in the IC model are multiplied following a cascade route. For instance, the probability that u_1 activates u_2 and u_2 activates u_3 is $p_{(u_1,u_2)} \cdot p_{(u_2,u_3)}$.

Instead of directly estimating the raw values in $\boldsymbol{\pi}_i$, our model M uses the following more effective estimation method, which is inspired by the monotonicity (see Proposition 1):

$$M(\boldsymbol{\pi}_{i-1}, \cdots, \boldsymbol{\pi}_{i-e}, \mathbf{P}; \boldsymbol{\theta}) := \min\{\boldsymbol{\pi}_{i-1} + \mathbf{h}^l, \mathbf{u}_i\}, \quad (1)$$

where $\boldsymbol{\theta}$ is the learnable parameters of M; $\mathbf{u}_i \in \mathbb{R}^{|\mathcal{V}|}$ is a theoretical upper bound of $\boldsymbol{\pi}_i$ (see Eq. (2) below); and $\mathbf{h}^l \in \mathbb{R}^{|\mathcal{V}|}$ is the vector concatenating $\mathbf{h}_v^l \in \mathbb{R}$ (i.e., $d_l = 1$) for all $v \in V$. By Eq. (1), $\boldsymbol{\pi}_i$ always lies between $\boldsymbol{\pi}_{i-1}$ and $\mathbf{u}_i$, which are lower and upper bounds of $\boldsymbol{\pi}_i$. By adding these bounds, we relieve the difficulty of the estimation task. In our preliminary studies, directly estimating the raw values in $\boldsymbol{\pi}_i$ was not as successful as the proposed way.

Of many possible loss functions, we train our model M using the following loss function:

$$\mathcal{L} := \frac{1}{|T|} \sum_{t \in T} \left(\frac{\|M(t; \boldsymbol{\theta}) - \boldsymbol{\pi}_i\|_1}{|\mathcal{V}|} + \lambda \frac{|\mathbf{1} \cdot M(t; \boldsymbol{\theta}) - \langle \mathbf{1}, \boldsymbol{\pi}_i \rangle|}{\langle \mathbf{1}, \boldsymbol{\pi}_i \rangle} \right),$$

where T is a training set; $t = (\boldsymbol{\pi}_i, \boldsymbol{\pi}_{i-1}, \cdots, \boldsymbol{\pi}_{i-e}, \mathbf{P}) \in T$

is a training sample; and λ is a hyperparameter. We aim to fit both individual infection probabilities and overall influence.

Upper Bound of Infection Probabilities. We prove the upper bound $\mathbf{u}_i$ of $\boldsymbol{\pi}_i$, which our estimation relies on.

Theorem 1: For all $i \geq 2$, the vector $\mathbf{u}_i$, defined as,

$$\mathbf{u}_i := \boldsymbol{\pi}_{i-1} + (\boldsymbol{\pi}_{i-1} - \boldsymbol{\pi}_{i-2})\mathbf{P}, \quad (2)$$

is an upper bound of $\boldsymbol{\pi}_i$. That is, $\boldsymbol{\pi}_i \leq \mathbf{u}_i$.

Proof: For each node $v \in \mathcal{V}$, we let t_v be the step at which v gets infected under the IC model. If v is never infected, $t_v = \infty$. For each node $v \in \mathcal{V}$ and its neighbor u, let $X_{u \to v}^i$ be the event that v is infected by u at step i. Then,

$$\mathbb{P}(t_v = i) = \mathbb{P}(\bigcup_{u \in \mathsf{NEI}(v)} X_{u \to v}^i) \leq \sum_{u \in \mathsf{NEI}(v)} \mathbb{P}(X_{u \to v}^i). \quad (3)$$

In Eq. (3), $\mathbb{P}(X_{u \to v}^i) = \mathbb{P}(t_u = i-1) \cdot p_{(u,v)}$ and $\mathbb{P}(t_u = i-1) = \rho_{i-1}(u) - \rho_{i-2}(u)$. Therefore, the following inequality holds for all $v \in \mathcal{V}$:

$$\mathbb{P}(t_v \leq i) = \mathbb{P}(t_v \leq i-1) + \mathbb{P}(t_v = i)$$
$$\leq \mathbb{P}(t_v \leq i-1) + \sum_{u \in \mathsf{NEI}(v)} (\rho_{i-1}(u) - \rho_{i-2}(u)) \cdot p_{(u,v)}$$

From Definition 3, $\mathbb{P}(t_v \leq i) = \rho_i(v)$ and $\mathbb{P}(t_v \leq i-1) = \rho_{i-1}(v)$ hold, and these imply $\boldsymbol{\pi}_i \leq \mathbf{u}_i$. ∎

Complexity and Runtime Analysis. Once MONSTOR is trained (potentially using graphs smaller than a target graph $\mathcal{G}$), estimating $\boldsymbol{\pi}_i$ in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a seed set S takes $\mathcal{O}(ls|\mathcal{E}|)$ time, where s is the number of stacks, and l is the number of convolutional layers per stack. In our experiments, the runtime of an estimation by MONSTOR amounts to the runtime of performing MC simulations only 100 times. According to the standard configurations [1], [3], Greedy and UBLF perform MC simulations 10,000 times per seed set.

IV. EXPERIMENTS

We review our experiments for answering the following questions:

- **Q1. Influence Maximization**: How accurate are simulation-based IM algorithms equipped with MON-STOR, compared to state-of-the-art competitors?
- **Q2. Influence Estimation**: How accurately does MON-STOR estimate the influence of seed sets?
- **Q3. Scalability**: How rapidly does the estimation time grow as the size of the input graph increase?
- **Q4. Submodularity**: Is MONSTOR submoudular as the ground-truth influence function is?

We made the datasets, the source code, and the trained models available at https://github.com/jihoonko/asonam20-monstor.

A. Experimental Settings

Datasets. We used three real-world social networks: Extended, WannaCry, and Celebrity (see Table I). For Extended, we crawled more tweets and retweets in addition to those used in [10]. In each dataset, we used online postings (and their cascade logs) during the first 50% of time for training/validation and those during the remaining 50% for testing — two

TABLE I
STATISTICS OF EACH SOCIAL NETWORK

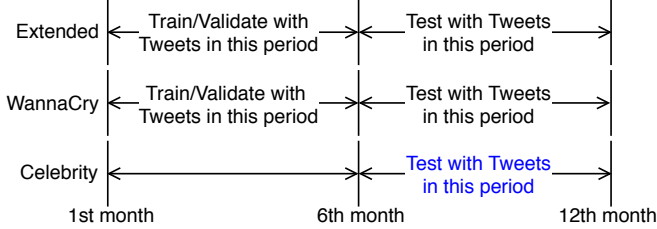|	$|\mathcal{V}|$	$|\mathcal{E}|$	$\frac{\sum p_{(u,v)}}{|\mathcal{E}|}$ in BT		$\frac{\sum p_{(u,v)}}{|\mathcal{E}|}$ in JI		$\frac{\sum p_{(u,v)}}{|\mathcal{E}|}$ in LP	
			Train	Test	Train	Test	Train	Test
Extended	11,409	58,972	0.07974	0.09194	0.03345	0.04095	0.16138	0.18371
WannaCry	35,627	169,419	0.07255	0.09466	0.02977	0.04494	0.19785	0.16297
Celebrity	15,184	56,538	0.03206	0.02787	0.00163	0.00159	0.26142	0.256

Fig. 4. Example train-test split in our experiments. We test with tweets newly posted in the test period. Note that MONSTOR can be applied to unseen social networks (i.e., the test data highlighted in blue), while existing learning approaches [4], [5] cannot be applied.

sets are disjoint. Specifically, we computed the activation probability matrices $\mathbf{P}_{BT}$, $\mathbf{P}_{JI}$, and $\mathbf{P}_{LP}$ for each of the training/validation and testing periods based on the logs. Then, we collected $1,600$ training tuples, 400 validation tuples and $2,000$ testing tuples, as described in Sec. III-A.

We trained MONSTOR with training data consisting of two out of the three networks (e.g., Extended and WannaCry) and tested it with each of the three networks (i.e., each of Extended, WannaCry, and Celebrity), as shown in Fig. 4.

Competitors. We consider the following IM algorithms as competitors: (i) Greedy [1], UBLF [3], and CELF [2] among simulation-based algorithms; (ii) SSA [6], D-SSA [6], PMIA [8], and IRIE [7] among non-simulation-based algorithms; and (iii) U-MON and C-MON, where UBLF and CELF equipped with MONSTOR which replaces MC simulations. We note that Greedy, UBLF, and CELF output the same seed users. Among them, UBLF is the fastest, and CELF and Greedy follow. U-MON and C-MON also output the same seed users. UBLF and U-MON can be used only when activation probabilities satisfy a certain property [3].

Hyperparameters. For MONSTOR, we set $e{=}4$, $l{=}3$, $\lambda{=}0.3$, $d_1{=}\cdots{=}d_{l-1}{=}16$ after some preliminary studies. At each t-th epoch, we set the learning rate to $10^{-4}{\cdot}t$ if $t \leq 10$, and $10^{-2}/t$ otherwise. We chose the best s using the validation data, and $s = 2, 3, 5$ were the best for JI, BT, and LP, respectively. For (D-)SSA, we set $\epsilon{=}0.1$ and $\delta{=}1/|\mathcal{V}|$ as in [6] and used the influence averaged over 100 independent runs. For IRIE and PMIA, we followed the settings in [7], [8].

B. Q1. Influence Maximization (IM)

We report the accuracy (i.e., influence of output seeds) of IM methods in Table II. Target influences mean influences obtained by Greedy, UBLF, and CELF with $10,000$ MC simulations, which took at least two orders of magnitude more time than the compared methods.

For BT, U-MON was most accurate in most cases. Its score was very close to the target influence. IRIE was most accurate in one case. For JI, while U-MON and IRIE were most accurate in many cases, and there was no clear winner. Each algorithm (except SSA and D-SSA) was most accurate in at least one case. One possible reason for this is that activation probabilities are relatively small compared to BT and LP, as shown in Table I. For LP, C-MON was most accurate in most cases, and its influence was very close to the target influence. Note that the activation probabilities are relatively large in LP (see Table I), and thus even small changes in the seed set can decrease influence significantly. Interestingly, IRIE was outperformed by D-SSA and SSA in many cases, while the results were the opposite when BT and JI were used.

In summary, only U-MON and C-MON were consistently accurate regardless of the activation probabilities. In most cases and even in unseen social networks, their influences were close to the target influence made by Greedy, UBLF and CELF with $10,000$ MC simulations. The other algorithms (i.e., D-SSA, SSA, IRIE, and PMIA) were not as accurate as our methods for at least one among BT, JI and LP.

C. Q2. Influence Estimation (IE)

To explain how our methods yielded high quality seed sets, as shown in Table II, we show that MONSTOR estimates the influence of candidate seed sets accurately. For each test seed set described in Section IV-A, we estimated its influence using MONSTOR and used the mean influence of $10,000$ MC simulations as the ground-truth influence. To measure the similarity between true and estimated influences over all test seed sets, we used Pearson's correlation coefficients and Spearman's Rank correlation coefficients. As seen in Table III and IV, the ground-truth influences of test seed sets and the estimated influences are highly correlated, and both coefficients were close to 1.0. That is, MONSTOR yielded near perfect estimates.

D. Q3. Scalability

We demonstrate the linear scalability of MONSTOR. To this end, we generated realistic graphs of various sizes using the R-MAT generator [11] with $a = 0.7$ and $b = c = d = 0.1$. The number of edges in the generated graphs ranged from 2^{20} to 2^{26}, and for every graph, we set the number of nodes to 20% of the number of edges. We used the weighted cascade

	Extended			WannaCry			Celebrity		
	k=10	50	100	10	50	100	10	50	100
Target Influence	481.5	968.3	1222.9	991.6	2123.5	2752.3	52.8	105.3	155.2
U-MON (E+W)	**481.1**	966.8	**1222.3**	991.0	2122.8	2745.4	**52.7**	104.1	154.3
U-MON (E+C)	480.7	967.1	1221.9	991.1	2123.1	2744.8	52.6	104.2	154.5
U-MON (W+C)	481.1	967.7	1221.7	991.0	2123.2	2745.6	51.8	104.8	155.1
D-SSA	467.5	949.0	1189.8	984.2	2071.9	2687.1	49.8	102.0	152.4
SSA	467.5	949.1	1189.8	984.3	2071.9	2687.2	49.8	102.0	152.4
IRIE	479.6	966.0	1221.7	986.6	2118.4	**2751.8**	51.7	103.0	153.0
PMIA	473.5	960.0	1199.5	989.0	2106.8	2739.9	51.7	100.0	152.1

(a) Test with BT

	Extended			WannaCry			Celebrity		
	k=10	50	100	10	50	100	10	50	100
Target Influence	244.4	529.3	706.6	533.9	1238.5	1648.0	43.7	90.3	140.2
U-MON (E+W)	244.5	529.1	706.8	**534.2**	**1239.2**	1647.4	43.7	90.4	140.4
U-MON (E+C)	244.5	**529.2**	706.6	534.2	1239.1	1647.4	43.7	**90.5**	140.4
U-MON (W+C)	244.4	529.1	706.8	534.1	1239.1	1647.3	43.7	90.4	**140.5**
D-SSA	244.5	521.2	682.8	532.3	1213.6	1589.7	43.6	89.9	139.9
SSA	244.5	521.2	682.8	532.3	1213.7	1589.8	43.6	89.9	139.9
IRIE	244.3	**529.2**	**707.3**	**534.2**	1239.0	**1647.8**	**43.8**	90.4	140.3
PMIA	**244.6**	529.1	705.4	534.1	1239.0	1646.8	42.7	90.4	140.3

(b) Test with JI

	Extended			WannaCry			Celebrity		
	k=10	50	100	10	50	100	10	50	100
Target Influence	1852.4	2876.5	3264.9	5271.6	7880.0	9098.3	5508.4	5616.7	5657.7
C-MON (E+W)	1843.0	**2863.0**	**3253.8**	5246.4	**7862.1**	**9073.5**	5508.9	5615.0	5664.9
C-MON (E+C)	1840.6	2848.5	3236.5	5253.0	7844.5	9041.7	5508.8	5616.4	5666.4
C-MON (W+C)	1839.5	2853.1	3242.0	5248.6	7850.7	9045.7	5508.8	5615.0	5665.0
D-SSA	**1844.3**	2858.7	3236.1	5256.7	7783.4	8977.3	5509.0	5606.2	5633.8
SSA	1843.8	2858.6	3236.1	**5257.2**	7783.6	8977.0	5508.8	5606.3	5633.9
IRIE	1816.2	2829.8	3201.2	5109.1	7714.1	8840.1	**5509.1**	**5617.4**	**5667.4**
PMIA	1830.0	2828.9	3243.2	5196.7	7807.6	8981.8	5508.5	5604.2	5630.2

(c) Test with LP

model [1] to determine the activation probability of each edge in the generated graphs. We measured the runtime of influence estimations by MONSTOR with $1,000$ different seed sets. For each seed set, we chose its size uniformly at random from 1 to 10% of $|\mathcal{V}|$ and then chose seed nodes uniformly at random. The runtime scaled linearly with the number of stacked GCNs (i.e., s), and we measured the runtime per stacked GCN. In Table V, we report the runtime of $1,000$ influence estimations by MONSTOR in graphs of different sizes. Consistently with the theoretical result in Section III, the runtime was near linear in the number of edges in the input graph.

E. Q4. Submodularity

It is well known that the influence maximization is a submodular maximization problem, and many IM algorithms exploit the submodularity of the ground-truth influence func-

tion. Therefore, it is crucial to show that MONSTOR has the same characteristic for claiming that IM algorithms based on the submodularity work properly when being integrated with MONSTOR.

In this section, we review our experiments for testing the empirical submodularity of MONSTOR. To this end, we used $5,000$ test seed sets that were not used for training. For each seed set, we chose its size uniformly at random from 1 to 10% of $|\mathcal{V}|$ and then chose seed nodes uniformly at random. Using each pair S and T of the seed sets, we tested whether the following submodularity condition is met:

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

In Table VI, we show the ratio of the pairs where the above submodularity condition is met. When BT or JI was used, for

TABLE III
THE ACCURACY OF MONSTOR ON THE INFLUENCE ESTIMATION PROBLEM. WE HIGHLIGHT THE ACCURACY IN UNSEEN DATASETS IN BLUE.

	Pearson Correlation			Spearman's Rank		
	BT	JI	LP	BT	JI	LP
MON. (E+W)	1.000	1.000	1.000	1.000	1.000	1.000
MON. (E+C)	1.000	1.000	1.000	1.000	1.000	0.999
MON. (W+C)	1.000	1.000	1.000	1.000	1.000	0.999

(a) Extended

	Pearson Correlation			Spearman's Rank		
	BT	JI	LP	BT	JI	LP
MON. (E+W)	1.000	1.000	1.000	1.000	1.000	1.000
MON. (E+C)	1.000	1.000	1.000	1.000	1.000	1.000
MON. (W+C)	1.000	1.000	1.000	1.000	1.000	1.000

(b) WannaCry

	Pearson Correlation			Spearman's Rank		
	BT	JI	LP	BT	JI	LP
MON. (E+W)	0.999	1.000	1.000	0.999	1.000	0.998
MON. (E+C)	1.000	1.000	1.000	1.000	1.000	0.999
MON. (W+C)	1.000	1.000	1.000	1.000	1.000	0.998

(c) Celebrity

more than 99.6% of the pairs, the submodularity condition held. When LP was used, the ratio was 99% or higher.

For each pair S and T where the submodularity condition was not met, we measured the MAPE (i.e., mean absolute percentage error) as follows:

$$\frac{f(S \cup T) + f(S \cap T) - f(S) + f(T)}{f(S \cup T) + f(S \cap T)}.$$

As shown in Table VII, the error (i.e., $f(S\cup T)+f(S\cap T)- f(S)+f(T)$) was marginal compared to the actual influence (i.e, $f(S\cup T)+f(S\cap T)$). All these experiment results support that influence estimation by MONSTOR can be considered as submodular in practice.

V. RELATED WORK

Influence maximization, i.e., to find a certain number of seed nodes who maximize the information spread through a social network, is an NP-hard problem. *Independent cascade* (IC) and *linear threshold* (LT) are the two most popular information cascade models. In the IC model, once a node u is activated (or influenced), it attempts once to activate each neighbor v with probability $p_{(u,v)}$. In the LT model, a node v is activated if a sufficient number of its neighbors (larger than a threshold) are activated. In this work, we focus on the IC model. There are several successful real-world studies based on the IC model [12], [13].

Numerous methods have been proposed for influence maximization. Since it is an NP-hard problem, all these methods approximate optimal seed nodes. They can be categorized into the following three types: i) simulation-based, ii) proxy-based, and iii) sketch-based methods. Among them, simulation-based
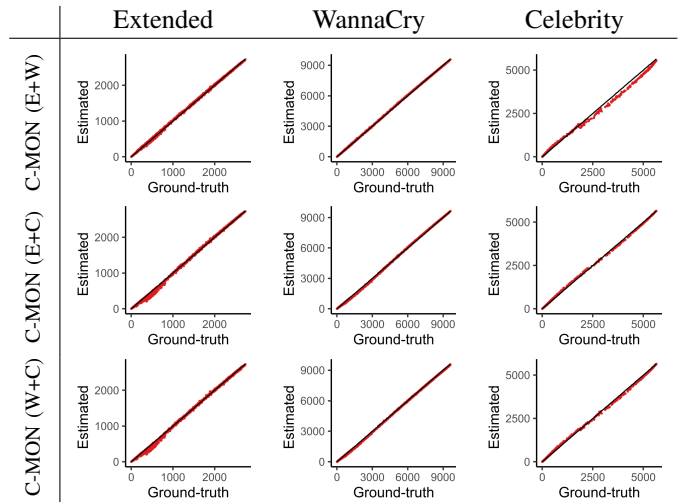
TABLE IV
SCATTER PLOTS FOR COMPARISON OF GROUND-TRUTH INFLUENCES AND THOSE ESTIMATED BY MONSTOR.
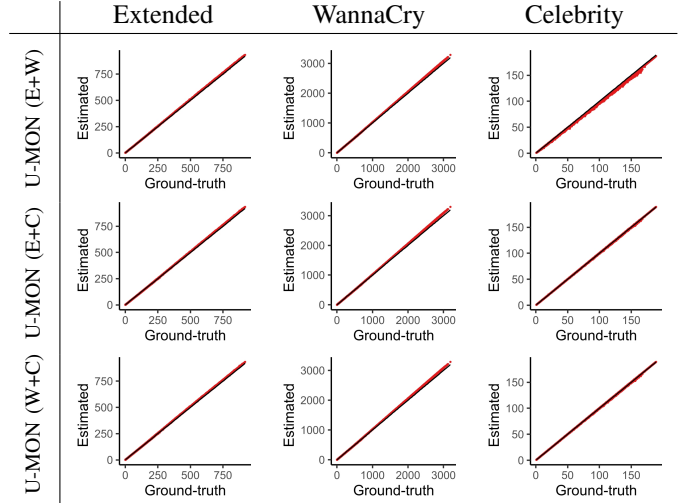
(a) Test with BT
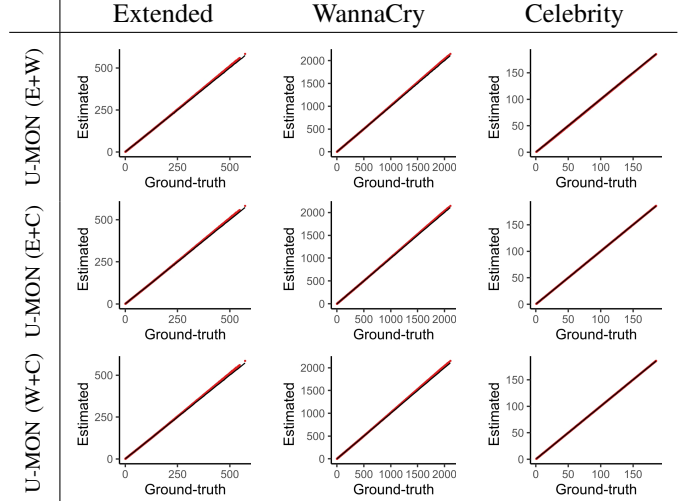
(b) Test with JI

(c) Test with LP

$|\mathcal{E}|$	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}
Estimation time (sec)	11.5	17.7	31.0	56.3	108.9	411.0	819.7

TABLE VI
THE RATIO OF THE CASES WHERE THE SUBMODULARITY HOLDS. WE
HIGHLIGHT THE CASES WITH UNSEEN DATASETS IN BLUE.

	Extended	WannaCry	Celebrity
U-MON (E+W)	0.9996	0.9999	0.9999
U-MON (E+C)	0.9994	0.9999	1.0000
U-MON (W+C)	0.9997	0.9999	1.0000

(a) Test with BT

	Extended	WannaCry	Celebrity
U-MON (E+W)	0.9988	0.9995	0.9966
U-MON (E+C)	0.9993	0.9995	0.9999
U-MON (W+C)	0.9990	0.9994	0.9998

(b) Test with JI

	Extended	WannaCry	Celebrity
C-MON (E+W)	0.9993	0.9997	0.9938
C-MON (E+C)	0.9994	0.9997	0.9970
C-MON (W+C)	0.9992	0.9996	0.9945

(c) Test with LP

TABLE VII
MAPE WHEN THE SUBMODULARITY DOES NOT HOLD. WE HIGHLIGHT
THE CASES WITH UNSEEN DATASETS IN BLUE.

	Extended	WannaCry	Celebrity
U-MON (E+W)	2.50e-04	5.40e-07	1.55e-04
U-MON (E+C)	1.48e-05	4.42e-08	1.50e-04
U-MON (W+C)	7.67e-08	7.11e-08	8.02e-08

(a) Test with BT

	Extended	WannaCry	Celebrity
U-MON (E+W)	6.91e-06	5.60e-08	4.58e-05
U-MON (E+C)	1.28e-06	4.69e-08	2.22e-05
U-MON (W+C)	3.93e-07	1.39e-07	1.10e-05

(b) Test with JI

	Extended	WannaCry	Celebrity
C-MON (E+W)	0.0022	0.0013	0.0114
C-MON (E+C)	0.0025	0.0021	0.0113
C-MON (W+C)	0.0034	0.0011	0.0095

(c) Test with LP

methods are known to be able to find better seeds than the other methods.

In the simulation-based methods, MC simulations are explicitly repeated to estimate the influence of seed sets [2], [3]. These methods focus on pruning unnecessary (redundant) simulations to minimize the required number of simulations. SSA [6] and D-SSA [6] are strong among sketch-based method, and IRIE [7] and PMIA [8] are strong among proxy-based methods. See a recent survey [14] for details.

There exist two on-going studies [4], [5] most relevant to ours. In [4], a neural network-based influence function is used to predict whether a given node will activate or not. DISCO [5] learns a function that estimates the influence of a given seed set and then selects seed nodes that maximize estimated influence based on deep reinforcement learning. However, their learning models are *transductive*, i.e., incapable of estimating influence in social networks unseen during training. They can estimate influence only in a training network potentially with different activation probabilities. We aim at designing an *inductive* method, which is capable of estimating the influence of seed nodes in networks whose connections and activation probabilities are completely unseen during training. In addition, our method estimates MC simulation results and thus can be equipped with greedy-based IM algorithms [1]–[3], while the former [5] directly searches seed nodes.

Machine learning approaches have been used for different but relevant tasks, including to predict the future size of a cascade from its initial stage [15] and to predict whether a node has been activated or not given its near neighbors' activation states and a subgraph around the node [16]. There also exist many attempts to predict solutions of various NP-hard problems, and our work was greatly inspired by them. Solutions for the satisfiability, maximal independent set, minimum vertex cover, traveling salesman, and knapsack problems can be predicted by deep learning models [17]–[21].

VI. DISCUSSION ON LINEAR THRESHOLD

We briefly discuss why we focus on the IC model but exclude the LT model in this work. As described earlier, we use an upper bound information to ease the prediction task. For the LT cascade model, however, it is not easy to find tight upper bounds [3]. If $\mathbf{u}_i$ is not tight, our design not to directly predict the raw values will be discouraged. In our experience, directly predicting raw values is inferior to our design relying on an upper bound. For this reason, we leave influence prediction under LT as an open problem.

VII. CONCLUSIONS

In this work, we present MONSTOR, an inductive learning algorithm for estimating the influence of seed nodes under the IC model. In our experiments, MONSTOR accurately

estimated the influence even in networks unseen during training. Moreover, simulation-based influence maximization algorithms equipped with MONSTOR, which replaces repeated MC simulations, performed reliably well, outperforming state-of-the-art competitors in 17 out of the 27 considered cases.

REFERENCES

[1] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003.

[2] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: Optimizing the greedy algorithm for influence maximization in social networks," in *WWW*, 2011.

[3] C. Zhou, P. Zhang, J. Guo, X. Zhu, and L. Guo, "Ublf: An upper bound based approach to discover influential nodes in social networks," in *ICDM*, 2013.

[4] B. Yan, K. Song, J. Liu, F. Meng, Y. Liu, and H. Su, "On the maximization of influence over an unknown social network," in *AAMAS*, 2019.

[19] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NeurIPS*, 2015.

[5] H. Li, M. Xu, S. S. Bhowmick, C. Sun, Z. Jiang, and J. Cui, "DISCO: influence maximization meets network embedding and deep learning," *CoRR*, vol. abs/1906.07378, 2019.

[6] 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*, 2016.

[7] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *ICDM*, 2012.

[8] C. Wang, W. Chen, and Y. Wang, "Scalable influence maximization for independent cascade model in large-scale social networks," *DMKD*, vol. 25, 11 2012.

[9] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010.

[10] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *USENIX Security*, 2015.

[11] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *SDM*, 2004.

[12] A. Yadav, H. Chan, A. Xin Jiang, H. Xu, E. Rice, and M. Tambe, "Using social networks to aid homeless shelters: Dynamic influence maximization under uncertainty," in *AAMAS*, 2016.

[13] B. Wilder and Y. Vorobeychik, "Controlling elections through social influence," in *AAMAS*, 2018.

[14] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *TKDE*, vol. 30, pp. 1852–1872, 2018.

[15] Z. Shafiq and A. Liu, "Cascade size prediction in online social networks," in *Networking*, 2017, pp. 1–9.

[16] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *KDD*, 2018.

[17] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *NeurIPS*, 2017.

[18] A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, "Learning Heuristics over Large Graphs via Deep Reinforcement Learning," p. arXiv:1903.03332, 2019.

[20] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," vol. abs/1611.09940, 2016.

[21] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.