PIANO: Influence Maximization Meets Deep Reinforcement Learning

Hui Li, Mengting Xu, Sourav S Bhowmick, Joty Shafiq Rayhan, Changsheng Sun, Jiangtao Cui

***Abstract*—Since its introduction in 2003, the influence maximization (IM) problem has drawn significant research attention in the literature. The aim of IM, which is NP-hard, is to select a set of k users known as seed users who can influence the most individuals in the social network. The state-of-the-art algorithms estimate the expected influence of nodes based on sampled diffusion paths. As the number of required samples have been recently proven to be lower bounded by a particular threshold that presets tradeoff between the accuracy and efficiency, the result quality of these traditional solutions is hard to be further improved without sacrificing efficiency. In this paper, we present an orthogonal and novel paradigm to address the IM problem by leveraging deep reinforcement learning to estimate the expected influence. Specifically, we present a novel framework called PIANO that incorporates *network embedding* and *reinforcement learning* techniques to address this problem. In order to make it practical, we further present PIANO-E and PIANO@$\langle d \rangle$, both of which can be applied directly to answer IM without training the model from scratch. Experimental study on real-world networks demonstrates that PIANO achieves the best performance w.r.t efficiency and influence spread quality compared to state-of-the-art classical solutions. We also demonstrate that the learned parametric models generalize well across different networks. Besides, we provide a pool of pretrained PIANO models such that any IM task can be addressed by directly applying a model from the pool without training over the targeted network.**

***Index Terms*—influence maximization, deep reinforcement learning, graph embedding, social network.**

I. Introduction

Online social networks have become an important platform for people to share and disseminate information. Given the widespread usage of social media, individuals' perceptions, preferences and behavior are often influenced by their peers/friends in social networks. Since 2003, the *influence maximization* (IM) problem has been extensively studied to maximize diffusion of innovations and ideas in a network. The purpose of IM is to select a set of k seed nodes who can influence the most individuals in the network [1]. For instance, an advertiser may wish to send promotional material about a product to the k seed users of a social network that are likely to sway the largest number of users to buy the product.

A large number of greedy and heuristic-based IM solutions have been proposed in the literature to improve efficiency, scalability, or influence quality. State-of-the-art IM techniques attempt to generate $(1 - 1/e - \epsilon)$-approximate solutions with a smaller number of RIS (*Random Interleaved Sampling*) samples, which are mainly used to estimate the expected maximum influence (denoted as $\sigma(v, S)$) for an arbitrary node v given the current selected seeds S. They use sophisticated estimation methods to reduce the number of RIS samples closer to a theoretical threshold θ [2]. As θ provides a lower bound for the number of required RIS samples, these methods have to undertake a *diffusion sampling phase* and generate sufficient propagation samples in order to estimate the expected influence before selecting a seed. Despite the improvements of the sampling model and reduction in the number of required samples brought by recent studies, the cost of generating these samples is large especially in huge networks. Consequently, in this paper we ask the following question: *is it possible to avoid the diffusion sampling phase in* IM *solutions by utilizing a learned parametric function to estimate* $\sigma(v, S)$? We answer to this question affirmatively by proposing a novel framework that utilizes network embedding and reinforcement learning to tackle the IM problem.

The core challenge in IM lies in the process of estimating $\sigma(v, S)$ given v and the partial solution S, which is known to be #P-hard [1]. Traditional IM solutions address it by sampling the diffusion paths to generate an unbiased estimate for $\sigma(v, S)$. In essence, $\sigma(v, S)$ can be viewed as a mapping $\sigma : V \times G \times \Psi \rightarrow \mathbb{R}$, where $G(V, E)$ denotes the network and Ψ refers to the set of diffusion models, which defines the criteria to determine whether a node is influenced or not. That is, given a network G, an arbitrary node $v \in V$, and a particular diffusion model (*e.g.,* Independent Cascade, Linear Threshold model [1]), σ outputs the expected maximum number of influenced nodes by v. In this paper, we take a radically different approach where we view IM as a problem of finding the *optimal policy* to select the k-best seeds (i.e., k-best action sequence) in a deep reinforcement learning (RL) framework called PIANO (dee**P** re**I**nforcement le**A**rning-based i**N**fluence maximizati**O**n). Crucially, PIANO approximates σ as a (value) function $\tilde{\sigma}(v, S; \Theta)$ parameterised by Θ and *learns* the values of the parameters Θ. It is worth mentioning that learning such mapping function is non-trivial and challenging. First, we need to transform the topology information of the target network into features. Second, there is no target expected maximum influence to supervise σ. Hence, supervised learning approaches cannot be adopted in this scenario. To address these challenges, PIANO seamlessly *integrates* RL [3] with network embedding [4] in an end-to-end deep RL framework.

The network embedding method considers the network topology to encode each node $v \in V$ into a feature vector, which acts as input to an RL algorithm to learn the value

H. Li, M. Xu, J. Cui are with School of Computer Science and Technology, Xidian University, Xi'an, China.

S.S. Bhowmick and J. S. Rayhan are with School of Computer Science and Engineering, Nanyang Technological University, Singapore.

C. Sun is with School of Computing, National University of Singapore, Singapore

Manuscript received Nov 03, 2021; revised Mar 01, 2022; accepted Mar 31, 2022.

function $\tilde{\sigma}(v, S; \Theta)$ from rewards. The learned value function (more specifically, state-action value function) implicitly represents the learned optimal policy that can be applied to an unseen network for selecting k-best seeds. The learning-based framework makes the approach generalizable (*i.e.,* robust to small changes in the network) and scalable to large networks (*i.e.,* no need to compute all individual states and store them in memory). Moreover, we theoretically show that under our model the k seeds can be selected at one time, instead of one-after-another, which requires reevaluating $\sigma(v, S)$ for k times. Our exhaustive empirical studies demonstrate that once the mapping function is learned, it can be applied to other homogeneous networks with the same topological characteristic (*i.e.,* average degree).

The main contributions of the paper are as follows.

- We present a novel framework called PIANO that exploits learning methods to solve the classical IM problem. Specifically, a novel learning method is proposed to approximate $\sigma(v, S)$ as a parameterized value function $\tilde{\sigma}(v, S; \Theta)$, by exploiting model-free RL and deep learning for network embedding. PIANO generates seeds with superior running time to state-of-the-art machine learning-oblivious IM techniques without compromising on result quality. Specifically, it is up to 36 times faster than SSA [5]. Furthermore, our influence quality is slightly better than the traditional methods.
- We show how PIANO can be utilized to address the IM problem in large-scale networks even in the presence of evolution.
- We provide a pool of pretrained PIANO models such that any IM task can be addressed by directly applying a model from the pool without training over the targeted network. The facilitates generalization of the framework to different datasets.

The rest of this paper is organized as follows. Section II reviews related work. We formally present the learning-based IM problem in Section III. We introduce the model training and seeds selection procedures in Sections IV and V, respectively. Experimental results are reported in Section VI. Finally, we conclude this work in Section VII.

II. Related Work

Influence Maximization. Since the elegant work in [1], the IM problem has been studied extensively. Kempe *et al.* [1] proved that the problem of IM is NP-hard and provided a greedy algorithm that produces a $(1 - 1/e - \epsilon)$-approximate solution. Since then, series of works [6], [7] have been proposed to improve the response time while preserving the approximation guarantee. Besides these approximate algorithms, many excellent heuristic algorithms [8], [9], [10], which do not provide an approximation ratio, have been proposed to reduce running time further. Although these heuristic algorithms reduce the execution time by orders of magnitude, they sacrifice accuracy of the results significantly [11].

Since the introduction of *reverse influence sampling* (RIS) [12], which reversely samples a group of influence paths, a series of advanced approximate algorithms have been proposed, which can not only provide approximation guarantees, but also exhibit competitive running time compared to heuristic solutions. TIM/TIM+ [7] significantly improved the efficiency of [12] and is the first RIS-based approximate algorithm to achieve competitive efficiency with heuristic algorithms. Afterwards, IMM [2] and SSA/D-SSA [5] were presented, both of which have been recognized as the state-of-the-art solution that achieve the best efficiency [13]. Notably, none of these efforts integrate machine learning with the IM problem to further enhance the performance.

In recent years, there have been increasing efforts to address the IM problem utilizing learning methods. [14], [15], [16], [17] have employed RL to find best strategy in competitive IM problem [18], [19]. Both of [14], [15] treat the competition between multiple parties as an adversarial game; and employ reinforcement learning to find the best policy (*i.e.,* strategy) that can maximize the profit given the opponents' choices. [20] uses deep learning to learn topic-aware influence in order to solve the Topic-Aware IM problem. In contrast, we propose a learning model for general IM problem and pre-trained model pool for practical applications without training from scratch. The authors in [16] employ deep learning to infer the entity correlations and influence cascading behavior. They do not natively model the influence estimation as a learning task and hence it is orthogonal to our problem. Besides, [21], [22] adopt learning methods to study the diffusion model and optimize the linear threshold model parameters, respectively. They do not consider IM as a machine learning problem and are also orthogonal to our problem. [23] studied seed selection in multiple communities. They converted their multi-community coverage maximization problem into a resource allocation problem. Instead of finding seeds from the given network, they used reinforcement learning to study how to assign selected seeds among multiple communities. [21] separates the framework from the breakthrough diffusion model, and uses the learning framework to learn the factors that affect information dissemination. And we believe that the basic diffusion model is very important for the good operation of the existing IM algorithm, because it is different from our problem setting.

Recently, [24] used deep Q-learning to discover subgraph that can act as a surrogate to the entire network. They select the influential node set as the target set of the entire network by applying the traditional IM algorithm on the obtained subgraph, which is orthogonal to our strategy. Besides, DeepIM [25] employs Word2vec model to learn the structural vectors for each node in a network. By using Word2vec, nodes that are observed in many paths are similar to each other, and is then believed to influence each other during information diffusion. Accordingly, DeepIM propose to record a relevant vector to mark down those nodes that appeared as similar to a target node. A node appearing in the most relevant vector is believed to exhibit the most influence. DeepIM is acknowledged as an early effort in addressing IM via network embedding. In comparison, it only learns network embeddings purely from linkage of the network, while our solution incorporate the embedding and influence quality into a unified learning model. Thus, the embedding learning phase in our model is influence-

driven and can better reflect the quality of influence for each node. Secondly, by integrating embedding and influence quality learning into an end-to-end manner, the quality of each particular action in our model is particularly computed with respect to the current embeddings for all the nodes. As a result, the embeddings and action rewards for each selection step are closely associated with the temporal state of selection, while DeepIM does not satisfy this fact. Thirdly, DeepIM requires embedding the nodes from scratch for each target network, thus it requires training for every independent task. In comparison, we provide in PIANO a pair of application models, namely PIANO-E and PIANO@$\langle d \rangle$, that do not require training for addressing IM. GCOMB [26] is a recent solution towards combinatorial optimization over large graph. The key contribution of GCOMB is its hybrid learning model, *i.e.,* combining both supervised and reinforcement learning. By introducing a supervised learning step into the Q-learning framework, GCOMB can filter out 'bad nodes' at an early step, such that it shows excellent scalability comparing with baselines. However, in stochastic processes, such as IM, it is hard to distinguish good nodes from bad ones. In another word, as each influence spread instance is in fact a single sample from the possible world, which is a combinatorial space, 'good nodes' identified in a limited training samples cannot be always 'good'. Similarly, those filtered out cannot be always 'bad'. Hence, GCOMB performs excellent on the graphs where good and bad nodes have already been distinguished in a supervised manner, but shows poor generality, especially when the good and bad cannot be (or there is not enough time to be) distinguished beforehand, which will be justified in our experimental study.

Network Embedding Methods. Network embedding learns a mapping function that converts each node in a network into a vector representation. The learned vectors can be used as features for solving various downstream tasks, such as classification, clustering, and link prediction. The major benefit is that the resulting vector representation can be directly fed into most machine learning models to solve specific problems. When solving the IM problem, we can map the influence information of the nodes in the network to the vector through the network embedding method, and approximate this mapping through the deep reinforcement learning technology to complete the prediction of the expected influence of the node.

Early methods for learning node representations focused primarily on matrix decomposition, which was directly inspired by classical techniques for dimensionality reduction [27]. However, these methods introduce a lot of computational cost. Recent approaches aim to learn the embedding of nodes based on random walks as word contexts. DeepWalk [28] was proposed as the first network embedding method using deep learning technology, which compensates for the gap between language modeling and network modeling by treating nodes as words and generating short random walks. LINE [29] uses the Breadth First Search strategy to generate context nodes, in which only nodes that are up to two hops from a given node are considered to be neighbors. In addition, it uses negative sampling to optimize the Skip-gram model compared to the layered softmax used in DeepWalk. node2vec [30] is a sequence extraction strategy optimized for random walks on DeepWalk framework. It introduces a biased random walk program that combines Breadth First Search and Depth First Search during neighborhood exploration. SDNE [31] captures the nonlinear dependency between nodes by maintaining the proximity between 2-hop neighbors through a deep autoencoder. It designs an objective function that describes both local and global network information, using a semi-supervised approach to fit optimization. There is also a kernel-based approach where the feature vectors of the graph come from various graphics kernels [32]. Structure2vec [4] models each structured data point as a latent variable model, then embeds the graphical model into the feature space, and uses the inner product in the embedded space to define the kernel, which can monitor the learning of the graphical structure. DeepInf [33] presents an end-to-end model to learn the probability of a user's action status conditioned on her local neighborhood.

III. Problem Statement

In this section, we first formally present IM as a learning-based problem. Next, we briefly describe the information diffusion models discussed in these definitions.

A. Problem Definition

Let $G = (V, E, W)$ be a social network, where $|E| = m$, and $|V| = n$. $(u, v) \in E$ represents an edge from node u to node v. Let $W(u, v)$ denote the weight of the edge indicating the strength of the influence. Accordingly, the IM problem can be formally defined as follows.

Definition 1. (Influence Maximization) Given a social network $G = (V, E, W)$, an information diffusion model ψ, integer k, the **influence maximization problem** aims to select k nodes as the seed set S ($S \subseteq V$), such that, under the diffusion model ψ, the expected number of influenced nodes by S, namely $\sigma(S)$, is maximized. The problem can be formulated as $\operatorname{argmax} \sigma(S)$ s.t. $|S| = k$.

As IM is proved to be NP-hard [1], all approximate solutions need to greedily select the next seed with the maximum marginal improvement in expected influence. In particular, let S_i be a partial solution with i seeds (*i.e.,* $i < k$), then in $(i + 1)$-th iteration, an approximate algorithm shall choose a node v, such that $\sigma(S_{i+1}) - \sigma(S_i)$ is maximized, where $S_{i+1} = S_i \bigcup \{v\}$. To facilitate the following discussions, we refer to $\sigma(v, S)$ as the *maximum marginal expected influence* of v given a partial solution S. As $\sigma(v, S)$ is #P-hard to calculate based on v and S, traditional efforts in IM generate unbiased estimates for $\sigma(v, S)$ using a set of RIS samples. In this paper, we solve the IM problem by adopting a completely different strategy *i.e.,* a learning method. In our solution, $\sigma(v, S)$ is not estimated using RIS-based sampling. Instead, it is modeled as a parameterized function and approximated using deep RL. To this end, we introduce the notion of *learning-based* IM *problem*.

The *learning-based* IM *problem* consists of two phases, namely, *learning phase* and *inference phase*. In the learning phase, given a set of homogeneous networks $\mathcal{G} = \{G_1, \ldots, G_\ell\}$ and an information diffusion model ψ, we train a set of parameters Θ such that function $y = \tilde{\sigma}(v, S; \Theta)$ can

be used to approximate $\sigma(v, S)$ as accurately as possible. In the inference phase, given a target network G, integer k and a function $y = \tilde{\sigma}(v, S; \Theta)$ that approximately calculates the marginal influence of v *w.r.t.* the partial solution S, we solve the IM problem in G *w.r.t.* budget k and diffusion model ψ.

B. Diffusion Models

Based on the definition of IM, one can observe that a diffusion model ψ is vital for the selection of seeds. Currently, there exist two popular diffusion models, namely *Linear Threshold* (LT) and *Independent Cascade* (IC). Throughout a diffusion process, a node has two possible states, activated or inactivated. Both models assume that, when a node is activated, its state will not change further.

Linear Threshold (LT) model. The LT model is a special case of the triggering model [34]. To explain the concept briefly, we introduce $N(v)$ (*resp.,* $N^{\eta}(v)$), which is a set of (*resp.,* activated) neighbors of node v where each node has a threshold θ_v. $\forall u \in N(v)$, an edge (v, u) has a non-negative weight $w(v, u) \leq 1$. Given a graph G and a seed set S, and the threshold for each node, this model first activates the nodes in S. Then it starts spreading in discrete timestamps according to the following random rule. In each step, an inactivate node v will be activated if $\sum_{u \in N^{\eta}(v)} w(v, u) \geq \theta_v$. The newly activated node will attempt to activate its neighbors. The process stops when no more nodes are activated.

Independent Cascade (IC) model. Given a graph G and a seed set $S \subset V$, this model first activates the nodes in S, and then starts spreading in discrete timestamps according to the following rule. When a node u is first activated at timestamp t, it gets a chance to activate a node in its neighborhood that is not activated. The success probability of activation is $w(u, v)$. If v is activated successfully, v will become an active node in step $t + 1$ and u can no longer activate other nodes in subsequent steps. This process continues until no new nodes can be activated. In other words, whether u can activate v is not affected by previous propagation.

IV. Learning the Mapping Function

In traditional approximate IM solutions it is inevitable to sample the diffusion phase to generate a set of RIS samples. The cardinality of this set is *at least* as large as the threshold θ. In this paper, we turn to machine learning methods to avoid the traditional diffusion sampling phase in seed selection to make it more generalizable and scalable. In this section, we shall present our PIANO framework that casts the IM problem as finding the optimal policy for selecting the seeds within a deepRL framework. As remarked earlier, the key challenge in IM lies in the estimation of expected influence function $\sigma(v, S)$. In our deep RL framework, $\sigma(v, S)$ plays the role of values (*i.e.,* the expected return for selecting action v in state S), which we approximate with value function $\tilde{\sigma}(v, S; \Theta)$ as accurately as possible. Note that since the IM problem is NP-hard, the ground-truth label for σ is hard to acquire to train a supervised model. In PIANO, we adopt *Deep Q-Network* (DQN) [3], a popular deep RL model to learn the parameters Θ from reward.

Next, we introduce the learning phase of PIANO, which consists of network embedding, training of the parameters Θ, and approximating σ with $\tilde{\sigma}$ for use in DQN. The test phase for selecting nodes according to the learned parameters and predicted influences is detailed in the next section.

A. Embedding the Nodes

Before DQN model can be applied, we shall first embed the nodes into feature vectors based on the topological information. To this end, we need the embedding of each node $v \in V$ as a vector $\mathbf{x_v}$. Among series of embedding methods, *e.g.,* DeepWalk [28], node2vec [30], DeepInf [33], etc., we select Structure2vec [4] to accomplish this step due to the following reasons. Firstly, the other alternatives are 'transductive' embedding methods, that is, they assume that the test graph is observed during training of embeddings. As a result, embeddings extracted across graphs are not consistent, since they only care about intra-graph proximity. In our framework, we aim to use an 'inductive' method to complete the cross-graph extraction of embedded results. This means that the parameters trained in the subgraphs can be applied to the target graph. Secondly, since they are unsupervised network embedding methods (*i.e.,* DeepWalk and node2vec) or supervised for a particular task (*i.e.,* DeepInf), they may not capture the desired information (*i.e.,* expected influence spread) for the IM problem. In our case, the network embedding is trained end-to-end for the optimization target (RL reward), thus the encoded features can be more discriminative for the IM task.

Structure2vec learns nonlinear mapping with discriminative information using stochastic gradient descent, and embeds latent variable models into feature space. It combines the characteristics of nodes, edges and network structure in the target network. These characteristics will recursively aggregate according to the state of the target network. Notably, Structure2vec can learn the embedding of nodes in an *end-to-end* manner through the combination with DQN. That is, the parameters learned from such combination are exclusively suitable for the test network and application scenarios.

Given the current partial solution S, Structure2vec will calculate the q-dimensional feature embedding for each node v ($v \in V$). Firstly, we initialize the vectors of all nodes and set each of them as a q-dimensional zero vector[1]. Then Structure2vec recursively defines the network architecture based on the input network structure G. After I iterations, (I is usually small, set to 4 or less), each node v reaches to the final state, and the embedding at this time can simultaneously take into account the topological features and remote interaction between these nodes. In addition, in the IM scenario, each node also needs a specific flag to indicate whether node v is in the partial solution S or not, which is denoted as a_v. That is, $a_v = 1$ if the node appears in the seed set S, otherwise $a_v = 0$. Because Structure2vec is implemented in a scheme similar to the network model inference process, the nodes' specific label a_v is also recursively aggregated according to the input network topology.

The formula for the update of vectors is as follows:

$$x_v^{(i)} := ReLU(\alpha_1 \sum_{u \in N(v)} x_u^{(i-1)} + \alpha_2 \sum_{u \in N(v)} ReLU(\alpha_3 w(v, u)) + \alpha_4 a_v). \tag{1}$$

[1]In line with [4], q is generally set to 64, it can be adjusted according to the size of the network

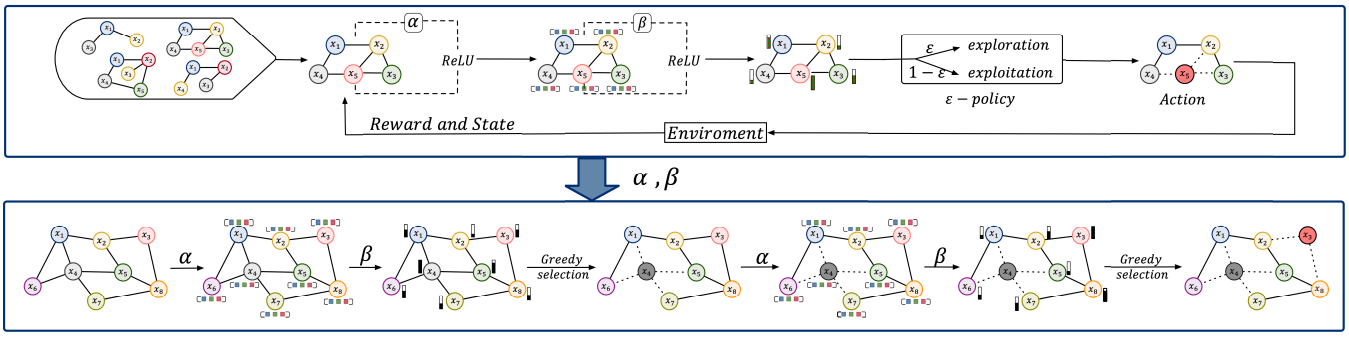


Fig. 1: PIANO framework by incorporating network embedding and deep reinforcement learning. (with $k = 3$)

In Eq. 1, $ReLU$ refers to the non-linear activation Rectified Linear Unit, $N(v)$ is the neighbor set of node v, $x_v^{(i)}$ represents the vector of node v during the i-th iteration, $w(v, u)$ is the weight of edge (v, u), and $\alpha_1, \ldots, \alpha_4$ are the parameters that need to be trained. Although all the neighbors of each node v remain unchanged during the update process, in order to better model the nonlinear mapping of the input space, we add two parameters α_2 and α_3 to construct two independent layers of Multi-Layer Perceptron in the above formula.

It can be seen that the first two items in the equation aggregate the surrounding information by summing up the neighbors of v. Besides, during the iterations, the update formula can also pass information and network characteristics of a node across iterations. When the embeddings of all nodes have been updated, the next iteration begins. After I iterations, the vector of v will contain information about all neighbors within the I-hop neighborhood.

B. The Reinforcement Learning Formulation

In the IM problem, we are unable to acquire sufficient labeled samples for training the parameters Θ in a supervised way, as the exact evaluation of $\sigma(v, S)$ is #P-hard. Hence, we frame the IM problem as policy optimization through deep RL, where the learning task is to find the optimal policy (action sequence) for selecting the seed nodes. The deep RL formulation enables end-to-end learning from the *Agent's Perception to Action* and automatically learns complex features suitable for the task. In our IM scenario, the Agent is the machine (computer program) that is learning to act (selecting seed nodes) optimally in an unknown and uncertain Environment (a network). The whole decision making process is modeled as a Markov Decision Process (MDP), where at each time step t, the agent perceives a state S_t (the current observation of the environment), performs an action A_t and gets a reward R_{t+1} while moving to the next state S_{t+1}, and the process continues for an episode. For the IM problem, we define these RL components as follows:

- *State*: S_t is the current configuration of the network G, where some nodes have already been selected as seed set and some have not. The final state S_k is the configuration where k nodes have been selected. The state representation of G is given by the network embedding method described above.
- *Action*: A_t adds a node v ($v \in \bar{S}$) to the current seed set S. In model-free RL, policy optimization needs values (total expected return) for state-action pairs, which we approximate using $\tilde{\sigma}(v, S_t; \Theta)$, *i.e.,* the expected return for selecting node v at state S_t. We define the value function formally in Eq. 2.
- *Reward*: The environment returns a reward $R_{t+1} \in \mathbb{R}$ for its action $A_t = v$ in state S_t (*i.e.,* for adding node v to the current seed set), and moves to state S_{t+1}. The increment of the influence range is the reward for selecting v node in state S_t, $R_{t+1}(S_t, A_t = v) = \sigma(S_{t+1}) - \sigma(S_t)$.
- *State transition*: When a node $v \in \bar{S}$ is selected into S, a_v will change from 0 to 1. This will in turn change the node embeddings (or state) as shown in Eq. 1.
- *Episode*: A training episode $\mathcal{E}$ is a sequence of state-action-reward values $\mathcal{E} = (S_1, A_1, R_2, \ldots, S_k, A_k, R_{k+1})$.
- *Optimal policy*: The goal in RL is to find the policy, *i.e.,* action sequence $\pi_* = (A_1, \ldots A_k)$ that gives the maximum cumulative reward (or return) defined as $\mathbb{E}[\sum_{i=1}^{k} R(S_i, A_i = v)]$.

We frame IM as a model-free value-based RL problem [35] that finds the optimal policy π_* implicitly in terms of state-action values. Following the RL terminology, we will use Q function to refer to the state-action value function, *i.e.,* $Q(v, S_t; \Theta) = \tilde{\sigma}(v, S_t; \Theta)$. In state S_t, the $Q(v, S_t, \Theta)$ for node/action $A_t = v$ is defined as follows:

$$Q(v, S_t; \Theta) := {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I]). \quad (2)$$

where $x_v^I \in \mathbb{R}^q$ is the vector generated after I iterations; $[,]$ is the concatenation operator; and $\beta_1 \in \mathbb{R}^{128}$, $\beta_2, \beta_3 \in \mathbb{R}^{64 \times 64}$. Because $Q(v, S_t; \Theta)$ is mainly determined by the embedding of the current node v and its surrounding (I-hop) neighbors and their status (selected or not), the Q function is related to the parameters $\alpha_1 \sim \alpha_4, \beta_1 \sim \beta_3$, all of which are learned end-to-end. We denote all these parameters using Θ for simplicity.

The optimal Q function (specifically, the learned Θ) implicitly represents the optimal policy to find the k-best seed nodes. Given the network G, a budget of k nodes, a seed set S, and $\bar{S} = V \backslash S$ as a set of candidate nodes, at each step t, upon evaluating the quality of each node using the Q function, the node with the highest Q value (*i.e.,* marginal expected influence) is added to the seed set S. Formally, $A_t = argmax_{v \in \bar{S}} Q(v, S_t, \Theta)$. After adding a new node v to the seed set, a_v is changed from 0 to 1. In the new state S_{t+1}, all node embeddings need to be updated, and the $Q(v, S_{t+1}; \Theta)$ for the remaining nodes $v \in \bar{S}$ need to be

Algorithm 1: The complete training procedure

Input: Batch of training network, a positive number k,experience replay memory $\mathcal{J}$ to capacity N.
Output: parameters $\Theta = \{\alpha_1, \ldots, \alpha_4, \beta_1, \ldots, \beta_3\}$

1 **for** *episode* $e = 0; e < E$ **do**
2 Given a networks G and set seed set $S = \phi$;
3 parameters $\Theta = \{\alpha_1, \ldots, \alpha_4, \beta_1, \ldots, \beta_3\}$=0;
4 $\varepsilon\_start = 1, \varepsilon\_end = 0.05, max\_iter = 10,000$;
5 **for** $i = 0$ *to* 10 **do**
6 Pre-training parameters Θ with small graphs;
7 **for** $iter$ *to* max_iter **do**
8 update ε through ε_start,ε_end and iter;
9 save training model when $iter = 300$;
10 generate new training network when $iter = 5000$;
11 **for** $i = 0$ *to* $k-1$ **do**
12 **for** $j = 0$ *to* $I-1$ **do**
13 **for** $v \in V$ **do**
14 $x_v^{(j)} := ReLU(\alpha_1 \sum_{u \in N(v)} x_u^{(j-1)} + \alpha_2 \sum_{u \in N(v)} ReLU(\alpha_3 w(v,u)) + \alpha_4 a_v)$;
15 **for** $v \in V$ **do**
16 $Q(v, S, \Theta) = {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I])$;
17 $v_i = \begin{cases} \text{Random node } v \in \bar{S}, & \text{w.p. } \varepsilon \\ argmax_{v \in \bar{S}} Q(v, S, \Theta), & \text{w.p. } 1-\varepsilon \end{cases}$;
18 add v_i to S;
19 update $S, \bar{S}$;
20 **if** $i \geq \delta$ **then**
21 Add tuple $\langle S_i, a_i, \sum_i^{i+\delta} R_i, S_{i+\delta} \rangle$ to $\mathcal{J}$;
22 Sample random batch from $j \sim \mathcal{J}$;
23 Update Θ by SGD with j;
24 **if** ε==0.5 **then**
25 break;

reevaluated to select the next node $A_{t+1} = v'$ in S. This process is continued k times.

C. Model Training via DQN

We use the well-known DQN (Deep Q-Network) algorithm [3], [36] to learn the Q function $Q(v, S; \Theta)$. Since $Q(v, S; \Theta)$ is modeled as a deep neural network (the graph embedding model), it is also called a Q-net. DQN exhibits several advantages over other existing RL algorithms. It gives stability to Q-learning through the use of *experience replay* and Q-targets. Experience replay is a technique where the network uses *off-policy* learning to learn from experiences generated by old policies as opposed to online learning where the experiences are thrown away. This gives DQN sample efficiency compared to policy-based RL (*e.g.,* policy gradients), which only uses on-policy samples. Experience replay also helps to de-correlate the trajectories (action sequences) exhibiting better learning path. The loss function used by our DQN can be defined as:

$$\mathcal{L}(\Theta) = \mathbb{E}[(\underbrace{r + \gamma \max_{v'} \overbrace{Q(v', s'; \Theta)}^{\text{inference net}}}_{\text{Q-target (greedy policy)}} - \underbrace{\overbrace{Q(v, s; \Theta)}^{\text{learner net}}}_{\text{Q-net } (\varepsilon\text{-greedy policy})})^2] \quad (3)$$

where γ is a decaying factor, which controls the importance of future rewards in learning. If $\gamma = 0$, the model will not learn any thing from future (short-sighted), and only pay attention to the current reward; for $\gamma >= 1$, the expected return is continuously accumulated and may diverge. Therefore, γ is generally set to a number slightly smaller than 1 (*e.g.,* 0.95 in our implementation).

The training process is outlined in Algorithm 1, including both graph embedding and deep Q network training.. Our behaviour policy is ε-greedy, which selects a random node with probability ε (exploration), and with probability $(1-\varepsilon)$ it selects a node greedily (exploitation). We vary ε within $0.05 \sim 1$ in our implementation to find the best model. We use a batch of homogeneous networks for training. An episode represents the process of obtaining a complete sequence of a network seed set S (Lines 1-25). For each network, the seed set S is first initialized (Lines 2). According to the embedding process discussed in the paper, we update the nodes' embeddings and calculate the Q value for each node (Lines 11-16). After getting the influence quality of each node, we apply the aforementioned ε-greedy policy. The selected node is then added to the seed set (Lines 17-18). These tuples are added to the replay memory $\mathcal{J}$ from which a random batch of tuples is selected (with IID assumption) to update Θ with SGD (Lines 20-23). For each iteration in a training episode, we shall perform k-rounds of updates. During each update, the embedding for each node should be iteratively updated by I rounds, where each round of updates requires traversing all the neighboring nodes through the edges (*i.e.,* upper bounded by $|E|$). Taking all the above together, the time (*resp.,* space) complexity for training an episode in each iteration requires $O(kI|E||V|)$ (*resp.,* $O(q\delta|V|)$).

An overview of the PIANO framework is depicted in Fig. 1, where the top half depicts the learning phase while the bottom half illustrates the test phase (*i.e.,* seed selection). In particular, given 4 training samples (*i.e.,* networks shown in the left top corner), we first perform learning over them to infer the parameters α, β. For instance, use the second training sample as an example (a 5-nodes network), we encode each node using a $q-$dimensional vector, which shall be iteratively updated via Eq. 1. Further, the embedding of each node is further incorporated into DQN, whose Q-function is outlined in Eq. 2. By training the DQN, we are able to find the values of α, β.

For the seeds selection phase, given the values of α, β, we are able to directly apply them in Eq. 1 and 2 to calculate the node embedding and the reward for each action given the current state. In particular, given a network with 8 nodes shown in the left bottom corner in the figure, by substituting $\alpha_1, \ldots, \alpha_4$ with the learned values into Eq. 1, the embedding (a $q-$dimensional vector) for each node can be acquired (see the second network in the bottom half of Fig. 1). Afterwards, by inserting the learned values of $\beta_1, \ldots, \beta_3$ into Eq. 2, we can acquire the initial Q-value (in DQN) of each node (when there is no seed selected) as well as the reward of each action (the quality of selecting each node into the seeds). Given that, we can select the first seed, which exhibit the highest Q-value (*i.e.,* x_1 in the network, as shown in the 3rd and 4th network in the bottom half of the figure). Accordingly, the reward of each particular action is updated, by eliminating the selected seeds from the network. Afterwards, we shall select other nodes with the highest Q-value into the seeds set iteratively, until there are k (*e.g.,* 3 in the example shown in Fig. 1) seeds.

V. Seeds Selection

The training process results in the learned parameters Θ, which implicitly represent the learned optimal policy. Once the parameters are learned, they can be used to address IM problem in multiple networks. In this section, we describe the

Algorithm 2: Seeds selection procedure

Input: network $G = (V, E, W)$, a positive number k, parameters $\Theta = \{\alpha_1, \ldots, \alpha_4, \beta_1, \ldots, \beta_3\}$
Output: optimal solution S_k

Initialize a seed set $S = \phi$ and $x_v^{(0)} = 0 (v \in V)$.
for $j = 1$ *to* I **do**
  for $v \in V$ **do**
    $x_v^{(j)} := ReLU(\alpha_1 \sum_{u \in N(v)} x_u^{(j-1)} + \alpha_2 \sum_{u \in N(v)} ReLU(\alpha_3 w(v,u)) + \alpha_4 a_v)$
for $v \in V$ **do**
  $Q(v, \emptyset; \Theta) = {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I])$
$S_k \leftarrow$top-k node v with the highest $Q(v, \emptyset; \Theta)$;

test phase of PIANO framework, *i.e.,* online selection of seeds.

A. Generating Result Set via Learned Function

As we have learned all the parameters in $\tilde{\sigma}(v, S; \Theta)$, we are able to directly predict the expected marginal influence for each node. Afterwards, intuitively, it is natural for one to follow the existing hill-climb strategy to iteratively select the best node that exhibits the highest predicted marginal influence. However, our empirical and theoretical studies [37] reveal that the order of the nodes with respect to their Q values remains almost unchanged whenever we select a seed and recompute the network embeddings as well as the Q values. Therefore, during the seeds selection phase, instead of iteratively select-and-recompute the embeddings and Q values according to each seed insertion, we simplify the procedure into only one iteration, by embedding only once and selecting the top-k nodes with the maximum Q.

The seeds selection process is outlined in Algorithm 2. Given a network G and a budget k, we first initialize each node in the network as a p-dimensional zero vector and perform embeddings according to Equation 1 using the learned parameters $\alpha_1, \ldots, \alpha_4$. Afterwards, the influence quality of each node v is evaluated using Equation 2 via the learned parameters $\beta_1, \ldots, \beta_3$. Finally, the node with the top-k influence quality is added to the seed set.

The time complexity of the selection process is determined by three parts. First, in the embedding phase, the complexity is influenced by the number of nodes $|V|$ and the number of neighbors $|\overline{N(v)}|$. As I is usually a small constant (*e.g.,* $I = 4$) in Algorithm 2, network embedding takes $O(|V| \times |\overline{N(v)}|)$. Second, after all the nodes in the network are embedded, the quality of nodes in the graph is evaluated using the formula $Q(v, S, \Theta) = {\beta_1}^T ReLU([\beta_2 \sum_{u \in V} x_u^I, \beta_3 x_v^I])$. Since the values of Θ have been learned, this step is influenced by the time taken to find the neighbors of node v. Hence, it takes $O(|V| \times |\overline{N(v)}|)$ time. Finally, selecting the optimal node according to Q function and adding the node to the set S takes $O(|V|)$. Consequently, the total time (*resp.,* space) complexity for seeds selection is $O(|V| \times |\overline{N(v)}|)$ (*resp.,* $O(q|V|)$).

B. Applying PIANO in Practice

As a learning-based framework, we need to pretrain the embedding and Q function parameters offline using a group of training networks. Intuitively, the training and testing (*i.e.,* target) networks should be homogeneous in terms of topology such that the quality of the learned parameters can be guaranteed. In general, the homogeneity in terms of topology can be reflected w.r.t topological properties, *e.g.,* degree distribution, spectrum, etc. Therefore, in order to select seeds within a targeted network, we need to train the required parameters Θ in a group of homogeneous networks offline. Afterwards, the trained model can be further used to select seeds in a target network. In the following, depending on the specific characteristic of target networks, we shall present three pretraining strategies, namely PIANO-S, PIANO-E and PIANO@$\langle d \rangle$, respectively.

PIANO-S: applying PIANO in large-scale stationary networks. Consider a large-scale stationary network without any evolution log. We can turn to subgraph-sampling technique to generate sufficient homogeneous training networks. Afterwards, with learned parameters Θ from these small networks, we can address IM over the target large-scale network. In order to ensure that the topological features of the sampled training subgraphs are as consistent as possible with the original large-scale target network, we evaluate different sampling algorithms following the same framework adopted in [38], [39]. In particular, we apply different sampling methods to real large-scale networks and sample subgraphs over a series of sample fractions $\varphi = [1\%, 5\%, 10\%, 15\%]$. Following the methods introduced in [38], Kolmogorov-Smirnov D-statistic is adopted to measure the differences between the network statistics distributions of the sampled subgraphs and the original graph, where D-statistic is typically applied as part of the Komogorov-Smirnov test to reject the null hypothesis. It is defined as $D = \max_x\{|F_0(x) - F(x)|\}$, where x denotes the range of random variables; F_0 and F are two empirical distribution functions of x. Our experimental results show similar phenomenon with the benchmarking papers [38], [39]. That is, Topology-Based Sampling is superior to Node Sampling and Edge Sampling in the distribution of graph features such as degree distribution and clustering coefficient distribution. So in the next section, we adopt Topology-Based sampling methods in our model and compare several existing Topology-Based subgraph sampling methods, including Breadth First Sampling (BFS)[40], Simple Random Walk (SRW) and its variants, namely Random Walk with Flyback (RWF) and Induced Subgraph Random Walk Sampling (ISRW) [41], as well as Snowball Sampling (SB)[42], a variant of Breadth First Search which limits the number of neighbors that are added to the sample. Notably, as the subgraph-sampling technique is beyond the focus of this work, we do not discuss the detailed techniques for these methods.

PIANO-E: applying PIANO in evolutionary networks. Almost all existing social networks evolve with time as nodes and edges are inserted or deleted. However, the structural properties remain relatively stable [43]. During the evolution of a particular real-world network, we advocate that any two historical snapshots of the same network are homogeneous to each other. By leveraging on the aforementioned technique, we briefly describe how PIANO can address the IM problem in dynamic networks via time-based sampling. In particular, given a dynamic network G, whose snapshot at time t is referred to as G_t, we can apply PIANO in the following way. During the *Learning phase*, we can train the model using a series of temporal (sampled) network snapshots (*i.e.,* $\mathcal{G} = \{G_{t1}, \ldots, G_{t\ell}\}$). For the *Seeds selection phase*, the

TABLE I: Datasets

Dataset	n	m	Type	Avg.Degree
HepPh	34K	421K	Directed	9.83
DBLP	317K	1.05M	Undirected	3.31
LiveJournal	4.85M	69M	Directed	6.5
Orkut	3.07M	117.1M	Undirected	4.8

trained model can be used to select the best seed set in an arbitrary snapshot of G. Besides, if any snapshot of the network is very large, we can also adopt the subgraph sampling method mentioned above. Based on this strategy, we can accomplish the seeds selection task over a target network in real-time, although it keeps on evolving. This provides a practical solution for the IM problem in evolutionary networks.

PIANO@$\langle d \rangle$: applying PIANO with our pre-trained model pool. In addition to the above approaches, we provide a user-friendly method for applying PIANO in practice, *i.e., pretrained once and apply everywhere*. We build a model pool by pretraining a series of PIANO models, each is learned over 200 synthetic networks (each contains 500 nodes) with identical average degree at $3, 4, \ldots, 9$, respectively via SNAP tool[2], using its *ForestFire* graph generation model (method name: GenForestFire), which generates random graph with given probabilities (under Forest Fire graph model). By varying the probabilities, we generate a series of random graphs (each contains 500 nodes) with different average degrees. We further group the graphs with an average degree at $[d-0.05, d+0.05]$ to group d, where $d = 3, 4, \ldots, 9$. For each group, we train an independent PIANO model. For instance, we can train a PIANO model from 200 synthetic networks with average degree of 5, referred to as PIANO@5. Given a target network G (with average degree d) for IM, we can find from our model pool PIANO@$\langle d \rangle$, where $\langle \cdot \rangle$ refers to the closest integer of d, and directly apply that on G without training again. The detailed justification and evaluation of the pretrained model pool is given in the next section.

VI. EXPERIMENTS

In this section, we evaluate the performance of the PIANO framework. We compare our model with two state-of-the-art traditional IM solutions, namely IMM [2] and SSA [5], as suggested by a recent benchmarking study [11], as well as a latest deep learning based solution, namely GCOMB [26]. Recall that the efforts in [14], [15], [44], [45] are designed for competitive IM and hence are orthogonal to our problem. In line with all the IM solutions, we evaluate the performances from two aspects, namely computational efficiency and influence quality. Besides, as a learning-based solution, we also justify our model generality within evolutionary scenarios. All the experiments were performed on a machine with Intel Xeon CPU (2.2GHz, 20 cores), 512GB of DDR4 RAM, Nvidia Tesla K80 with 24GB GDDR5, running Ubuntu 16.04[3].

A. Experimental Setup

Datasets. In the experiments, we present results on four real-world social networks, taken from the *SNAP* repository[4], as shown in Table I. In these four datasets, *HepPh* and *DBLP* are citation networks, *LiveJournal* and *Orkut* are the largest online social networks ever used in influence maximization. For each real-world network, we use the following sampling methods: Breadth First Sampling (BFS) [40], Simple Random Walk (SRW), Induced Subgraph Random Walk Sampling (ISRW) [41], and Snowball Sampling (SB) [42].

Diffusion Models. PIANO can be easily adapted to different diffusion models. For instance, we can simply revise the Reward definition to switch from IC model to LT model. As our experimental results under both LT and IC models are qualitatively similar, we mainly report the results under the IC model here. In IC model, each edge $w(u, v)$ has a constant probability p. In vast majority of the IM techniques, $w(u, v)$ takes the value of 0.5 assigned to all the edges of the network. In order to fairly calculate the expected range of influence for the three approaches, we first record the seed set of each algorithm independently, and then perform $10,000$ simulated propagations based on the selected seeds. Finally, we take the average result of $10,000$ simulations as the number of influenced nodes for each tested approach.

Training data generation: Use ISRW, BFS, SRW, Snowball sampling methods to sample from the original network. The size of the sampled subgraph is determined by the size of the original graph. In our experiment, we set the number of edges of the sampled subgraph to 0.1% of the entire corresponding graph. Although we do not limit the number of points in the subgraph, we hope that the method you choose should be able to maintain the ratio between the number of points and edges in the original graph.

Parameter settings: For the learning rate, we use exponential decay after a certain number of steps, where the decay factor γ is fixed to 0.95. We also anneal the exploration probability ε from 1.0 to 0.05 in a linear way. We set the batch size, *i.e.,* the number of samples extracted from M each time, as 64. Besides, we set δ as 5 and the learning rate of SGD to 0.001. We set the embedding dimension as 64.

As suggested in IMM [2], we set $\epsilon = 0.1$ throughout the experiments in both IMM and SSA. It should be noted that the seed set produced by SSA is not constant. Therefore, for SSA and IMM, we report the average results over 100 independent runs (*i.e.,* 100 independent results seed sets, each of which is averaged for $10,000$ simulations).

B. Experimental Results

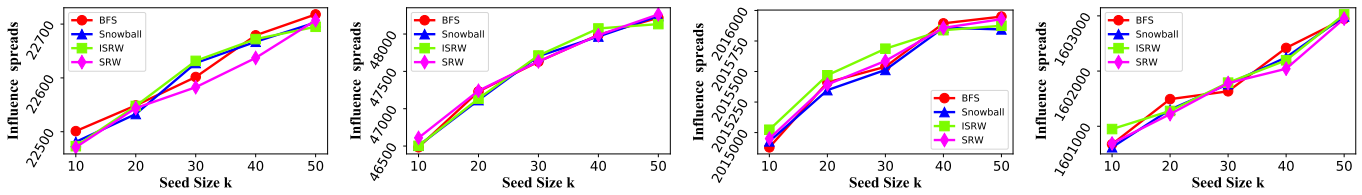
Training issues. As remarked earlier, during DQN training, we need to obtain a batch of training graphs by sampling the real-world network. In our experiments, we mainly compare the aforementioned four sampling methods (BFS, ISRW, Snowball, and SRW). The results are shown in Fig. 2. Observe that the results of trained PIANO-S obtained by using different Topology-Based sampling methods have negligible differences, indicating that these sampling methods have little impact on the result seeds quality. Therefore, we chose to use simple and mature BFS (Breadth First Sampling) as the default method in rest of the experiments.

Given the sampled sub-networks, we train the DQN parameters using ε-greedy exploration described in Section IV-C. That is, the next action is randomly selected with probability

[2]https://snap.stanford.edu/snap/

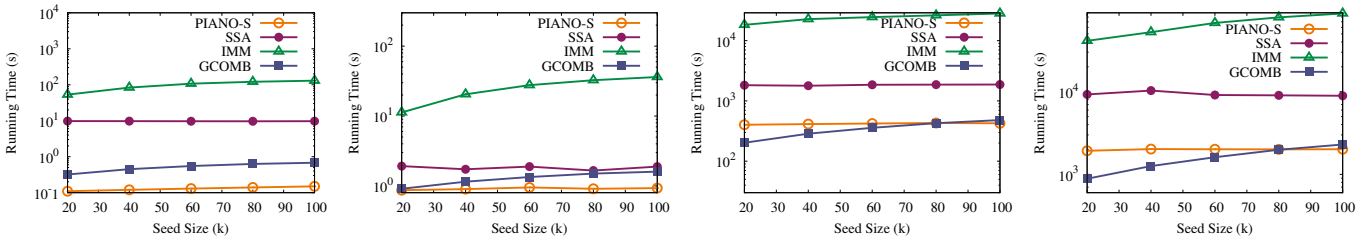
[3]The src code is available in https://github.com/lihuixidian/PIANO.

[4]https://snap.stanford.edu/data/



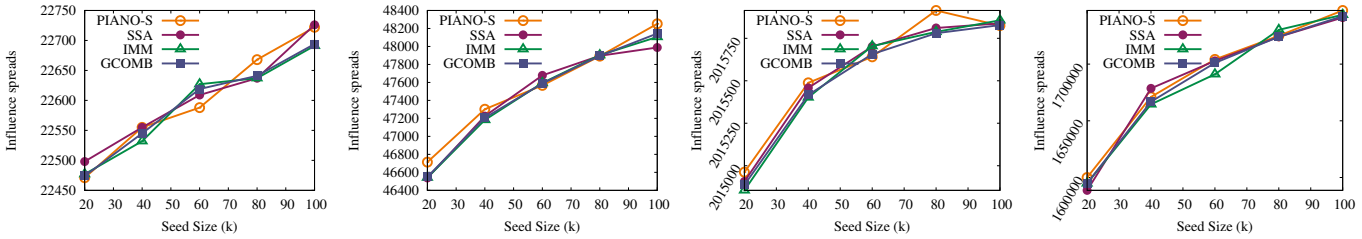
(a) **HepPh** (b) **DBLP** (c) **LiveJournal** (d) **Orkut**

Fig. 2: Influence spreads with different sampling algorithms.



(a) **HepPh** (b) **DBLP** (c) **LiveJournal** (d) **Orkut**

Fig. 3: Running time.



(a) **HepPh** (b) **DBLP** (c) **LiveJournal** (d) **Orkut**

Fig. 4: Influence spreads quality.

ε, and the action of maximizing the Q function is selected with probability $1-\varepsilon$. During training, ε is linearly annealed from 1 to 0.05 in ten thousand steps. The training procedure terminates when the value of ε is less than 0.05 or the training time exceeds 3 days. For both *HepPh*, *DBLP*, the training phases terminate at 2.5 and 23.5 hours, respectively; for *LiveJournal* and *Orkut*, we limit the training time within 3 days and apply the learned parameters to node selection.

Notably, *the training phase is performed only once and we do not need to train while running an* IM *query*. The study of PIANO-E on evolutionary networks further justifies that, once trained, it can be applied many times to address the IM problem.

Computational efficiency. We compare the running times of the three algorithms by varying k from 20 to 100. The results on *HepPh*, *DBLP*, *LiveJournal* and *Orkut* are reported in Fig. 3. We can make the following observations. In terms of computational efficiency, PIANO-S significantly outperforms IMM and SSA by a huge margin. Regardless of the value of k, the cost of PIANO-S is significantly less than that of IMM and SSA. Admittedly, traditional solution like IMM and SSA do not require training time, while PIANO, especially PIANO-S has to perform training before answering IM. In comparison, as we shall see in both PIANO-E and PIANO@$\langle d \rangle$, once a model is trained in a network, it can be used in IM tasks over a series other networks. In practical applications, answering such IM problems only need to perform seeds selection under PIANO-E and PIANO@$\langle d \rangle$, which does not include any training time.

Particularly, when the number of seed nodes selected is small, the performance gain is more prominent. In addition, as we select the nodes top k ranked nodes at the same time, when we select seed sets of different sizes, the time variation is small, but the premise of this experimental result is that we have sorted the scores of all nodes (*i.e.,* cost is $O(nlogn)$). However, if k nodes are directly selected from the unordered set, the time required is only $n \times k$. Therefore, the running time of PIANO-S has increased significantly over k.

Influence quality. Next, we compare the quality of the seed sets generated by IMM, SSA and PIANO-S. For each algorithm we set the optimal parameter values according to their original papers and then evaluate their quality. As can be seen from Fig. 4, PIANO-S is as good as IMM and SSA in real networks of different sizes, and the growth of influence spread with k have few minor fluctuations. Note that the scales of the vertical axis in the four figures are different. In comparison, our method can choose superior result set, and the results are stable across 100 runs. Importantly, PIANO-S produces the same or even better quality results as the state-of-the-art. In addition, the author proposed in IMM that ϵ is a tunable parameter. When $\epsilon = 0.1$, the quality of the IMM seed set is equivalent to ours. If we increase ϵ although the time efficiency will increase, the quality of the result will also decrease.

Performance of PIANO-E in evolutionary networks. We evaluate the performance of PIANO on evolutionary networks

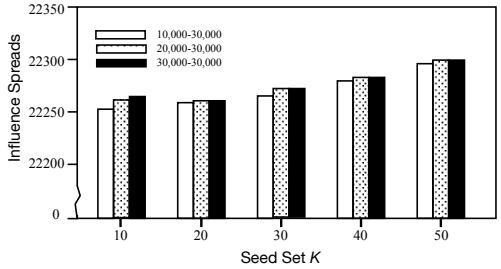


Fig. 5: Performance of PIANO-E in evolutionary networks.

following the strategy discussed in Section V-B. Among all the tested datasets, *HepPh* is associated with evolution logs, *i.e.,* the time-stamps when each node/edge is inserted to the network. Therefore, following the proposed implementation model of PIANO-E in evolutionary network, we train the model using a series of temporal snapshots for *HepPh* networks when it has $10,000, 20,000$, and $30,000$ nodes, respectively. Then the three trained models are tested on a future snapshot of the network containing $30,000$ nodes. As can be seen in Fig. 5, the trained model from $10,000$-nodes snapshot and that from $30,000$-nodes ones have little differences in terms of the influence quality. For instance, when the number of seeds is 50, the difference between $10,000$-nodes model and $30,000$-nodes model in the influence spread is 224, which is about 1% of the total influenced nodes. Therefore, in real-world dynamic networks, PIANO-E can be easily trained using earlier snapshot or even the subnetworks of some snapshots. When the network evolves and expands, we can continue to use the pretrained model for selecting seeds in a larger network snapshot.

Moreover, we compare the generality of PIANO-E and GCOMB by varying the size of training snapshots and test ones, based on the evolution logs of *HepPh*. To highlight the effectiveness of trained model at an earlier snapshot over the IM task at a later one, we showcase the relative influence quality comparing with the influence spread of (training and) running PIANO-S at the later snapshot. The results of both PIANO-E and GCOMB are shown in Table II, where k is fixed as 50. For instance, in the first row, we train PIANO-E and GCOMB using the 5000-node snapshot, and apply both models for seeds selection at a later snapshot (*i.e.,* with 10000, 20000, 30000 nodes respectively), without training again. We compute the relative ratio on the number of influenced nodes with respect to training and applying PIANO-S at the snapshots (*i.e.,* 10000, 20000, 30000 nodes), respectively. Obviously, in all cases, PIANO-E exhibits almost the same influence quality comparing with training from scratch. In comparison, GCOMB shows poor performance in this setting, up to 15% decrease in the influence quality. This fact justify our discussion in Section II that GCOMB highly relies on supervision of the nodes. In scenarios where supervision is not practical, *e.g.,* evolutionary cases, the effectiveness shall drop down significantly.

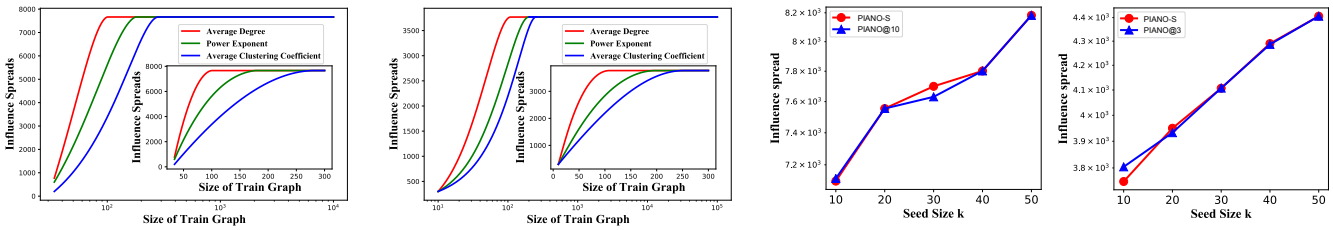
Pre-trained model pool and generalization. According to the above experiments, whenever one wants to apply PIANO to address IM problem in a target network G, she by default needs to sample a group (*e.g.,* 200) of subgraphs of G and train PIANO first. In this experiment, we extensively discuss how this training phase can be pre-performed and avoided in practical IM tasks. That is, *is it possible to provide a pretrained*

TABLE II: Generality over evolutionary network.

	Test snapshot					
	10000		20000		30000	
Training snapshot	PIANO-E	GCOMB	PIANO-E	GCOMB	PIANO-E	GCOMB
5000	96.8%	84.4%	94.6%	80.5%	92.9%	78.8%
10000	100%	100.1%	100%	96.2%	99.9%	94.3%
15000			99.9%	98.1%	99.9%	95.4%
20000			100%	99.9%	99.9%	97.6%
25000					99.9%	98.8%
30000					100%	99.9%

PIANO *model that can be directly applied (*i.e., *without training phase) over a new target network for* IM *task?* To answer the question, we generate a series of synthetic networks with *SNAP* tool for training and evaluate the performance of the corresponding learned model. We test its performance with respect to the size and the generation schemes, and compare its performance with the approach of sampling from the target network. Herein, generation scheme refers to the golden rule we adopt to guide the synthetic graph generation procedure. In particular, we can generate synthetic graphs with a pre-defined average degree (*resp.,* power exponent of power-law degree distribution, average clustering coefficient). Based on this group of study, we are able to unveil the correlation between the model performance and training graphs, and also provide a high-level suggestion on how the training graphs should be selected. We ensure that a particular network topological property of the generated graph is consistent with the original graph, and then change the size of the generated graphs, train each batch of generated graphs to observe the changes in the quality of the results. The results are shown in Fig. 6a and 6b ($p = 0.05$).

Observe that as the size of training graphs increases from 10^1 to 10^2, the influence spreads for the learned model also increase. As shown in Fig. 6a (*resp.,* 6b), when the sizes of training graphs are larger than 10^2, the performance of the learned model remains unchanged. This means that small training graphs are sufficient to train an accurate model as long as the network topology of the training graph is consistent with the target one. In addition, the training efficiency of small graphs is superior to large graphs, which can be reduced from a few days to a few hours. It can be seen from Fig. 6 that the quality of the results obtained by the selection of different topological properties shows negligible differences. Among the three properties, we recommend that the average degree of the generated graph and the original graph should be the same to achieve the expected result quality. Besides, we also compare the performance of training from synthetic graph and sampled graph with the same size and average degree. Notably, among all datasets, there is negligible difference between both approaches. That is, a PIANO model trained over some synthetic graphs can be directly applied to a real-world network as long as they exhibit the same average degree. Therefore, we build a model pool by pretraining a series of PIANO models, each is learned over 200 synthetic networks (each contains 500 nodes) with identical average degrees (set to $3, 4, \ldots, 9$, respectively). Given a target network G (with average degree d), we can select from our model pool PIANO@$\langle d \rangle$ and directly apply it on G without training again. To justify the effect of our



(a) **HepPh** (b) **DBLP** (c) **HepPh** (d) **DBLP**

Fig. 6: Performance of PIANO@$\langle d \rangle$: (a-b) varying the size of training graph and topological properties; (c-d) comparison with sampling-based PIANO-S.

model pool, we compare the performance of PIANO@$\langle d \rangle$ with PIANO-S, which trains the model from scratch (over subgraph samples of the target network) in Fig. 6c and 6d. Observe that there is little difference between PIANO-S and PIANO@$\langle d \rangle$.

Summary of the results. Given the above experimental study, we conclude the effectiveness of PIANO as follows. Firstly, after training over the target network, PIANO-S achieves the same effectiveness with the state-of-the-art solutions, including IMM and SSA. Secondly, as PIANO-S requires training from scratch each time, we propose PIANO@$\langle d \rangle$, which provide a pre-trained model pool. Given the topology property of the target network, *e.g.,* average degree, we can directly perform seeds selection based on a particular trained model from the pool of PIANO@$\langle d \rangle$. Empirical study demonstrate that PIANO@$\langle d \rangle$ achieves similar effectiveness as PIANO-S, while avoiding the costly training phase. Thirdly, for dynamic networks, our PIANO-E generalizes well, without reperforming the training, to other evolutionary snapshots of the network even if the number of nodes has been enlarged by 2 times.

VII. Conclusions

In this work, we have presented a novel framework, PIANO, to address the IM problem by exploiting deep reinforcement learning technique. Our framework incorporates both DQN and network embedding methods to train superior approximation of the σ function in IM. Compared to state-of-the-art sampling-based IM solutions, PIANO can avoid the costly diffusion sampling phase. By training with sampled subnetworks once, the learned model can be applied many times. Therefore, we are able to achieve superior efficiency compared to state-of-the-art classical solutions. Besides, the result quality in terms of influence spread of PIANO is the same or better than the competitors. As the seminal effort towards the paradigm of learning-based IM solution, PIANO demonstrates exciting performance in terms of result quality and running time. We believe that our effort paves the way for a new direction to address the challenging classical IM problem.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 61972309), XD-Inspur DB Innovation Lab grant.

References

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

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

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML*, 2016, pp. 2702–2711.

[5] 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, pp. 695–710.

[6] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *KDD*, 2007, pp. 420–429.

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

[8] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *KDD*, 2009, pp. 199–208.

[9] Q. Jiang, G. Song, G. Cong, Y. Wang, W. Si, and K. Xie, "Simulated annealing based influence maximization in social networks," in *AAAI*, 2011.

[10] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "SIMPATH: an efficient algorithm for influence maximization under the linear threshold model," in *ICDM*, 2011, pp. 211–220.

[11] A. Arora, S. Galhotra, and S. Ranu, "Debunking the myths of influence maximization: An in-depth benchmarking study," in *SIGMOD*, 2017, pp. 651–666.

[12] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *SODA*, 2014, pp. 946–957.

[13] K. Huang, S. Wang, G. S. Bevilacqua, X. Xiao, and L. V. S. Lakshmanan, "Revisiting the stop-and-stare algorithms for influence maximization," *PVLDB*, vol. 10, no. 9, pp. 913–924, 2017.

[14] K. Ali, C. Wang, and Y. Chen, "Boosting reinforcement learning in competitive influence maximization with transfer learning," in *WI*, 2018, pp. 395–400.

[15] S. Lin, S. Lin, and M. Chen, "A learning-based framework to handle multi-round multi-party influence maximization on social networks," in *KDD*, 2015, pp. 695–704.

[16] H. Huang, Z. Meng, and S. Liang, "Recurrent neural variational model for follower-based influence maximization," *Inf. Sci.*, vol. 528, pp. 280–293, 2020.

[17] K. Ali, C. Wang, M. Yeh, and Y. Chen, "Addressing competitive influence maximization on unknown social network with deep reinforcement learning," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2020, The Hague, Netherlands, December 7-10, 2020*, M. Atzmüller, M. Coscia, and R. Missaoui, Eds. IEEE, 2020, pp. 196–203.

[18] T. Carnes, C. Nagarajan, S. M. Wild, and A. van Zuylen, "Maximizing influence in a competitive social network: a follower's perspective," in *EC*, 2007, pp. 351–360.

[19] H. Li, S. S. Bhowmick, J. Cui, Y. Gao, and J. Ma, "Getreal: Towards realistic selection of influence maximization strategies in competitive networks," in *SIGMOD*, 2015, pp. 1525–1537.

[20] S. Tian, S. Mo, L. Wang, and Z. Peng, "Deep reinforcement learning-based approach to tackle topic-aware influence maximization," *Data Sci. Eng.*, vol. 5, no. 1, pp. 1–11, 2020.

[21] S. Vaswani, B. Kveton, Z. Wen, M. Ghavamzadeh, L. V. S. Lakshmanan, and M. Schmidt, "Model-independent online learning for influence maximization," in *ICML*, 2017, pp. 3530–3539.

[22] T. Cao, X. Wu, X. T. Hu, and S. Wang, "Active learning of model parameters for influence maximization," in *PKDD*, 2011, pp. 280–295.

[23] X. Tong, H. Fan, X. Wang, J. Li, and X. Wang, "Seeds selection for influence maximization based on device-to-device social knowledge by reinforcement learning," in *KSEM*. Springer, 2020, pp. 155–167.
[24] H. Kamarthi, P. Vijayan, B. Wilder, B. Ravindran, and M. Tambe, "Influence maximization in unknown social networks: Learning policies for effective graph sampling," in *AAMAS*, 2020, pp. 575–583.
[25] M. M. Keikha, M. Rahgozar, M. Asadpour, and M. F. Abdollahi, "Influence maximization across heterogeneous interconnected networks based on deep learning," *Expert Systems with Applications*, vol. 140, pp. 1–11, 2020.
[26] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. K. Singh, "GCOMB: learning budget-constrained combinatorial algorithms over billion-sized graphs," in *NeurIPS*, 2020.
[27] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *NIPS*, 2002, pp. 585–591.
[28] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*. ACM, 2014, pp. 701–710.
[29] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. Alemi, "Watch your step: Learning graph embeddings through attention," *CoRR*, vol. abs/1710.09599, 2017.
[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*. ACM, 2016, pp. 855–864.
[31] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.
[32] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.
[33] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *KDD*, 2018, pp. 2110–2119.
[34] D. Kempe, J. M. Kleinberg, and É. Tardos, "Influential nodes in a diffusion model for social networks," in *ICALP*, 2005, pp. 1127–1138.
[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.
[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.
[37] 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.
[38] N. K. Ahmed, J. Neville, and R. R. Kompella, "Network sampling: From static to streaming graphs," *TKDD*, vol. 8, no. 2, pp. 7:1–7:56, 2013.
[39] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *KDD*, 2006, pp. 631–636.
[40] C. Doerr and N. Blenn, "Metric convergence in social network sampling," in *HotPlanet@SIGCOMM*, 2013, pp. 45–50.
[41] C. Lee, X. Xu, and D. Y. Eun, "Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling," in *SIGMETRICS*. ACM, 2012, pp. 319–330.
[42] L. A. Goodman, "Snowball Sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.
[43] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *KDD*, 2006, pp. 611–617.
[44] L. Sun, W. Huang, P. S. Yu, and W. Chen, "Multi-round influence maximization," in *KDD*, 2018, pp. 2249–2258.
[45] K. Han, K. Huang, X. Xiao, J. Tang, A. Sun, and X. Tang, "Efficient algorithms for adaptive influence maximization," *PVLDB*, vol. 11, no. 9, pp. 1029–1040, 2018.

Hui Li Hui Li is currently a professor with the School of Computer Science and Technology, Xidian University, China. His research interests include data mining, database, and privacy-preserving query. He has published many papers in major venues in these areas such as SIGMOD, VLDB, ICDE, SIGKDD, the IEEE Transactions on Knowledge and Data Engineering, and the VLDB Journal.

Mengting Xu Mengting Xu is a master student with the School of Cyber Engineering, Xidian University, China. Her research interests include social network mining.

Sourav S Bhowmick Sourav S Bhowmick is an associate professor with the School of Computer Science and Engineering, Nanyang Technological University. His current research interests include data management, data analytics, computational social science, and computational systems biology. He has published many papers in major venues in these areas such as SIGMOD, VLDB, ICDE, SIGKDD, MM, the IEEE Transactions on Knowledge and Data Engineering, the VLDB Journal, and the Bioinformatics.

Joty Shafiq Rayhan Joty Shafiq Rayhan is an associate professor with the School of Computer Science and Engineering, Nanyang Technological University. His current research interests include natural language processing, and machine learning systems. He has published many papers in major venues in these areas such as ACL, ICML, NeurIPS, SIGMOD, AAAI, and ICLR.

Changsheng Sun Changsheng Sun is currently a PhD student with the School of Computing, National University of Singapore. His current research interests include trustworthy machine learning. He has published several papers in NeurIPS, ICSE.

Jiangtao Cui Jiangtao Cui received the MS and PhD degrees in computer science from Xidian University, Xi'an, China, in 2001 and 2005, respectively. Between 2007 and 2008, he has been with the Data and Knowledge Engineering group working on high-dimensional indexing for large scale image retrieval, in the University of Queensland, Australia. He is currently a professor with the School of Computer Science and Technology, Xidian University, China. His current research interests include data and knowledge engineering, data security, and high-dimensional indexing.