ToupleGDD: A Fine-Designed Solution of Influence Maximization by Deep Reinforcement Learning

Tiantian Chen, Siwen Yan, Jianxiong Guo, *Member, IEEE*, and Weili Wu, *Senior Member, IEEE*

***Abstract*— Aiming at selecting a small subset of nodes with maximum influence on networks, the influence maximization (IM) problem has been extensively studied. Since it is #P-hard to compute the influence spread given a seed set, the state-of-the-art methods, including heuristic and approximation algorithms, are faced with great difficulties such as theoretical guarantee, time efficiency, generalization, and so on. This makes it unable to adapt to large-scale networks and more complex applications. On the other side, with the latest achievements of deep reinforcement learning (DRL) in artificial intelligence and other fields, lots of work have been focused on exploiting DRL to solve combinatorial optimization (CO) problems. Inspired by this, we propose a novel end-to-end DRL framework, ToupleGDD, to address the IM problem in this article, which incorporates three coupled graph neural networks (GNNs) for network embedding and double deep *Q*-networks (DQNs) for parameters learning. Previous efforts to solve the IM problem with DRL trained their models on subgraphs of the whole network and then tested them on the whole graph, which makes the performance of their models unstable among different networks. However, our model is trained on several small randomly generated graphs with a small budget and tested on completely different networks under various large budgets, which can obtain results very close to IMM and better results than OPIM-C on several datasets and shows strong generalization ability. Finally, we conduct a large number of experiments on synthetic and realistic datasets and experimental results prove the effectiveness and superiority of our model.**

***Index Terms*— Deep reinforcement learning (DRL), generalization, graph neural networks (GNNs), influence maximization (IM), social network.**

I. Introduction

ONLINE social platforms, such as Twitter and LinkedIn, have shown to be one of the most effective ways for people to communicate and share information with each other. Many companies have turned to social networks as a primary way of promoting products and use "word of mouth" effects to maximize product influence. To maximize earned profits, companies may apply a variety of methods, such as distributing free samples or coupons. Many works have focused on the diffusion phenomenon on social networks. Kempe et al. [1] first formally defined the influence maximization (IM) problem as a combinatorial optimization (CO) problem and presented the independent cascade (IC) model and linear threshold (LT) model to depict the information diffusion process.

It has been proved IM is NP-hard, and the objective (influence spread) is monotone and submodular under IC and LT models [1]. Kempe et al. [1] used a Greedy algorithm to solve IM, which selects the node with a maximum marginal gain of influence spread and can achieve $(1-1/e-\epsilon)$-approximation ratio. However, it is #P-hard to compute the influence spread of a seed set under both IC [2] and LT model [3]. The hardness of estimating the influence spread lies in the randomness of the probabilistic diffusion models, that is, random choices and diffusion paths. The key to approximating the influence spread is to effectively and efficiently sample diffusion paths. Kempe et al. [1] used the Monte Carlo method to simulate diffusion paths, which can obtain good estimations when simulation times are large enough. But it is too time-consuming. Borgs et al. [4] first proposed a novel reverse influence sampling (RIS) technique to reduce the running time. However, RIS still incurs significant computational overheads in practice to obtain a good solution. Subsequently, a series of algorithms based on RIS were proposed, such as TIM/TIM+ [5], IMM [6], SSA/D-SSA [7], and OPIM-C [8], which can achieve $(1-1/e-\epsilon)$-approximation solution with high probability when the number of generated random reachable reverse (RR) sets are large enough and were recognized as the state-of-the-art methods to solve IM. However, these algorithms, such as IMM, still have scalability issues in large insurance networks.

On the other hand, the development of deep learning and reinforcement learning (RL) has blossomed in the last few years, resulting in an increasing number of works addressing the CO problem by learning-based methods. A natural question is: can we estimate the influence spread by learnable parametric function and avoid costly sampling random RR sets? The answer is Yes. Khalil et al. [9] first designed an end-to-end deep RL (DRL) framework, S2V-DQN, to solve the common CO problem. Then, Manchanda et al. [10] proposed a supervised deep-learning-based model for the CO problem, called GCOMB, where IM was used as an example to test the

Manuscript received 6 November 2022; revised 7 March 2023; accepted 26 April 2023. This work was supported in part by the NSF under Grant 1907472 and Grant 1822985 and in part by the National Natural Science Foundation of China (NSFC) under Grant 62202055. *(Tiantian Chen and Siwen Yan contributed equally to this work.) (Corresponding author: Jianxiong Guo.)*

Tiantian Chen, Siwen Yan, and Weili Wu are with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: tiantian.chen@utdallas.edu; siwen.yan@utdallas.edu; weiliwu@utdallas.edu).

Jianxiong Guo is with the Advanced Institute of Natural Sciences, Beijing Normal University, Zhuhai 519087, China, and also with the Guangdong Key Laboratory of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai 519087, China (e-mail: jianxiongguo@bnu.edu.cn).

Digital Object Identifier 10.1109/TCSS.2023.3272331

performance. However, the exact value of influence spread is not available, and therefore, no accurate target value can be used for supervised learning. On the contrary, Li et al. [11] presented an end-to-end DRL model, called PIANO, by revising S2V-DQN [9]. PIANO is trained on subgraphs of the entire network and then tested on the entire network, which makes it not able to generalize on nonhomogeneous networks with different topological characteristics. To address the above drawbacks, we integrate the latest strategies and design a new solution framework for IM.

In this article, we model the IM problem as an RL problem, which aims to find the optimal policy of selecting b seeds (b action sequences) to maximize the influence spread (cumulative rewards) of these b seeds. However, the exact Q value in this RL is not available, and therefore deep Q-network [12] (DQN) is a natural solution to solve this issue. Instead of using DQN, we use its improvement double DQN (DDQN) [13], which can avoid the over-optimistic issue of a simple DQN and achieve better performance. On the other hand, except for the network topology structure, the function approximator in DDQN also needs to well capture the crucial influence cascading effects in IM, which makes it more challenging. The cascading effect represents that the activation of a node will trigger its neighbors in a successive manner, forming a diffusion cascade on social networks. This is consistent with the message-passing effect in graph neural networks (GNNs) [14]. Therefore, based on these two techniques, in this article, we propose a novel end-to-end DRL framework, called ToupleGDD (**T**hree **Couple**d **G**raph Neural Networks with **D**ouble **D**eep Q-networks), to solve the IM problem, which incorporates three coupled GNNs for network embedding and DDQN technique for parameter learning. The main contributions can be summarized as follows.

1) To the best of our knowledge, we are the first to present such an end-to-end framework, ToupleGDD, which combines coupled GNNs and the DRL method to effectively solve the IM problem.
2) We propose a personalized DeepWalk (PDW) method to learn initial node embedding as input features for the following customized GNN layer, which considers both local and global influence contexts of nodes.
3) To capture the crucial cascading effects of information diffusion and network topology, we design three coupled GNNs to learn node embeddings.
4) Extensive experiments are conducted on synthetic graphs and real-world datasets. Empirical results show that our model can achieve performance very close to IMM and even outperform OPIM-C on several datasets, which demonstrates the superiority and effectiveness of our proposed model.

Organization: Section II reviews the related works. Section III presents some preliminaries and frameworks of the ToupleGDD model. The two main parts of ToupleGDD: network embedding and RL formulation, are introduced in Section IV and V, respectively. Section VI is dedicated to experiments and results. Section VII concludes the article.

II. Related Works

A. Influence Maximization

Kempe et al. [1] first formulated IM as a CO problem and presented a $(1-1/e-\epsilon)$-approximation algorithm, Greedy, by applying Monte Carlo method to estimate the expected spread of a seed set. But it is too time-consuming. Borgs et al. [4] made a breakthrough for this issue with the RIS technique, which guaranteed $(1-1/e-\epsilon)$-approximation solutions and significantly reduced the expected running time. Subsequently, a series of more efficient randomized approximation algorithms were proposed, such as TIM/TIM+ [5], IMM [6], SSA/D-SSA [7], OPIM-C [8], and HIST [15]. They not only can provide $(1-1/e-\epsilon)$-approximation solution but also is efficient even on billion-size networks, which are state-of-the-art approximation algorithms for IM. Later, these algorithms are widely used to solve variations of IM, such as [16], [17].

B. ML/RL for CO

Recent advancements in deep learning and RL has resulted in an increasing number of works addressing IM by learning-based methods. Since IM can be formulated as a CO problem, many works aiming for CO problems have used IM as an example to test the performance of their models. Khalil et al. [9] first proposed a DRL model for CO problems, called S2V-DQN, which utilized the graph embedding method, structure2vec [18], to encode nodes states to formulate the value approximator, and the fit Q-learning to select the node to add to the current seed set. Li et al. [19] approximated the solution quality by graph convolutional networks and applied a learning framework based on guided tree search. Manchanda et al. [10] proposed a supervised deep-learning-based model, GCOMB, for CO problems over large graphs. By introducing a supervised learning step into the Q-learning framework, GCOMB can predict the quality of nodes and filter out "bad nodes" at an early step. Instead of solving CO problems on the entire graph, [20] and [21] are focused on how to prune the graph and discover a subgraph that can act as a surrogate to the entire graph. For readers interested in more works of CO, refer to [22], [23] and [24] for detailed reviews.

C. ML/RL for IM

Fan et al. [25] proposed the DRL model for network dismantling problem, FINDER, which aimed to find key players in complex networks, and applied GraphSAGE as the function approximator for DQN. Kamarthi et al. [20] utilized deep Q-learning for discovering the subgraph and solved the IM problem on the subgraph and utilized the selected influential node set as the seeds on the complete graph. There were some researches [26], [27], [28] focusing on using DRL to solve the competitive IM problem, which aims to find an optimal strategy against a competitor to maximize the commutative reward under the competition against other agents. Besides, [29], [30] considered the contingency-aware IM problem, where there is a probability of a node willing to be seeded when selected as a seed node. Tian et al. [31]

proposed a DIEM model for the topic-aware IM problem, which aims to maximize the activated number of nodes under the specific query topics. DIEM modified the structure2vec method [18] for network embeddings and utilized DDQN with prioritized experience replay to learn parameters. The work most related to ours is [11], which proposed a DRL model, called PIANO, for the IM problem, and presented with small modification from S2V-DQN [9].

D. Comparisons of Related Models to Our Model

FINDER model [25] was proposed for network dismantling problem and cannot work on directed graphs and weighted graphs. However, our model can work on undirected graphs and different edge weight settings. The GCOMB framework [10] was based on supervised learning which introduced large extra computational overhead and efforts of hand-crafting the learning pipeline, while our model can learn parameters end-to-end. PIANO method [11] applied structure2vec to learn node embeddings, while we designed three coupled GNNs to learn the network representation. Additionally, both GCOMB and PIANO are trained on subgraphs of the entire graph and tested on the rest of the entire network, which makes them graph-specific. However, our ToupleGDD model does not have this limitation and performs well on different training and testing datasets, which shows more generalization ability.

III. Preliminaries and Framework

A. Background

The social network is usually represented by a directed graph $G = (V, E)$, where V denotes the node (user) set and E is a set of relationships between nodes. For an edge $(u, v) \in E$, u is called the in-neighbor of v, and v is called the out-neighbor of u. For a node v, denote by $N_{\text{in}}(v)$ and $N_{\text{out}}(v)$ the in-neighbor set and out-neighbor set of v, respectively. There are many diffusion models to describe the information propagation process on the social network. Since the IC model will be used in our experiments, we will introduce it here.

Definition 1 (IC Model): Given $G = (V, E)$ with weight function $p : E \to [0, 1]$, where p_{uv} represents the propagation probability when u tries to activate v by edge (u, v). The IC model considers a timestamped propagation process: 1) each node has two possible states: active and inactive; 2) initially, all nodes in seed set S are activated and all other nodes are set inactive; 3) if a node u is first activated at timestamp t, then u will try to activate its inactive out-neighbor v at timestamp $t + 1$ with successful probability p_{uv}. After timestamp $t + 1$, u cannot activate any of its out-neighbors; and 4) once a node is activated, it remains active in the following timestamps.

The diffusion process will continue until there is no more node activated. Given a seed set S, denote by $I(S)$ the number of activated nodes when the diffusion process terminates. Let $\sigma(S)$ be the expected number of nodes that can be activated by S. That is, $\sigma(S) = \mathbb{E}[I(S)]$ and $\sigma(S)$ is called the influence spread of S.

Definition 2 (IM): Given a social network $G = (V, E)$, a positive integer b, and a diffusion model, IM aims to find a small set S of nodes as seeds with $|S| \le b$, which has the maximum influence spread.

Denote by $\sigma(v; S) = \sigma(S \cup \{v\}) - \sigma(S)$ the marginal gain obtained by adding v into a seed set S. Let S_t be the currently selected seed set. The greedy algorithm will select the node which can achieve the maximum of $\sigma(v; S_t)$ as the next seed. However, computing the influence spread of a seed set is #P-hard under the IC [2], resulting in the difficulty of calculating the marginal gain. Instead of generating a large number of RR sets such as in the state-of-art approximation algorithms, in this article, we regard IM as an RL problem, which aims to find an optimal policy to select k nodes or k action sequence with the maximum influence spread. In this case, the marginal gain can be considered as the value function in RL, whose value is difficult to be obtained in our problem. To address this issue, we approximate the value function (marginal gain) by a parameterized function through the DRL method.

Definition 3 (Learning-Based IM Problem): It can be divided into two phases.

1) *Learning Phase:* Given a set of training graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_c\}$, diffusion model ψ and influence spread function $\sigma : S \to \mathbb{R}^+$, train a group of parameters Θ such that $\hat{\sigma}(v, S; \Theta)$ could approximate $\sigma(v; S)$ as accurately as possible.
2) *Testing Phase:* Given a target social network G, the learned parameters Θ and an integer b, solve the IM problem with respect to budget b under some diffusion model ψ.

As a special type of RL, DRL applies deep neural networks for state representation and function approximation for value function, policy, transition model, or reward function. In this article, we use GNNs to obtain node embeddings and formulate the parameterized function using node embeddings, where all parameters are learned by DDQN.

B. General Framework of GNN

As an effective framework of node embedding learning, GNNs usually follow a neighbor-aggregation strategy, where the embedding of a node is updated by recursively aggregating embedding from its neighborhood. Formally, u's embedding at the $k+1$th layer $F_u^{(k+1)}$ is updated by the following equation:

$$\begin{aligned} m_{\mathcal{N}(u)}^{(k)} &= \text{AGGREGATE}^{(k)}\left(F_v^{(k)} : v \in \mathcal{N}(u)\right) \\ F_u^{(k+1)} &= \text{UPDATE}\left(F_u^{(k)}, m_{\mathcal{N}(u)}^{(k)}\right) \end{aligned}$$

where AGGREGATE and UPDATE are neural networks and $\mathcal{N}(u)$ is u's neighborhood.

C. Framework of ToupleGDD

In this section, we present the proposed framework ToupleGDD, which solves the IM problem by incorporating three coupled GNNs and DDQNs. The framework of ToupleGDD is illustrated in Fig. 1. Given a set of training graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_c\}$, we first apply the PDW method to get the initial node embedding, since it has been found that deep walk embedding rather than randomly initialized embedding is vital for stable training of Geometric-DQN, which also works well in our model and will be shown in experiments. Then GNN and attention mechanism are combined to learn node embeddings. Specifically, three coupled GNNs (ToupleGNN) are designed to capture the cascading effect of information diffusion. After K iterations of ToupleGNN, we use the obtained node embedding to construct the parameterized function $\hat{Q}(v, S; \Theta)$ and use the RL technique to learn the parameters. Instead of using DQN, we apply the DDQN to learn the parameters Θ for $\hat{Q}(v, S; \Theta)$ to approximate the marginal gain $\sigma(v; S)$ and adopt ε-greedy policy to select the next seed. The reason why we use DDQN will also be explained through experiments.

IV. Representation: Node Embedding

As a way of representing the node as a vector, node embeddings can capture the network topology. For our IM problem, more importantly, node embeddings need to capture the influence cascading effects, which represent that the activation of a node will successively trigger its out-neighbors, forming a diffusion cascade on networks. For a target node, whether it will be activated is intrinsically governed by three components: the states of in-neighbors, the influence capacity of in-neighbors, and its tendency to be influenced by in-neighbors. In this sense, the cascading effect is intrinsically the iterative interplay between node states, nodes' influence capacity, and nodes' tendency to be influenced by others. Therefore, for each node u, we include three parts in u's embedding: X_u, S_u, and T_u, where $X_u \in \mathbb{R}$ indicates the activation state of node u, $S_u \in \mathbb{R}^l$ is the capacity of u to influence other users, and $T_u \in \mathbb{R}^l$ is the tendency of being activated by other users.

A. Initial Embedding Learning

Instead of randomly generating initial embeddings, we proposed the PDW method to learn embeddings as input features for the following GNN layer. The main part of PDW is to generate node contexts and then utilize the skip-gram technique to predict contexts for a given node. Inspired by the Inf2vec model [32], for node $u \in V$, our method includes two parts as u's influence context C_u: local and global influence context, where local context is a sampled set of nodes that can be activated by u and global contexts are sampled from the r-hop out-neighbors of u. To limit the size of node contexts, assume the length threshold of the node context is L and $\alpha \in [0, 1]$. For a node u, we use random walk with restart (RWR) strategy (restart probability is set as 0.15 in this article) to obtain the local influence context L_u of node u, and the walk will stop when threshold $\alpha \cdot L$ is reached. After generating local contexts, we randomly sample $(1-\alpha) \cdot L$ nodes from the r-hop out-neighbor set N^r_{out} of u as global influence context G_u.

Given a user u, the probability of user v being influenced by user u is formulated as a softmax function by their node embeddings: $\Pr(v|u) = e^{X_u \cdot S_u \cdot T_v + X_v}/Z(u)$, where $Z(u) = \sum_{w \in V} e^{X_u \cdot S_u \cdot T_w + X_w}$ is the normalization term. Assume users in C_u are independent of each other, then the probability of observing context C_u conditioned on u's embedding is $\Pr(C_u|u) = \Pi_{v \in C_u} \Pr(v|u)$.

We will sample a set of influence contexts, $\mathcal{D} = \{(u_1, C_{u_1}), \ldots, (u_q, C_{u_q})\}$ from social network G. We consider all the observed influence contexts and attempt to maximize the log probability of them

$$\max \sum_{(u, C_u) \in \mathcal{D}} \sum_{v \in C_u} \log \Pr(v|u). \tag{1}$$

However, it is time-consuming to compute $Z(u)$ directly since we need to enumerate each $w \in V$. In this article, we utilize the negative sampling technique, which is popularly used to compute softmax functions. Instead of enumerating all nodes, the negative sampling method only considers a small set of sampled nodes. For each node $u \in V$, we randomly generate a small set of nodes N as negative instances to approximate the softmax function

$$\log \Pr(v|u) \approx \log \sigma(z_v) + \sum_{w \in N} \log \sigma(-z_w) \tag{2}$$

where $z_v = X_u \cdot S_u \cdot T_v + X_v$, $z_w = X_u \cdot S_u \cdot T_w + X_w$, and $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function.

The stochastic gradient descent (SGD) method is applied to learn all the parameters. In each step, we update the parameters Φ by calculating the gradient

$$\Phi \leftarrow \Phi + \eta \frac{\partial}{\partial \Phi} (\log \Pr(v|u)) \tag{3}$$

where η is the learning rate and $(\partial/\partial\Phi)$ represents the gradient of parameters Φ. Based on (2), the gradient for corresponding parameters can be computed as follows:

$$\begin{aligned}
\frac{\partial}{\partial S_u} &= (1 - \sigma(z_v)) \cdot X_u \cdot T_v + \sum_{w \in N} (-\sigma(z_w)) \cdot X_u \cdot T_w \\
\frac{\partial}{\partial T_v} &= (1 - \sigma(z_v)) \cdot X_u \cdot S_u, \quad \frac{\partial}{\partial T_w} = (-\sigma(z_w)) \cdot X_u \cdot S_u \\
\frac{\partial}{\partial X_u} &= (1 - \sigma(z_v)) \cdot S_u \cdot T_v + \sum_{w \in N} (-\sigma(z_w)) \cdot S_u \cdot T_w \\
\frac{\partial}{\partial X_v} &= 1 - \sigma(z_v), \quad \frac{\partial}{\partial X_w} = -\sigma(z_w).
\end{aligned} \tag{4}$$

The proposed PDW method is summarized in Algorithm 1. It contains two parts: influence context generation (lines 3–8) and parameters learning (lines 9–14), which have been illustrated above. In the influence context generation part, for each node u, local influence context is sampled by the RWR strategy, and we use breath first search method to obtain u's r-hop out-neighbor set $N^r_{\text{out}}(u)$ for generating global influence context (upper bounder by $|E|$). Therefore, the time complexity of the influence context generation part is $O(|V|(\alpha \cdot L + |E|)) = O(|V||E|)$. For the parameters learning part, for each tuple $(u, C_u) \in W$ (where $|W| = |V|$), L iterations are performed for nodes in C_u. At each iteration, we first update node embeddings of u and v and then update node embeddings for each node in the negative samples set N. Therefore, the running time of the parameters learning part is $O(|V| \cdot L \cdot |N|) = O(|V|)$. Here, we consider L and $|N|$ are fixed constants. Thus, the total time complexity of Algorithm 1 is $O(|V| + |V||E|) = O(|V||E|)$.

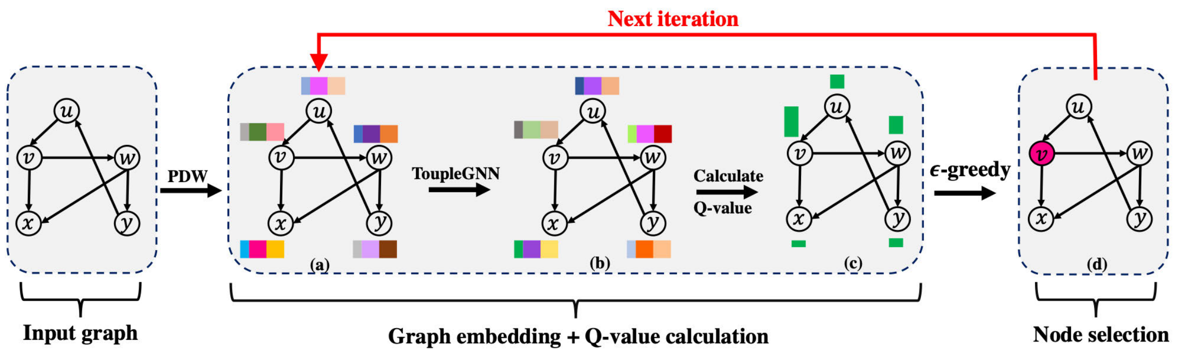


Fig. 1. Framework of ToupleGDD: (a) apply PDW to obtain initial embedding; (b) utilize ToupleGNN to capture network topology structures and influence cascading effects to get node embedding; (c) construct the parameterized function $\hat{Q}(v, S; \Theta)$ based on node embedding input from ToupleGNN; and (d) use ε-greedy to select the next seed and DDQN to learn the parameters.

Algorithm 1 PDW

```
1  Initialize $X_u, S_u, T_u$ by Gaussian distribution $\mathcal{N}(0, 0.01)$;
2  Initialize $W \leftarrow \emptyset$;
3  foreach $u \in V$ do
4      $L_u \leftarrow \emptyset, G_u \leftarrow \emptyset, C_u \leftarrow \emptyset$;
5      $L_u \leftarrow$ Sample $\alpha L$ nodes by RWR starting from $u$;
6      $G_u \leftarrow$ Uniformly sample $(1-\alpha)L$ nodes from $N^r_{out}(u)$;
7      $C_u \leftarrow L_u \cup G_u$;
8      Insert $(u, C_u)$ into $W$;
9  foreach $(u, C_u) \in W$ do
10     foreach $v \in C_u$ do
11         Update $X_u, S_u, X_v, T_v$;
12         Sample a set of negative samples $N$;
13         foreach $w \in N$ do
14             Update $X_u, S_u, X_w, T_w$;
15 return $X_u, S_u, T_u$ for each node $u$;
```

B. ToupleGNN

Inspired by [33], we design three coupled GNNs (ToupleGNN) to naturally capture the iterative interplay between node states, nodes' influence capacity, and nodes' tendency to be influenced by others. Taking initial node embeddings as input, ToupleGNN includes three coupled GNNs: 1) state GNN: model the activation states of nodes; 2) source GNN: model the influence capacity of nodes; and 3) target GNN: model the tendency of nodes to be influenced by others. The framework of these three GNNs is illustrated in the middle part of Fig. 2, and we will introduce detailed structures in the following part. Given the currently selected seed set S_t, we need to update node representations accordingly by ToupleGNN.

1) State GNN: The state GNN is used to model the activation state of each node during the cascading effect. For a target user v, it will be activated by its active in-neighbors. Therefore, its activation state X_v is determined by the activation states of its in-neighbors and the influence weight/probability of these in-neighbors to it. Since the interaction strength between users will change with nodes' states, only using the given static edge weight is not enough to capture the importance and influence weight between users. Therefore, except for the given edge weights, we also consider applying v's in-neighbors' capacity embedding and v's tendency embedding by an influence attention mechanism to dynamically capture the diffusion weight between them. Specifically, define $e^{(k)}_{uv} = \eta^{(k)}[W^{(k)}S^{(k)}_u, W^{(k)}T^{(k)}_v]$ to measure the dynamic importance of node u to v, where $\eta^{(k)} \in \mathbb{R}^{2h^{(k+1)}}$ is a weight vector, $W^{(k)} \in \mathbb{R}^{h^{(k+1)} \times h^{(k)}}$ is a weight matrix to transform the source and target representation from dimension $h^{(k)}$ to $h^{(k+1)}$, and $[\cdot, \cdot]$ denotes the concatenation of vectors. To make coefficients comparable among nodes, a softmax function incorporated with the LeakyReLU [34] is adopted to normalize the attention coefficients

$$\text{InfluGate}\big(S^{(k)}_u, T^{(k)}_v\big) = \frac{\exp\big(\text{LeakyReLU}\big(e^{(k)}_{uv}\big)\big)}{\sum_{u \in N_{\text{in}}(v)} \exp\big(\text{LeakyReLU}\big(e^{(k)}_{uv}\big)\big)} \tag{5}$$

where LeakyReLU has a negative slope of 0.2.

The expected influence that node v aggregates from its in-neighbors is

$$a^{(k)}_v = \sum_{u \in N_{\text{in}}(v)} \Big(\delta^{(k)}_1 p_{uv} + \delta^{(k)}_2 \text{InfluGate}\big(S^{(k)}_u, T^{(k)}_v\big)\Big) \cdot X^{(k)}_u. \tag{6}$$

Since we expect that the activation state should indicate the possibility of a node being activated, the activation state of node v is set to 1 when it is selected into the current seed set S_t. Otherwise, v's activation state is updated by aggregating influence from its in-neighbors. That is, node v's activation state at $(k+1)$th layer is updated by the following equation:

$$X^{(k+1)}_v = \begin{cases} 1, & \text{if } v \in S_t \\ \sigma\big(\xi^{(k)}_X X^{(k)}_v + \xi^{(k)}_a a^{(k)}_v\big), & \text{otherwise} \end{cases} \tag{7}$$

where $\xi^{(k)}_X, \xi^{(k)}_a \in \mathbb{R}$ are weight parameters and $\sigma(\cdot)$ is the sigmoid function.

2) Source GNN: The source GNN is used to model the capacity of nodes to influence others. Intuitively, the capacity of a node v to activate others can be measured by both its activation state and how much influence its out-neighbors can

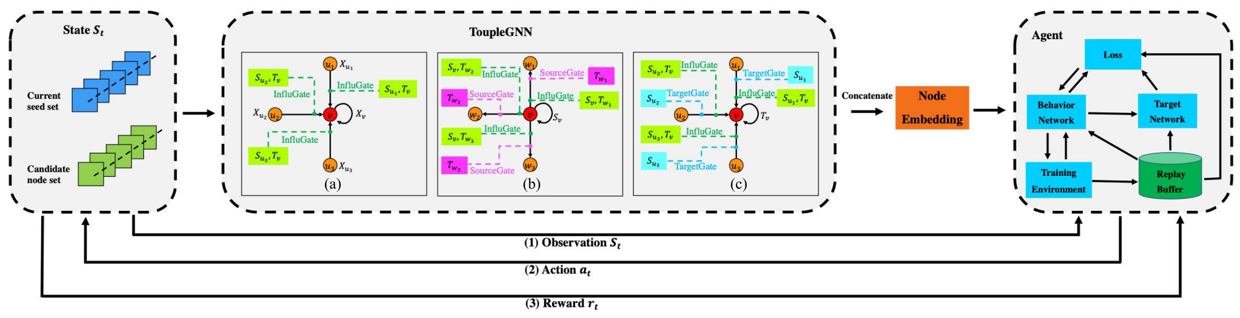


Fig. 2. Mechanism of DDQN incorporated ToupleGNN as a function approximator.

get when the information is spread from v to out-neighbors, which can be modeled by v's out-neighbors' tendency to be activated. Similar to the dynamic influence weight defined in state GNN, for edge $(v, w) \in E$, we also define the dynamic attention weight $f_{vw}^{(k)} = \beta^{(k)}[W^{(k)} S_v^{(k)}, W^{(k)} T_w^{(k)}]$ and its corresponding normalization for the weighted aggregation

$$\alpha_{vw}^{(k)} = \frac{\exp\left(\text{LeakyReLU}\left(f_{vw}^{(k)}\right)\right)}{\sum_{w \in N_{\text{out}}(v)} \exp\left(\text{LeakyReLU}\left(f_{vw}^{(k)}\right)\right)} \tag{8}$$

where $\beta^{(k)} \in \mathbb{R}^{2h^{(k+1)}}$ is a weight vector. Then the neighborhood aggregation is defined as follows:

$$b_v^{(k)} = \sum_{w \in N_{\text{out}}(v)} \left(\lambda_1^{(k)} p_{vw} + \lambda_2^{(k)} \alpha_{vw}^{(k)}\right) \cdot \text{SourceGate}\left(T_w^{(k)}\right) \tag{9}$$

where SourceGate($\star$) is the source gating mechanism implemented by a three-layer MLP in this article to reflect the nonlinear effect of out-neighbors' target tendency.

The source representation of node v at the $(k+1)$th layer is updated by incorporating its kth layer source representation, neighborhood aggregation, and its activation state

$$S_v^{(k+1)} = \sigma\left(\gamma_S^{(k)} S_v^{(k)} + \gamma_b^{(k)} b_v^{(k)} + \gamma_X^{(k)} X_v^{(k)}\right) \tag{10}$$

where $\gamma_S^{(k)}, \gamma_b^{(k)}, \gamma_X^{(k)} \in \mathbb{R}$ are weight parameters.

3) Target GNN: The target GNN is used to model the nodes' tendency to be influenced by others. Generally, the tendency of a node to be activated is determined by its current activation state and the influence diffusion from its in-neighbors to it. Similarly, for edge $(u, v) \in E$, define $d_{uv}^{(k)} = \tau^{(k)}[W^{(k)} S_u^{(k)}, W^{(k)} T_v^{(k)}]$ and

$$\phi_{uv}^{(k)} = \frac{\exp\left(\text{LeakyReLU}\left(d_{uv}^{(k)}\right)\right)}{\sum_{u \in N_{in}(v)} \exp\left(\text{LeakyReLU}\left(d_{uv}^{(k)}\right)\right)} \tag{11}$$

where $\tau^{(k)} \in \mathbb{R}^{2h^{(k+1)}}$ is a weight vector. Then the neighborhood aggregation is defined as follows:

$$c_v^{(k)} = \sum_{u \in N_{in}(v)} \left(\rho_1^{(k)} p_{uv} + \rho_2^{(k)} \phi_{uv}^{(k)}\right) \cdot \text{TargetGate}\left(S_u^{(k)}\right) \tag{12}$$

where TargetGate($\star$) is the target gating mechanism implemented by a three-layer MLP in this article to reflect the nonlinear effect of in-neighbors' source ability.

The target representation of node v at the $(k+1)$th layer is updated by incorporating its kth layer target representation, neighborhood aggregation, and its activation state

$$T_v^{(k+1)} = \sigma\left(\mu_S^{(k)} T_v^{(k)} + \mu_c^{(k)} c_v^{(k)} + \mu_X^{(k)} X_v^{(k)}\right) \tag{13}$$

where $\mu_S^{(k)}, \mu_c^{(k)}, \mu_X^{(k)} \in \mathbb{R}$ are weight parameters.

C. Putting it Together

At each iteration of ToupleGNN, information diffusion and network structure features can be passed across nodes. After K iterations, node embedding can aggregate information from its K-hop neighbors. For node u, denote by $X_u^{(K)}, S_u^{(K)}, T_u^{(K)}$ the three components of u's node embedding after K iterations. Then u's node embedding can be obtained by concatenating these three parts: $[X_u^{(K)}, S_u^{(K)}, T_u^{(K)}]$. For the kth layer of state GNN, the time complexity is $O(|V| + |E|)$, which is the same for source GNN and target GNN. Therefore, the overall time complexity of ToupleGNN is $O(K(|V| + |E|))$.

Based on the obtained node embeddings, the score function to measure the marginal gain of a node $u \in \bar{S}_t = V \setminus S_t$ with respect to the current seed set S_t is defined as $\hat{Q}(u, S_t; \Theta) =$

$$\theta_1^\top \text{ReLU}\left(\left[\theta_2 S_u^{(K)}, \theta_3 \sum_{v \in S_t} S_v^{(K)}, \theta_4 \sum_{w \in V \setminus (S_t \cup \{u\})} T_w^{(K)}\right]\right) \tag{14}$$

where $\theta_1 \in \mathbb{R}^{3l}, \theta_2, \theta_3, \theta_4 \in \mathbb{R}^{l \times l}$ are model parameters. Since the embeddings used to define $\hat{Q}(u, S_t; \Theta)$ are computed based on the parameters from ToupleGNN, $\hat{Q}(u, S_t; \Theta)$ will depend on $\{\theta_i\}_{i=1}^4$ and all parameters in ToupleGNN. We will train these parameters (denoted by Θ) end-to-end by RL.

V. Reinforcement Learning

A. RL Formulation

RL concerns about how intelligent agents can take actions according to the current state when interacting with the environment to maximize the total reward received. Why do we use the RL model to learn the parameters in $\hat{Q}(u, S_t; \Theta)$? Actually, the IM problem can be naturally formulated as an RL problem:

1) *Action:* An action selects a node $u \in \bar{S}_t$ as the next seed, and we use u's node embedding to represent the action.
2) *State:* A state $\mathcal{S}_t$ represents a sequence of actions of selecting nodes in the current seed set S_t. We use a $|V|$-dimensional vector to represent state $\mathcal{S}_t$, where the corresponding component of node u is 1 if $u \in S_t$, and 0 otherwise. For simplicity, we will use S_t instead of $\mathcal{S}_t$ to represent the state when there is no ambiguity. The terminal state S_b is the state after selecting b nodes.
3) *Transition:* Changing the activation state X_u from 0 to 1 when $u \in \bar{S}_t$ is selected as the seed.
4) *Reward:* The reward $r(S_t, u)$ at state S_t is defined as the change of reward after selecting node u into the

current seed set S_t and transition to a new state. That is, $r(S_t, u) = \sigma(S_t \cup \{u\}) - \sigma(S_t)$ and $r(\emptyset) = 0$. In this way, the cumulative reward R of a terminal state S_b coincides exactly with the influence spread of seed set S_b, that is, $R = \sum_{t=0}^{b-1} r(S_t, u_t) = \sigma(S_b)$.

5) *Policy:* Policy maps a state to possibilities of selecting each possible action. That is, a policy tells the agent how to pick the next action.

If we denote by Q^* the optimal Q-function for this RL problem, then our embedding parameterized function $\hat{Q}(u, S_t; \Theta)$ will be a function approximator for it, which will be learned by DDQN.

B. Training via DDQN

We use DDQN [13] to perform end-to-end learning of parameters in $\hat{Q}(u_t, S_t; \Theta)$, which can avoid the over-optimistic issue of a simple DQN by adopting two networks: behavior network and target network, parameterized with Θ and Θ', respectively. The target network provides Q-values estimation of future states during training of the behavior network, and only updates parameters Θ' from the behavior network Θ every m episodes. The detailed training process is illustrated in Algorithm 2. We use the term *episode* to represent a complete sequence of node additions starting from an empty set until termination, and a single action (node addition) within an episode is referred to as a *step*. To collect a more accurate estimate of future rewards, n-step Q-learning [35] is utilized to update the parameters, which is to wait for n steps before updating parameters. Additionally, we apply the fit Q-iteration [36] with experience replay for faster learning convergence. Formally, the update is performed by minimizing the following square loss:

$$(y - \hat{Q}(u_t, S_t; \Theta))^2 \tag{15}$$

where $y = \sum_{i=0}^{n-1} \gamma^i r(S_{t+i}, u_{t+i}) + \gamma^n \max_v \hat{Q}(v, S_{t+n}; \Theta')$, and $\gamma \in [0, 1]$ is the discount rate, determining the importance of future rewards.

Specifically, we first apply the PDW method (Algorithm 1) to obtain initial embeddings. Then for each episode (lines 2–20), the seed set is initialized to an empty set. For each step, ε-greedy policy is utilized to select a node, which selects a node randomly with probability ε and with $(1-\varepsilon)$ probability selects the node with the maximum Q value (lines 5–14). If $t \geq n$, it will add the current sample $(S_{t-n}, u_{t-n}, \sum_{i=0}^{n-1} \gamma^i r(S_{t-n+i}, u_{t-n+i}), S_t)$ to the replay buffer M. Instead of performing a gradient step with respect to the loss of the current example, the parameters are updated with a batch of random samples from the buffer (lines 24–25). For each episode, we will perform b steps. At each step, node embeddings for each node will be updated for K times by ToupleGNN. At each layer of ToupleGNN, each node aggregates information from its in/out-neighborhood (overall $O(|E|)$). Therefore, the time complexity of each layer is $O(|V| + |E|)$. Putting it all together, the time complexity of Algorithm 2 is $O(|V||E| + DbK(|V| + |E|))$.

Algorithm 2 Training of ToupleGDD

```
1  Obtain initial embedding for each $u \in V$ by Alg. 1;
2  for episode $e = 1$ to $D$ do
3      $S_0 = \emptyset$;
4      for $t = 1$ to $b$ do
5          Uniformly sample a number $c$ from [0, 1);
6          if $c < \varepsilon$ then
7              Randomly select a node $u_t \in V \setminus S_t$;
8          else
9              for $i = 1$ to $K$ do
10                 for $u \in V$ do
11                     Update $X_u^{(i)}, S_u^{(i)}, T_u^{(i)}$ by
                       ToupleGNN;
12             for $u \in V$ do
13                 Calculate $\hat{Q}(u, S_t; \Theta)$ by (14);
14             Select $u_t = \arg\max_{u \in \bar{S}_t} \hat{Q}(u, S_t; \Theta)$;
15         $S_t = S_{t-1} \cup \{u_t\}$;
16         if $t \geq n$ then
17             $(S_{t-n}, u_{t-n}, \sum_{i=0}^{n-1} \gamma^i r(S_{t-n+i}, u_{t-n+i}), S_t)$ to
               replay buffer $M$;
18     Sample random batch $B \sim M$;
19     Update $\Theta$ by Adam optimizer over (15) with $B$;
20     Update $\Theta'$ from $\Theta$ every $m$ episodes;
21 return $\Theta$;
```

VI. Experiments

In this section, we conduct several experiments on different datasets to validate the performance of our proposed ToupledGDD model. All experiments are conducted on a machine with an Intel Xeon CPU (2.40 GHz, 28 cores), 512 GB of DDR4 RAM, Nvidia Tesla V100 with 16-GB HBM2 memory, and running CentOS Linux 7. The source code is available at https://github.com/Dtrycode/ToupleGDD.

A. Experimental Setup

1) Datasets: To thoroughly evaluate the performance of the proposed model, both synthetic and real-world datasets are used for evaluation. We generate 20 random Erdős-Renyi (ER) graphs with node sizes varying from 15 to 50 for training and validation. Specifically, we first sample the number of nodes uniformly at random from 15 to 50 and then generate an ER graph with an edge probability of 0.15. Among those generated synthetic graphs, 15 graphs are used for training, and the others are used for validation with the soc-dolphins dataset [37]. The performance of the proposed model and baselines are tested on seven real-world datasets, whose detailed statistics are shown in Table I. For the undirected graph, we replace each edge with two reversed directed edges. Among these datasets, Twitter, Wiki-1, caGr, and Buzznet are from [37], while Wiki-2, Epinions, and Youtube are available on [38].

2) Diffusion Models: Our model can be easily adapted to distinct diffusion models by revising the definition of the reward function. In this article, we report the results under

TABLE I
DATASET CHARACTERISTICS

Dataset	*n*	*m*	Type	Average degree
soc-dolphins	62	159	directed	5
Twitter	0.8k	1k	directed	2
Wiki-1	0.9k	3k	directed	6
caGr	4.2k	13.4k	undirected	5
Wiki-2	7.1k	103.7k	directed	29
Epinions	76k	509k	directed	13
Buzznet	101k	3M	directed	55
Youtube	1.13M	3M	undirected	5

the IC model here. Unless otherwise specified, the probability on edge (u, v) is set to $1/N_{\text{in}}(v)$ (in-degree setting), which is widely used in previous works about IM [5], [6], [7]. To fairly evaluate the performance of different methods, we first record the seed set obtained by different methods independently and then perform 10 000 Monte Carlo simulations to estimate the expected influence spread. All experiments are run ten times and we report the average of the metric being measured.

3) Baselines: We compare the performance of ToupleGDD with the state-of-the-art approximation algorithm for the IM problem, IMM [6], and OPIM-C [8] and the DRL methods S2V-DQN [9] and GCOMB [10] for the CO problem. Note that S2V-DQN is originally designed for the CO problem, and we revised their code for the maximum cut (MC) problem to solve IM. Another baseline is PIANO [11], which is modified from the S2V-DQN model for IM. For all other baselines, we use the code shared by the authors. For IMM and OPIM-C, we set $\epsilon = 0.1$.

4) Training and Testing Details: For all training datasets, edge weights are set as in-degree settings. Edge weights on validation datasets and testing datasets have the same setting (we will only specify the set of testing datasets in the following) and may be set as one of the three settings: 1) in-degree setting; 2) set as 0.1 (0.1-setting); and 3) set as 0.5 (0.5-setting). We set the budget b as 5 for all training datasets, while in validation setting 5 and 7 for ER graphs and soc-dolphins dataset, respectively. For each testing dataset, we vary budget b such that $b \in \{10, 20, 30, 40, 50\}$. For S2V-DQN and ToupleGDD, we use the RIS method to estimate the influence spread for a given seed set in the training phase. For GCOMB, since their code is not able to deal with multiple training graphs, we follow the same instructions as in their paper and use the training graph shared by them by revising the edge weight to the in-degree setting.

B. Experimental Results

1) Ablation Study: In the early version (called DISCO [39]) of the PIANO model, they have shown that the order of candidate 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, instead of iteratively selecting and recomputing node embeddings and Q values according to each seed insertion (iterative operation), they simplified the process into only one iteration, by embedding only once and selecting the top-b nodes with the maximum Q (one-time operation). Inspired by this conclusion and operation, we compare the expected influence spread of seed sets obtained by our ToupleGDD model by these two operations. On the other hand, we also test the impact of the initial embedding on our model. Three groups of experiments are conducted: 1) both train and test with initial embedding (TIEI); 2) train with initial embedding but test without initial embedding (TIEN); 3) both train and test without initial embedding (TNEN). For all of these three types, the validation setting is the same as the testing, and all experiments of this part use an in-degree probability setting. Besides, for (1) and (2), they share the same training model, and validations are conducted independently for them. For each of the three groups, the iterative and one-time operations are performed in the same experiment. That is, after computing the Q-values, we first output the top-b nodes with the highest Q-values and then perform the iterative operation according to the greedy strategy. Therefore, both of these two operations share the same initial embeddings if there are any.

The results are shown in Table II. Note that TI and EI represent training and testing with initial embedding, respectively. First, for the same dataset and seeds selection operation (e.g., Twitter with iterative operation), comparing the results of three groups, we see that the expected influence spread of the seed set obtained by TIEI and TIEN are very close. However, the results of TNEN have big differences from the other two under the same budget and are not stable under different datasets, which indicates the necessity and importance of initial embedding in training. Second, the running time of TIEN and TNEN is less than that of TIEI for the same dataset and seeds selection operation, and this difference is significantly big for large datasets, such as Epinions. This is because there is no initial embedding generation in TIEN and TNEN when testing, which can save much time, especially for large datasets. Besides, the running time of iterative operation increases with the increase of budget due to more iterations and selections, and for one-time operations, there is no significant difference between different budgets. Third, for the same group of the experiment (e.g., Twitter under TIEI), comparing the expected spread obtained by iterative and one-time operations, we observe the difference between them is very small, but they actually do not share the exact same seed set in most cases. However, we cannot figure out the reason causing this difference due to the machine accuracy configuration for very close values. Besides, one-time operation can output seed set faster than iterative selection, due to its less iteration and computation. From these results, it is convincing that we can use a one-time operation for seed selection and TIEN setting to save time but without a large decrease in influence spread. For those who want to apply this algorithm to their problems, it is determined by the tradeoff between accuracy and running time.

2) Influence Spread: We test the performance of ToupleGDD and baselines on Wiki-1, Epinions, caGr, Buzznet, and Youtube datasets with the in-degree probability setting. Fig. 3 draws the expected influence spread and running time produced by different models on these five datasets. Note that the results obtained by our model are from the TIEN setting and one-time operation, which could not only provide close

TABLE II
PERFORMANCE OF TOUPLEGDD UNDER DIFFERENT SETTING

Dataset	Operation	TI	EI	**budget: 10** Influence spread (time:s)	**budget: 20** Influence spread (time:s)	**budget: 30** Influence spread (time:s)	**budget: 40** Influence spread (time:s)	**budget: 50** Influence spread (time:s)
Twitter	iterative	✓	✓	147.71 (9.17)	210.86 (16.41)	252.11 (23.42)	287.59 (29.84)	315.88 (38.33)
		✓		146.93 (7.03)	210.66 (13.06)	252.12 (21.13)	287.95 (29.82)	315.99 (33.55)
				139.36 (5.69)	188.23 (13.80)	234.41 (19.56)	270.09 (27.22)	301.59 (33.03)
	one-time	✓	✓	146.87 (3.01)	210.86 (3.0)	252.11 (3.08)	288.08 (2.97)	316.60 (3.13)
		✓		147.26 (0.67)	210.66 (0.58)	251.53 (0.64)	287.95 (0.62)	317.28 (0.63)
				139.36 (0.57)	188.29 (0.66)	232.85 (0.63)	270.11 (0.72)	300.34 (0.73)
caGr	iterative	✓	✓	213.13 (18.40)	368.74 (25.48)	489.15 (31.60)	602.95 (38.16)	696.95 (46.53)
		✓		214.18 (6.09)	372.13 (12.24)	488.60 (18.15)	602.33 (27.53)	697.99 (37.62)
				210.56 (5.78)	368.28 (12.89)	489.33 (19.15)	605.18 (27.80)	699.82 (36.81)
	one-time	✓	✓	208.29 (12.09)	355.87 (11.99)	487.99 (11.65)	604.93 (11.77)	695.95 (11.77)
		✓		210.10 (0.62)	372.79 (0.61)	488.27 (0.57)	608.25 (0.68)	704.79 (0.65)
				208.24 (0.57)	367.52 (0.63)	487.69 (0.63)	608.23 (0.66)	708.54 (0.67)
Wiki-2	iterative	✓	✓	290.48 (46.79)	423.96 (54.68)	521.79 (63.35)	601.39 (71.72)	669.43 (78.23)
		✓		290.81 (7.39)	424.77 (15.79)	523.38 (22.82)	600.29 (31.01)	670.27 (40.04)
				285.10 (7.24)	422.81 (15.4)	510.53 (19.99)	585.87 (30.18)	647.11 (36.36)
	one-time	✓	✓	288.97 (39.94)	421.39 (40.46)	518.63 (39.05)	599.93 (39.57)	668.39 (39.16)
		✓		290.22 (0.72)	424.56 (0.70)	516.09 (0.70)	599.04 (0.70)	666.61 (0.73)
				282.42 (0.67)	420.99 (0.68)	504.32 (0.66)	579.26 (0.72)	642.27 (0.69)
Epinions	iterative	✓	✓	6022.85 (2.77×10^3)	8303.34 (2.75×10^3)	9693.69 (2.81×10^3)	10866.88 (2.79×10^3)	11781.69 (2.8×10^3)
		✓		6018.67 (13.4)	8295.06 (29.9)	9708.72 (44.96)	10853.63 (58.2)	11771.37 (75.61)
				6013.50 (13.51)	8300.42 (29.11)	9700.72 (42.07)	10840.46 (57.35)	11795.48 (70.3)
	one-time	✓	✓	6022.85 (2.76×10^3)	8300.36 (2.72×10^3)	9694.49 (2.76×10^3)	10832.49 (2.73×10^3)	11736.67 (2.73×10^3)
		✓		6018.67 (1.36)	8315.56 (1.46)	9718.94 (1.42)	10829.7 (1.4)	11758.36 (1.46)
				6013.5 (1.38)	8310.07 (1.42)	9729.69 (1.42)	10864.27 (1.43)	11800.19 (1.34)

influence spread with the corresponding iterative operation, but also runs in less time. From the left column of Fig. 3, the expected influence spread increases with the increase of budget, which is consistent with the monotone increasing characteristic of influence spread under the IC model. Besides, the performance of ToupleGDD is very close to IMM and outperforms OPIM-C on Wiki-2, Buzznet, and Youtube datasets, which proves the effectiveness of our model. Comparing the performance of all DRL-based models, ToupleGDD can outperform all other DRL-based models on all tested datasets, demonstrating the superiority of our model. And PIANO and S2V-DQN do not perform stably across different datasets, where S2V-DQN performs better than PIANO on undirected graph caGr, but worse than PIANO on other datasets. This may be because S2V-DQN is designed for undirected graphs, and the original paper trained and tested the model on undirected graphs. Even though PIANO is revised from S2V-DQN, its performance is not close to S2V-DQN. This may be because the code of PIANO is revised from the code for the minimum vertex cover (MVC) problem in the shared S2V-DQN code, while our revised code is from the MC problem. Thus, we use different initial node features. The reason that we choose the code of MC is that they have considered edge weight and edge features in MC but not in MVC. Additionally, PIANO and GCOMB have close performance on Epinions, caGr, and Buzznet datasets.

3) Running Time: The right column of Fig. 3 draws the corresponding running time of different models to obtain the results in the left column. Note that we only record the time that the model needs to output the seed set for a budget not including the time to compute the influence spread of the seed set. We observe that S2V-DQN needs more time to output the seed set than ToupleGDD on all datasets except Youtube. This may be because S2V-DQN uses an iterative not one-time manner to select seeds, which needs to update embedding and recompute Q-values for b times. Among all the tested methods, OPIM-C needs the least time and the ToupleGDD model runs a little slower. This may be because our model has many parameters and need to compute the dynamic influence importance between nodes which is time-consuming. But our model's running time is acceptable since it is less than 3 s even for the million-size dataset Buzznet. Note that this time difference also includes the effects of different implementation languages, since ToupleGDD is implemented by Python, while IMM, OPIM-C, and most parts of PIANO are implemented by C++. We also observe that GCOMB runs slower than ToupleGDD and PIANO on Wiki-2, Epinions, and caGr datasets. This may be because GCOMB is proposed for CO problems over very large networks and in their paper, they claimed GCOMB is hundreds of times faster than IMM on million-size datasets. However, from the results in Fig. 3, for small graphs Wiki-2, caGr, and Epinions, the running time of GCOMB is longer than ToupleGDD and PIANO due to its extra computational overhead and efforts of hand-crafting the learning pipeline in the supervised learning part.

4) Generalization: To further validate ToupleGDD's generalization ability, we test the performance of the model trained under an in-degree setting and tested on Twitter and Wiki-1

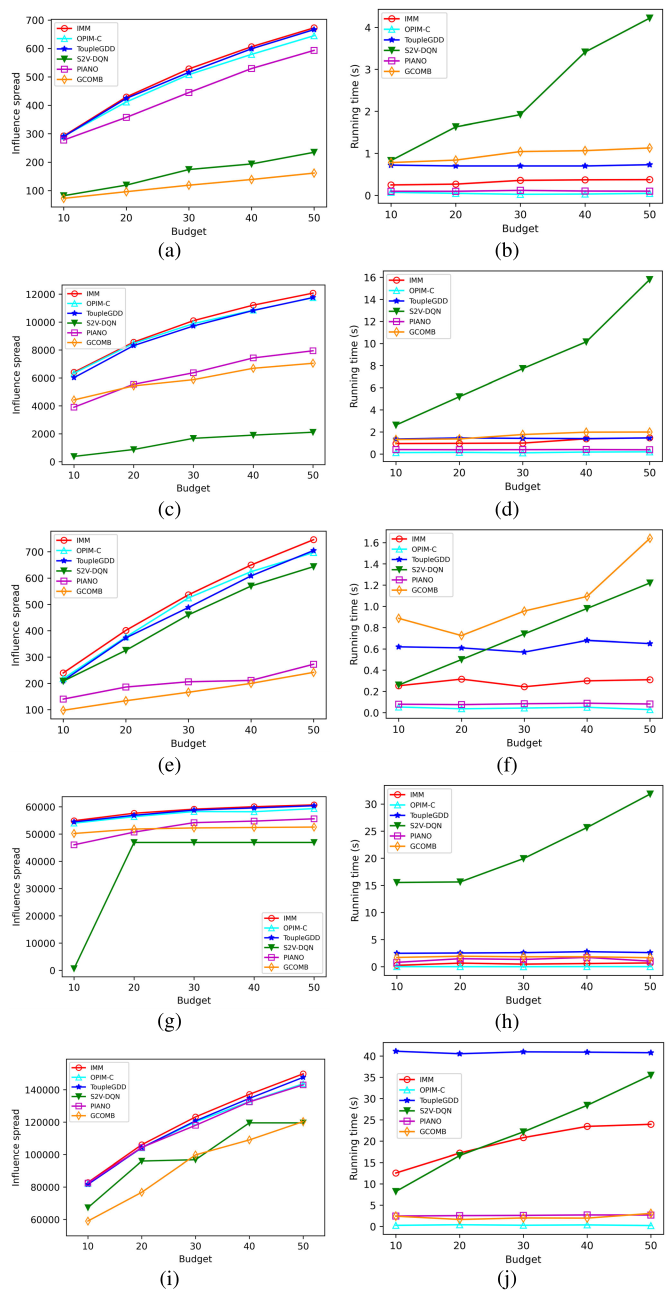


Fig. 3. Performance and running time comparisons among different methods. (a) Wiki-2, performance. (b) Wiki-2, running time. (c) Epinions, performance. (d) Epinions, running time. (e) caGr, performance. (f) caGr, running time. (g) Buzznet, performance. (h) Buzznet, running time. (i) Youtube, performance. (j) Youtube, running time.

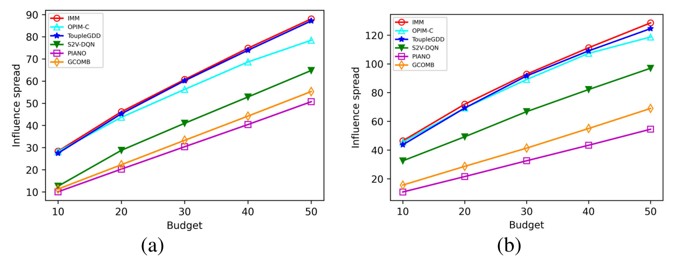


Fig. 4. Performance comparisons among different methods under 0.1-setting. (a) Twitter. (b) Wiki-1.

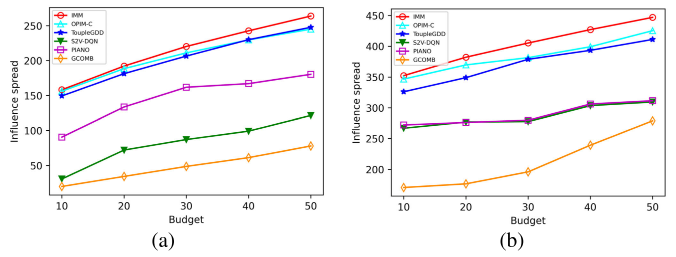


Fig. 5. Performance comparisons among different methods under 0.5-setting. (a) Twitter. (b) Wiki-1.

datasets with both 0.1- and 0.5-setting. Figs. 4 and 5 draw the results for 0.1- and 0.5-setting, respectively. From these results, the performance of ToupleGDD is almost equal to IMM and outperforms OPIM-C under 0.1-setting even though it is trained under an in-degree setting. And ToupleGDD outperforms all other DRL-based models for both of the two edge probability settings on the two tested datasets. This demonstrates the robustness and generalization ability of the proposed ToupleGDD model. The performance of S2V-DQN, PIANO, and GCOMB are not stable across different edge weight settings. S2V-DQN outperforms PIANO and GCOMB under 0.1-setting, while PIANO outperforms S2V-DQN and GCOMB under 0.5-setting. Furthermore, our model can obtain at least a 33% gain of the expected influence spread over S2V-DQN under 0.1-setting, and at least 20% gain of the expected influence spread over PIANO under 0.5-setting.

C. Intuition of Applying DDQN

Although DQN is an important milestone for deep learning, several limitations of this algorithm are now known. The improvement to DQN has blossomed in the last decades, such as dueling DQN (DuelDQN), double DQN (DDQN), and duel double DQN (DuelDDQN). How about the performance of S2V-DQN by replacing DQN with these improvements on IM? This is the main purpose of this group of experiments. We incorporate the structure2vec method with these improved models and compare their performance with S2V-DQN on IM.

We train the four models on the soc-dolphins dataset with 0.5-setting for edge weight and the budget is taken from $\{5, 7, 9\}$. Here, we keep the budget the same for training and testing to avoid the effect of changing the budget in the performance. For each budget, we train each framework 1000 epochs with exploration ratio ε starting from 1 and multiplied by a factor per epoch to balance exploration and exploitation. We run each framework five times to get the average and standard deviation.

The learning curves of the four frameworks with budgets 5, 7, and 9 are shown in Fig. 6(a)–(c), respectively. We expect S2V-DuelDQN to converge fast and S2V-DuelDDQN to perform the best. However, from the results, we observe that S2V-DuelDQN may not work and its advantage of fast convergence is not perceivable. The S2V-DDQN does perform well and DuelDDQN manages to make its influence score increase more in fewer epochs. The learning curves fluctuate more with the simple DQN- and DuelDQN-based models, while the DDQN-based models maintain much more stable learning curves across multiple runs.

We test the trained frameworks on a uniformly sampled graph with the same number of nodes and edges as in the

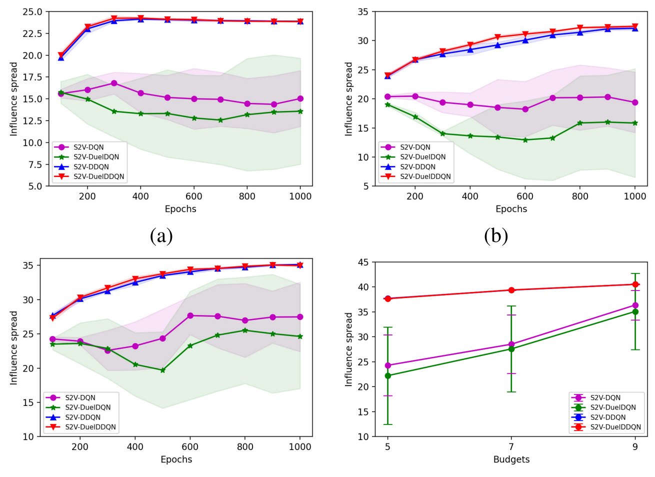


Fig. 6. Training and testing results for different models. (a)–(c) Learning curve with budget 5, 7, and 9, respectively. The solid line is average, and the shadow is one standard deviation. (d) Testing result: dot is average, and the bar shows one standard deviation.

TABLE III

p-Value Under Different Budgets (S2V is Saved in Methods Name for Space)

Model	Budget	DQN	DuelDQN	DDQN	DuelDDQN
DQN	5	-	0.3799	2.1029e-14	2.3740e-14
	7		0.6611	5.7903e-12	5.3526e-12
	9		0.4489	1.1655e-08	1.2368e-08
DuelDQN	5	-	-	4.4682e-10	4.8612e-10
	7			2.0871e-08	1.9727e-08
	9			0.0011	0.0011
DDQN	5	-	-	-	0.0082
	7				0.0457
	9				0.4111

training dataset. For each model from training, we run five times on the testing graph to get its average performance and the seeds are selected with iterative operation. Fig. 6(d) draws the expected spread of the seed set obtained by different models. We observe that the DDQN-based models still perform better than the simple DQN and DuelDQN models and are much more stable. Furthermore, we use p-value to check the significance of testing performance differences between frameworks as shown in Table III. Generally, when the p-value is less than 0.05, the performance difference between the two models is significant. The p-value results agree with our previous observation that DDQN-based models perform significantly better than DQN- and DuelDQN-based models. Though the difference between S2V-DDQN and S2V-DuelDDQN is subtle in Fig. 6(d), DuelDDQN does get significantly better performance when the budget is small.

Fig. 7(a) and (b) draws the training and testing time averaged from five runs for each framework. The time usage is approximately proportional to the budget size. DDQN-based models even maintain much lower time usage (both training and testing) compared to DQN-based models, which demonstrates the efficiency of DDQN-based models.

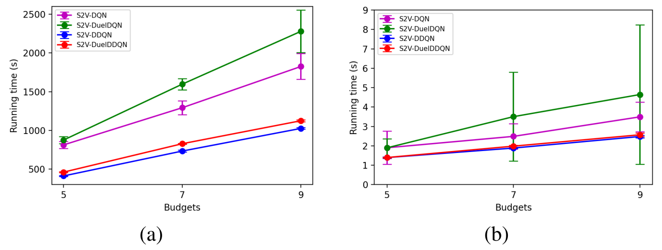


Fig. 7. Running time: dot is average, and bar shows one standard deviation. (a) Training time. (b) Testing time.

VII. Conclusion

In this article, we present a novel end-to-end framework, ToupleGDD, to address the IM problem by leveraging the DRL technique. Specifically, we incorporate GNNs for network embedding and RL technique, double DQNs, for parameter learning. Compared to the state-of-the-art sampling-based approximation algorithms, ToupleGDD can avoid costly sampling of the diffusion paths. Compared to previous works using the DRL method for the IM problem, our model has a stronger generalization ability and shows almost consistent performance across different social networks. We conduct extensive experiments to evaluate the performance of our proposed model. The empirical results show that ToupleGDD can achieve almost equal expected spread to that of IMM and outperform OPIM-C on several datasets, which is much better than other learning-based methods. This validates the effectiveness and efficiency of the proposed ToupleGDD model.

References

[1] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2003, pp. 137–146.

[2] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2010, pp. 1029–1038.

[3] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2010, pp. 88–97.

[4] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2014, pp. 946–957.

[5] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2014, pp. 75–86.

[6] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A Martingale approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2015, pp. 1539–1554.

[7] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *Proc. Int. Conf. Manage. Data*, Jun. 2016, pp. 695–710.

[8] J. Tang, X. Tang, X. Xiao, and J. Yuan, "Online processing algorithms for influence maximization," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 991–1005.

[9] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[10] 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 *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 20000–20011.

[11] H. Li, M. Xu, S. S. Bhowmick, J. S. Rayhan, C. Sun, and J. Cui, "PIANO: Influence maximization meets deep reinforcement learning," *IEEE Trans. Computat. Social Syst.*, early access, May 5, 2022, doi: 10.1109/TCSS.2022.3164667.

[12] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[13] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, no. 1, 2016, pp. 2094–2100.

[14] J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020.

[15] Q. Guo, S. Wang, Z. Wei, and M. Chen, "Influence maximization revisited: Efficient reverse reachable set generation with bound tightened," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2020, pp. 2167–2181.

[16] X. Yang, J. Shang, Q. Hu, and D. Liu, "ARIS: Efficient admitted influence maximizing in large-scale networks based on valid path reverse influence sampling," *IEEE Trans. Emerg. Topics Comput.*, early access, Dec. 27, 2022, doi: 10.1109/TETC.2022.3230933.

[17] Z. Jin et al., "IM2Vec: Representation learning-based preference maximization in geo-social networks," *Inf. Sci.*, vol. 604, pp. 170–196, Aug. 2022.

[18] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2702–2711.

[19] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 537–546.

[20] H. Kamarthi, P. Vijayan, B. Wilder, B. Ravindran, and M. Tambe, "Influence maximization in unknown social networks: Learning policies for effective graph sampling," in *Proc. 19th Int. Conf. Auto. Agents MultiAgent Syst.*, 2020, pp. 575–583.

[21] D. Ireland and G. Montana, "LeNSE: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 9622–9638.

[22] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[23] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Oper. Res.*, vol. 134, Oct. 2021, Art. no. 105400.

[24] Y. Yang and A. Whinston, "A survey on reinforcement learning for combinatorial optimization," 2020, *arXiv:2008.12248*.

[25] C. Fan, L. Zeng, Y. Sun, and Y.-Y. Liu, "Finding key players in complex networks through deep reinforcement learning," *Nature Mach. Intell.*, vol. 2, no. 6, pp. 317–324, May 2020.

[26] S.-C. Lin, S.-D. Lin, and M.-S. Chen, "A learning-based framework to handle multi-round multi-party influence maximization on social networks," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 695–704.

[27] K. Ali, C.-Y. Wang, and Y.-S. Chen, "Boosting reinforcement learning in competitive influence maximization with transfer learning," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Dec. 2018, pp. 395–400.

[28] K. Ali, C.-Y. Wang, M.-Y. Yeh, and Y.-S. Chen, "Addressing competitive influence maximization on unknown social network with deep reinforcement learning," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Dec. 2020, pp. 196–203.

[29] A. Yadav, R. Noothigattu, E. Rice, L. Onasch-Vera, L. S. Marcolino, and M. Tambe, "Please be an influencer?: Contingency-aware influence maximization," in *Proc. 17th Int. Conf. Auto. Agents MultiAgent Syst.*, 2018, pp. 1423–1431.

[30] H. Chen, W. Qiu, H.-C. Ou, B. An, and M. Tambe, "Contingency-aware influence maximization: A reinforcement learning approach," in *Proc. 37th Conf. Uncertainty Artif. Intell.*, vol. 161, in Proceedings of Machine Learning Research, C. de Campos and M. H. Maathuis, Eds. PMLR, Jul. 2021, pp. 1535–1545. [Online]. Available: https://proceedings.mlr.press/v161/chen21b.html

[31] S. Tian, P. Zhang, S. Mo, L. Wang, and Z. Peng, "A learning approach for topic-aware influence maximization," in *Proc. Asia–Pacific Web (APWeb) Web-Age Inf. Manage. (WAIM) Joint Int. Conf. Web Big Data*. Cham, Switzerland: Springer, 2019, pp. 125–140.

[32] S. Feng, G. Cong, A. Khan, X. Li, Y. Liu, and Y. M. Chee, "inf2vec: Latent representation model for social influence embedding," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 941–952.

[33] Q. Cao, H. Shen, J. Gao, B. Wei, and X. Cheng, "Popularity prediction on social platforms with coupled graph neural networks," in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 70–78.

[34] A. L. Maas et al., "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1. Princeton, NJ, USA: Citeseer, 2013, p. 3.

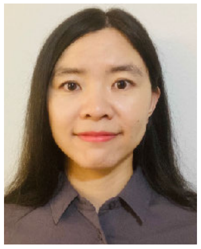
[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[36] M. A. Riedmiller, "Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.* Cham, Switzerland: Springer, 2005, pp. 317–328.

[37] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. AAAI Conf. Artif. Intell.*, vol. 29, no. 1, 2015, pp. 1–2.

[38] J. Leskovec and A. Krevl. (Jun. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. [Online]. Available: http://snap.stanford.edu/data

[39] H. Li, M. Xu, S. S. Bhowmick, C. Sun, Z. Jiang, and J. Cui, "DISCO: Influence maximization meets network embedding and deep learning," 2019, *arXiv:1906.07378*.



Tiantian Chen received the B.S. degree in mathematics and applied mathematics and the M.S. degree in operational research and cybernetics from the Ocean University of China, Qingdao, China, in 2016 and 2019, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA.

Her research focuses on reinforcement learning, deep learning, social networks, blockchain, and design and analysis of approximation algorithms.

Siwen Yan received the B.E. degree in measurement, control technique and instruments from the Harbin Institute of Technology, Harbin, China, in 2015, and the M.S. degree in electrical and computer engineering from the University of California San Diego, La Jolla, CA, USA, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA.

His current research interests include probabilistic graphical models, statistical relational AI, graph neural networks, AI applications in healthcare, and reinforcement learning.

Jianxiong Guo (Member, IEEE) received the Ph.D. degree in computer science from the University of Texas at Dallas, Richardson, TX, USA, in 2021.

He is currently an Assistant Professor with the Advanced Institute of Natural Sciences, Beijing Normal University, Zhuhai, China, and the Guangdong Key Laboratory of AI and Multi-Modal Data Processing, BNU-HKBU United International College, Zhuhai. He has authored or coauthored more than 40 peer-reviewed papers and been the reviewer for many famous international journals/conferences. His current research interests include social networks, wireless sensor networks, combinatorial optimization, and machine learning.

Weili Wu (Senior Member, IEEE) received the M.S. and Ph.D. degrees from the Department of Computer Science, University of Minnesota, Minneapolis, MN, USA, in 1998 and 2002, respectively.

She is currently a Full Professor with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA. Her research mainly deals in the general research area of data communication and data management. Her research focuses on the design and analysis of algorithms for optimization problems that occur in wireless networking environments and various database systems.