Finding Critical Users in Social Communities via Graph Convolutions

Kangfei Zhao*, Zhiwei Zhang†, Yu Rong‡, Jeffrey Xu Yu*, Junzhou Huang§
*The Chinese University of Hong Kong, {kfzhao, yu}@se.cuhk.edu.hk
†Beijing Institute of Technology, cszwzhang@outlook.com
‡Tencent AI Lab, yu.rong@hotmail.com
§University of Texas at Arlington, jzhuang@uta.edu

Abstract—Finding critical users whose existence keeps a social community cohesive and large is an important issue in social networks. In the literature, such criticalness of a user is measured by the number of followers who will leave the community together when the user leaves. By taking a social community as a k-core, which can be computed in linear time, the problem of finding critical users is to find a set of nodes, U, with a user-given size b in a k-core community that maximizes the number of nodes (followers) to be deleted from the k-core when all nodes in U are deleted. This problem is known to be NP-hard. In the literature, the state-of-the-art approach, a greedy algorithm is proposed with no guarantee on the set of nodes U found, since there does not exist a submodular function the greedy algorithm can use to get a better answer iteratively. Furthermore, the greedy algorithm designed is to handle k-core in any social networks such that it does not consider the structural complexity of a given single graph and cannot get the global optimal by the local optimal found in iterations.

In this paper, we propose a novel learning-based approach. Distinguished from traditional experience-based heuristics, we propose a neural network model, called Self-attentive Core Graph Convolution Network (*SCGCN*), to capture the hidden structure of the criticalness among node combinations that break the engagement of a specific social community. Supervised by sampling node combinations, *SCGCN* has the ability to inference the criticalness of unseen combinations of nodes. To further reduce the sampling and inference space, we propose a deterministic strategy to prune unpromising nodes on the graph. Our experiments conducted on many real-world graphs show that *SCGCN* significantly improves the quality of the solution compared with the state-of-the-art greedy algorithm.

Index Terms—social network analysis, k-core collapsion, graph neural network.

1 Introduction

Exploring the critical entities in networks has attracted interests in recent years [1], [2], [3], [4], as motivated by the fact that there is a need to understand who plays an important role in a large dense subgraph with possibly thousands of nodes found for analytical tasks in social networks (e.g., biological networks, etc.). This task is beneficial to many consequent mining tasks such as recommendation and influence maximization. For instance, in social network analysis, it is vital to identify the opinion leaders from existing communities.

To evaluate the criticalness of a user set, U, in a community, recent approaches model it by the number of followers, where the followers of U are the users who will take the same action to leave the community when the users in U leave. It is worth mentioning that a follower of U may not directly connect to any user in U and can be an indirect follower following some connections to a user in U. The problem of finding critical users over a social community is to identify U such that the largest number of nodes (followers) will be deleted from the social community when U are deleted.

Zhang et al. studied critical user detection as a collapsed k-core problem [1]. In brief, it considers a social community as a k-core for a given k and finds a set of nodes, U, with a given size b, that will trigger the max number of nodes to be deleted from the k-core, if all nodes in U are deleted. As shown in [1], the collapsed k-core problem is non-trivial and is proven to be NP-hard. In the scope of this paper, the problem of critical user detection is equivalent to the collapsed k-core problem. And we use critical user detection in the following definition as we concentrate on finding the node set U. To find such U, Zhang et al. in [1] proposed a greedy algorithm which is the state-of-the-art. We explain the greedy algorithm in brief. Let $f(u)$ be a function, for every node u in a k-core community, as the number of nodes to be deleted from the k-core community when the node u is deleted. The greedy algorithm will repeatedly select the node u with the largest $f(u)$ value to be added into U until $|U| = b$. The greedy algorithm performs well in practice. But, as $f(u)$ has been proven to be non-submodular, there is no error bound for the greedy algorithm to find such a set of U for any social communities. We show the drawback of the greedy algorithm using an example.

Example 1.1: DBLP dataset is a collection of researchers and their publications. We extract a co-author network from DBLP if two authors have collaborated more than 5 papers, and show a 20-core community with 119 researchers in Fig. 1. To find the critical users, U, for $b = 3$, the greedy algorithm will identify the red node as the top-1 critical user in the first iteration, because the red node has the largest

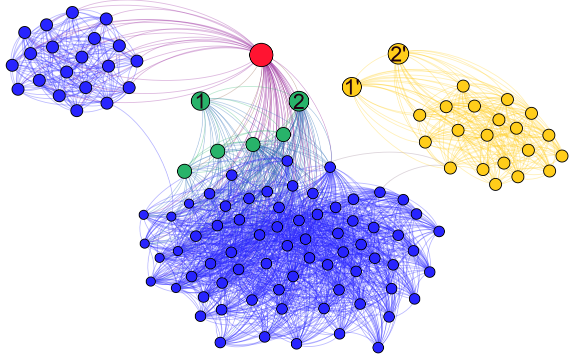


Fig. 1. Comparison between the greedy algorithm and *SCGCN*

number of nodes, 20, to be deleted from the community if it is deleted. After deleting 21 nodes (the critical user and its 20 followers) from the current 20-core, the remaining nodes in the 20-core have 0 followers, which means that every node has more than 20 neighbors in the 20-core and there is no follower from the community for any node to be deleted. In this case, the greedy algorithm selects the node 1 as the top-2 critical user in a random fashion. For the top-3 critical user, it then selects the node 2 which has 4 followers. As for the greedy algorithm, deleting the two nodes, 1 and 2, will lead to 6 nodes (the green nodes) to be deleted. In fact, there is a better answer. Suppose node $1'$ as the top-2 critical user instead. Even though node $1'$ itself cannot trigger any followers to be deleted, the next removal of the top-3 (e.g., node $2'$) will trigger 21 nodes to be deleted. In total, there are 22 nodes (the yellow nodes) to be deleted from the 20-core due to the removal of node $1'$ and node $2'$. □

As shown in Example 1.1, the greedy algorithm, which is designed to handle any possible k-core community, has its limit to find a better set of critical users, U, with a given size b for a certain k-core community due to the following two complexities, namely *structural complexity*, and *combinatorial complexity*. Here, structural complexity is related to the inherent topology of a given community. The greedy algorithm [1], like most greedy algorithms designed for solving NP-hard problems, adopts heuristics based on the problem solely. In an iteration, it is designed with heuristics to approximate the global optimum by the local optimum for some graph. The greedy algorithm does not take the characteristics of the data graph into account. It may work well for some graphs, but fails for some other graphs. On the other hand, the combinatorial complexity is related to the exploration of all possible combinations, which is prohibitively high.

To address such complexities for detecting critical users, in this paper, we propose a learning-based approach, which has achieved remarkable success in approximating complex functions in a wide range of applications. Different from the greedy algorithm which uses a fixed $f(u)$ function to select a node, we learn a specific $f(u)$ function via sampling and embedding node sets (i.e., partial solutions) from a given graph. Specifically, a graph neural network model, called Self-attentive Core Graph Convolution Network (*SCGCN*) is proposed to learn a better local approximation function $f(u)$ in an end-to-end fashion. *SCGCN* does not only consider the criticalness of every single node, but also the criticalness for multiple nodes, which leverages the hidden local properties of a complex graph. To summarize briefly, *SCGCN* takes both the structural complexity and combinatorial complexity into consideration and employs a graph convolution model as well as a self-attentive mechanism for the problem of critical user detection. We highlight our main contributions as follows.

TABLE 1
Frequently Used Notations

Notations	Definitions
$G(V, E)$	An undirected graph with node set V and edge set E.
$\deg(u, G)$	The number of adjacent vertices of u in G.
$\mathsf{C_k}(G)$	The k-core of G.
G_U	The induced subgraph of G after removing U.
$\mathsf{Cr}(u)/\mathsf{Cr}(U)$	The criticalness of a node u/a node set U.
$\mathsf{fl}(u, G)$	The node set representing the followers of u in G.

- We propose a learning-based approach for finding critical users to solve the collapsed k-core problem. Different from the greedy algorithm discussed, our model, *SCGCN*, leverages the hidden structure of node combinations with respect to followers collapsing, and performs inference on unseen combinations.
- To accelerate the model training, we explore the properties of the critical users and reduce unpromising nodes, which are only deterministic to the graph structure. Thereby, a large number of nodes can be reduced from both the training and inference space in advance, which does not affect the quality of the solution to be found.
- We conduct extensive experiments on real-world graphs. In our testing, *SCGCN* outperforms the existing approaches considerably. Our model is more effective in denser communities (cores of larger k) and has generalization ability on different b.

Organization: We give the problem statement in Section 2 and discuss the greedy algorithm in detail in Section 3. We introduce our learning approach in Section 4 and report the experimental studies in Section 5 followed by the related works in Section 6. We conclude this work in Section 7.

2 Problem Statement

We model an undirected graph as $G = (V, E)$, where $V(G)$ and $E(G)$ represent the set of nodes and edges of G, respectively. We denote the adjacent nodes of u in graph G as $\mathsf{nbr}(u, G) = \{v \mid (v, u) \in E(G)\}$, and denote the number of adjacent nodes of u in G, as $\deg(u, G)$ such that $\deg(u, G) = |\mathsf{nbr}(u, G)|$. To model the community in a graph, we focus on k-core, which has been widely used as a cohesive subgraph. The k-core of G is defined as the maximal subgraph, denoted as $\mathsf{C_k}(G)$, such that $\deg(u, \mathsf{C_k}(G)) \geq k$ for every node $u \in V(\mathsf{C_k}(G))$, for a given k.

In this paper, we study finding critical users in a social network, where a critical user is a user in a community whose leave will trigger his/her followers to leave. To capture such followers in a community, we adopt the idea used in [1], which studies the collapsed k-core problem. In brief, a community is a k-core, $\mathsf{C_k}(G)$, for a certain k value. Suppose a user (node), u, leaves (is deleted from) the

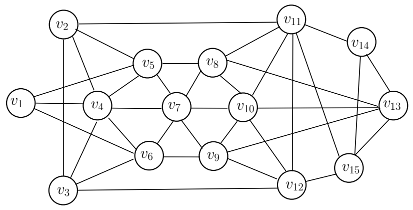


Fig. 2. A 3-core graph G

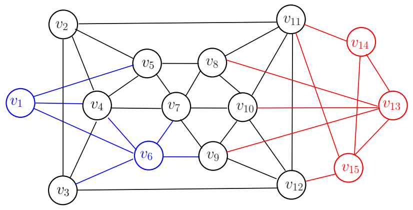
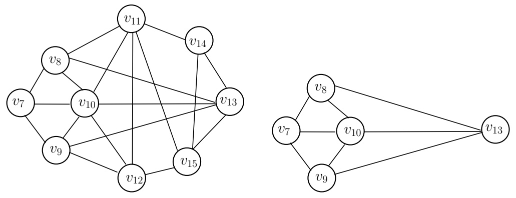


Fig. 3. The 3-cores as the results of the greedy algorithm



(a) A better solution (b) The optimal solution

Fig. 4. A better and the optimal solutions

Algorithm 1 *Greedy Critical User* [1]

```
Input: A Graph $G$, integers $k$ and $b$;
Output: Critical users $U$;
1: $U \leftarrow \emptyset$;
2: while $|U| < b$ do
3:     compute $f(u, U, G)$ for every node $u \in \mathsf{C_k}(G_U)$;
4:     $u^* \leftarrow \text{argmax}_u f(u, U, G)$, $U \leftarrow U \cup \{u^*\}$;
5: end while
6: return $U$;
```

community ($\mathsf{C_k}(G)$), such an action will trigger other nodes in $\mathsf{C_k}(G)$ to be deleted from $\mathsf{C_k}(G)$. Such other nodes are considered as followers of u in G, denoted as $\mathsf{fl}(u, G)$. The criticalness of a node u in $\mathsf{C_k}(G)$ is the number of nodes (followers) that are deleted from $\mathsf{C_k}(G)$ if u is deleted from $\mathsf{C_k}(G)$. Table 1 summarizes the frequently used notations in this paper.

In [1], the collapsed k-core problem is to find a set of nodes, U, instead of a single node u. Let G_U denote the graph G after deleting both the set of nodes U and the edges that incident to any node in U, the collapsed k-core is $\mathsf{C_k}(G_U)$. Accordingly, the followers of the set of nodes, U, are $\mathsf{fl}(U, G) = \mathsf{C_k}(G) \setminus (\mathsf{C_k}(G_U) \cup U)$, and the criticalness of U is the number of followers of U ($|\mathsf{fl}(U, G)|$). The problem is to find a set of nodes, U, that maximizes $|\mathsf{fl}(U, G)|$. In this paper, we study the same problem. Due to the reason that we will explore learning approaches with probability, we define the criticalness of U including U and its followers, such as $\mathsf{Cr}(U) = |\mathsf{fl}(U, G) \cup U|$. This modification does not affect the optimal answer of the collapsed k-core problem, and does not affect the hardness of the collapsed k-core problem [1]. We give the problem statement below.

Problem Statement [1]: Given a graph G, a value k for a k-core, and the size of a set nodes b, the problem of critical user detection is to find the set of nodes $U_{max} = \arg\max_U \mathsf{Cr}(U)$ where $|U| = b$.

The set of nodes U_{max}, representing the optimal critical users, is the set of nodes that have the largest criticalness, $\mathsf{Cr}(U)$. Note that this problem is proved to be NP-hard for any $k > 3$ in [1].

3 The Greedy Algorithm

Since the problem of critical user detection (collapsed k-core) is NP-hard for any $k > 3$, a greedy algorithm is proposed in [1]. The greedy algorithm is sketched in Algorithm 1. It takes three inputs, namely, the graph G, the value of k for the k-core, and the size of nodes, b. Initially, U is set to be empty. In a while loop (lines 2-5), it adds a node u^* into U in every iteration until $|U| = b$. Here, we use a function $f(u, U, G)$, to compute $\mathsf{Cr}(u)$ given the current G_U. The reason to use a function form is that we need a function in our learning approach. The node u^* selected is the one with the max $f(u, U, G)$ value among the nodes in the current G_U.

The greedy algorithm is designed to handle any graph G and can get a good answer in practice. A question we want to ask is whether we can do better than the greedy algorithm for a certain given graph G by learning. Below, we discuss it using an example and show that there are chances to do better by learning.

Example 3.1: Consider a 3-core community G shown in Fig. 1. Suppose we want to find top-2 critical users in G, by giving $k = 3$ and $b = 2$. Initially, $U = \emptyset$. Hence, $\mathsf{Cr}(u)$ for every node u in $G_U = G$ is computed and its value is shown in the first row in Table 2. Here, v_{13} has the largest criticalness 3 ($\mathsf{Cr}(v_{13}) = 3$) if v_{13} is deleted. We explain it as follows. By deleting v_{13} from the 3-core, the degree of v_{14}, $\deg(v_{14}, G)$, will change to 2, and will be deleted from the 3-core. Similarly, v_{15} will be deleted when v_{14} is deleted. Suppose G' is the subgraph G by deleting nodes, $\{v_{13}, v_{14}, v_{15}\}$, and their incident edges from G. the new 3-core, G', is shown in Fig. 1 without the red nodes/edges. In the 3-core G', the largest Cr value among the nodes is 2, as shown in the second row in Table 2. Without loss of generality, assume v_6 is deleted. In this scenario, v_1 will be deleted as $\deg(v_1, G') = 2$ after deleting v_6, as shown in the blue nodes/edges in Fig. 1. As a result in total, the greedy algorithm select, $U_1 = \{v_{13}, v_6\}$, as top-2 critical users whose deletion will delete 5 nodes in total including U_1 itself, such that $\mathsf{Cr}(U_1) = 5$.

Alternatively, suppose we pick v_6 initially in the first iteration, even though $\mathsf{Cr}(v_2) = 2$ which is smaller than $\mathsf{Cr}(v_{13}) = 3$, and assume that we find $U_2 = \{v_6, v_2\}$. The criticalness of U_2 is larger than that of U_1, since $\mathsf{Cr}(U_1) = 6$. The resulting collapsed 3-core by U_2 is shown in Fig. 4(a).

The optimal top-2 critical users are $U_O = \{v_4, v_{11}\}$, and the collapsed 3-core is shown in Fig. 4(b). □

There are some observations that can be made about the greedy algorithm (Algorithm 1). First, in practice, the greedy algorithm performs well. But, the function, $f(u, U, G)$ (which is $\mathsf{Cr}(u)$ in the greedy algorithm), is not submodular. This indicates that there is no error bound guarantee for the greedy algorithm. Second, the greedy algorithm designed attempts to find the local optimal in every iteration, which is impossible to find the global optimal top-b critical users in a k-core community, because social networks can be complex and the algorithm designed cannot explore all possible complexities in all different power-law graphs, given the efficiency constraint. Third, the function $f(u, U, G)$ is non-injective. As a simple example, consider Example 3.1.

TABLE 2
The criticalness computed in iteration by Algorithm 1

$V(G)$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
$\mathsf{Cr}(v)$ in G	1	1	1	2	2	2	1	1	1	1	2	1	**3**	1	2
$\mathsf{Cr}(v)$ in G'	1	1	1	**2**	**2**	**2**	1	1	1	1	1	1	-	-	-

Suppose it needs to find top-4 critical users for the 3-core community. When it is about to select the top-2 critical user, there are three candidates which have the same criticalness, $\mathsf{Cr}(v_4) = \mathsf{Cr}(v_5) = \mathsf{Cr}(v_6) = 2$, as shown in the second row in Table 2. The greedy algorithm will randomly pick up one among the three candidates with the hope that it can get a better answer for the top-4 critical users, which may result in an answer being away from the optimal.

Given the issues discussed, in this paper, we attempt to find a way to learn a function that can effectively capture the graph property, and guide it to a better solution for the critical users.

4 A New Learning-Based Approach

In this section, we propose a learning-based approach inspired by the universal approximation theorem [5]. Our main idea is to replace the function $f(u, U, G)$ $(= \mathsf{Cr}(u))$ in the greedy algorithm (Algorithm 1), by an expressive parameterized function as $f(u, S, G; \Theta)$, which is learned by a general neural network. Distinguished from $f(u, U, G)$, learning $f(u, S, G; \Theta)$ is supervised by the criticalness of possible partial solutions, where a partial solution is a set of nodes, S, with any size smaller than b ($|S| < b$). In other words, the neural network exploits node sets, S, for any size from 1 to $b-1$, over the k-core to encode the criticalness of nodes in forming possible solutions. Intuitively, in training, nodes with a higher criticalness tend to cluster in the vector space, whereas nodes with a lower value are kept away. Therefore, a set of high-quality critical users can be identified by set expansion towards a direction with high criticalness value in the vector space. In brief, $f(u, S, G; \Theta)$ takes a vectorized representation of a partial solution S with $1 \sim b-1$ nodes as input and predicts the likelihood of inserting a node u to S. The $f(u, S, G; \Theta)$ being learned will replace $f(u, U, G)$ seamlessly in Algorithm 1 to generate a solution. In the following, we discuss our approach including (a) modeling the combinational criticalness among nodes, (b) sampling on the graph G (or precisely $\mathsf{C_k}(G)$) to generate training data, and (c) designing the graph neural network based model named *SCGCN*.

4.1 Combinatorial Criticalness Modeling

From the learning perspective, $f(u, S, G; \Theta)$ indicates how probable a node u can be added into the current partial solution S to maximize $\mathsf{Cr}(S \cup \{u\})$. In our problem, the domain of partial solutions is a set of node sets, where a node set, S, is a subset of $V(\mathsf{C_k}(G))$, with the size, $|S|$, in the range of $[1, b-1]$. An evaluation function f is used to estimate the likelihood, $p(u|S)$, for a node, $u \notin S$, in $V(\mathsf{C_k}(G))$ to be selected as a critical user and added into S, given the current critical user set, S. The likelihood $p(u|S)$ is defined below, for a partial solution S and a node u.

$$p(u|S) = \frac{p(u, S)}{p(S)} \propto p(S \cup \{u\}) \tag{1}$$

In Eq. (1), $p(S \cup \{u\})$ is the marginal probability of the extended solution, which is estimated by computing the criticalness of the extended set $\mathsf{Cr}(S \cup \{u\})$ explicitly.

$$p(S \cup \{u\}) = \frac{\mathsf{Cr}(S \cup \{u\})}{\Sigma_{u \in V} \mathsf{Cr}(S \cup \{u\})} \tag{2}$$

Since the number of the possible combinations for S is exponential to $|U|$. It is impossible to memorize the results for all the possible partial solutions. To this end, we employ a parametric neural network $f(u, S, G; \Theta) = \hat{p}_\Theta(u|S)$ to learn a node representation with the preservation of the quality of the partial solutions. To train $f(u, S, G; \Theta)$, batches of partial solutions are fed into a neural network, where a partial solution S is sampled by a partial solution sampler. The corresponding soft-label is calculated by $p(u|S)$ (Eq. (2)). Hence, we use the cross entropy as the training loss, as given in Eq. (3).

$$L(S) = \Sigma_{u \in V} - p(u|S) \log(\hat{p}_\Theta(u|S)) \tag{3}$$

Given any partial solution S as input, the output of the model, $\hat{p}_\Theta(u|S) \in \mathbb{R}^{|V(\mathsf{C_k}(G))|}$ predicts the likelihood that each node should be added into S. While in the testing process, we replace $f(u, U, G)$ with the trained $f(u, S, G; \Theta)$ and generate the critical user set U w.r.t. Algorithm 1.

4.2 Reduced Classes Sampling

As mentioned in Section 4.1, we need a sampler to sample partial solutions as the training data. It is crucial to design an effective and efficient sampler to generate high-quality samples (partial solutions). Before we discuss the sampling method for the learning model to be used, we first prove that the sampling space can be reduced significantly. That is because a node v is impossible to be a critical user if it is the follower of any node u but u is not the follower of v.

Theorem 4.1: *Given a graph G and a budget b, assume $U_{max} = \arg\max_U \mathsf{Cr}(U)$. For the nodes $u, v \in G$, if $v \in \mathsf{fl}(u, G)$ and $u \notin \mathsf{fl}(v, G)$, then $v \notin U_{max}$ if $\mathsf{Cr}(U_{max}) < |V(G)|$.* □

Proof Sketch: Proof by contradiction. For any node pair $v, u \in V(G)$ s.t. $v \in \mathsf{fl}(u, G)$ and $u \notin \mathsf{fl}(v, G)$, consider two cases (1) $u \notin U_{max}$ and (2) $u \in U_{max}$.

- *Case 1: $u \notin U_{max}$.* Assume Theorem 4.1 is incorrect such that $v \in U_{max}$ and $\mathsf{Cr}(U_{max}) < |V(G)|$. As $v \in \mathsf{fl}(u, G)$, v will be deleted if u is deleted from the graph G, due to the degree constraint. Then, is has $\mathsf{fl}(u, G) = \mathsf{fl}(\{u, v\}, G)$. Also, since $u \notin \mathsf{fl}(v, G)$, we can derive that $|\mathsf{fl}(v, G)| < |\mathsf{fl}(\{u, v\}, G)|$. Thereby, $|\mathsf{fl}(v, G)| < |\mathsf{fl}(u, G)|$. In this scenario, consider the node u that $u \notin U_{max}$, we can get a new set of nodes, $U' = U_{max} \setminus \{v\} \cup \{u\}$, by replacing the node v in U_{max} with the node u. This will lead to $\mathsf{fl}(U', G) > \mathsf{fl}(U_{max}, G)$, which contradicts the assumption that $U_{max} = \arg\max_U \mathsf{Cr}(U)$.

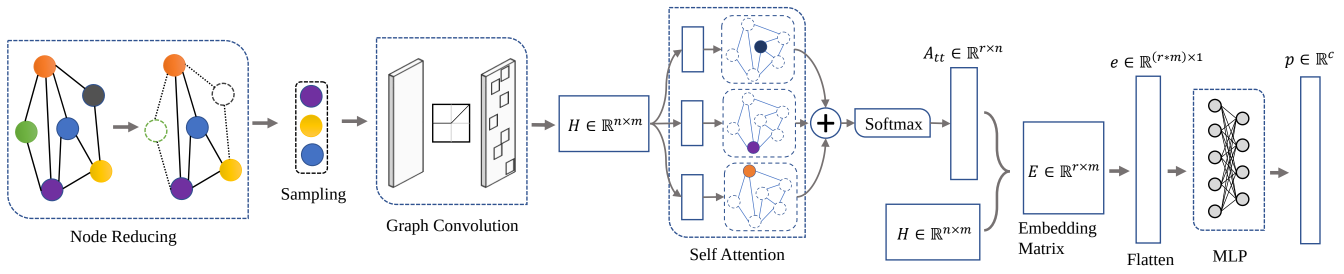

Node Reducing
Sampling
Graph Convolution
$H \in \mathbb{R}^{n\times m}$
Self Attention
Softmax
$A_{tt} \in \mathbb{R}^{r\times n}$
$H \in \mathbb{R}^{n\times m}$
$E \in \mathbb{R}^{r\times m}$
Embedding Matrix
$e \in \mathbb{R}^{(r*m)\times 1}$
Flatten
MLP
$p \in \mathbb{R}^{c}$

Fig. 5. The overview of *SCGCN*: The k-core structure and sampled node sets are fed into a two-layer GCN, followed by a self-attention layer. A fixed-length graph representation E is generated and flatten to a vector, passing through an MLP to approximate function f. It is a special application of *DeepSets* [6], where GCN generates element-wise embedding, self-attention performs a weighted sum pooling and MLP implements the set-level approximation.

- *Case 2:* $u \in U_{max}$. Assume Theorem 4.1 is incorrect such that $v \in U_{max}$ and $\mathsf{Cr}(U_{max}) < |V(G)|$. Since $\mathsf{Cr}(U_{max}) < |V(G)|$, at least one node w exists in $\mathsf{C_k}(G_{U_{max}})$ which is the k-core of the remaining graph. Consider the node set $U'' = U_{max} \setminus \{v\} \cup \{w\}$. As $v \in \mathsf{fl}(u,G)$, it can be derived that $\mathsf{fl}(U_{max}, G) = \mathsf{fl}(U_{max} \setminus \{v\}, G)$. Thus adding the node w into $U_{max} \setminus \{v\}$ will increase at least one $\mathsf{Cr}(U'')$, as w itself will be deleted. It contradicts the assumption of $U_{max} = \arg\max_U \mathsf{Cr}(U)$.

Thus, it has $v \notin U_{max}$ if $v \in \mathsf{fl}(u,G)$ and $u \notin \mathsf{fl}(v,G)$. □

According to Theorem 4.1, if $v \in \mathsf{fl}(u,G)$ and $u \notin \mathsf{fl}(v,G)$, there are two possible cases regarding a partial solution S. One case is when $v \in S$ but $u \notin S$. Here, we can replace v by u without the degradation of criticalness. The other is when $v \in S$ and $u \in S$. Then, we can safely delete v from S without the degradation of criticalness, which leaves more possibility for other nodes to be added into S.

Here, we use $\mathsf{re}(G)$ to denote all the nodes, v, in G such that $v \in \mathsf{fl}(u,G)$ for a certain node u and $u \notin \mathsf{fl}(v,G)$. To detect critical users is then reduced to finding a node set $U \subset V \setminus \mathsf{re}(G)$, where $\mathsf{Cr}(U)$ is maximized and $|U| = b$. This $\mathsf{re}(G)$ contains the nodes that cannot be critical users, and therefore it is not necessarily to sample nodes from $\mathsf{re}(G)$.

Example 4.1: Consider the graph G shown in Fig. 1 for a given $k = 3$. We have $\mathsf{fl}(v_{13}, G) = \{v_{14}, v_{15}\}$, $\mathsf{fl}(v_{15}, G) = \{v_{14}\}$, $\mathsf{fl}(v_4, G) = \mathsf{fl}(v_5, G) = \mathsf{fl}(v_6, G) = \{v_1\}$, and $\mathsf{fl}(v_{14}, G) = \mathsf{fl}(v_1, G) = \varnothing$ by imposing the degree constraint. Since $v_{14} \in \mathsf{fl}(v_{13}, G)$ and $v_{13} \notin \mathsf{fl}(v_{14}, G)$, we have $v_{14} \in \mathsf{re}(G)$. Similarly, since $v_{15} \in \mathsf{fl}(v_{13}, G)$ and $v_{13} \notin \mathsf{fl}(v_{15}, G)$, we have $v_{15} \in \mathsf{re}(G)$. Also, we have $v_1 \in \mathsf{re}(G)$ because of $v_1 \in \mathsf{fl}(v_4, G)(\mathsf{fl}(v_5, G), \mathsf{fl}(v_6, G))$ and $v_4(v_5, v_6) \notin \mathsf{fl}(v_1, G)$, Therefore, we have $\mathsf{re}(G) = \{v_1, v_{14}, v_{15}\}$, which are the nodes can be safely reduced, since they cannot be critical users. □

In our sampling, instead of uniformly sampling partial solutions, S, from the whole node set $V(G)$, we only sample S from $V(G) \setminus \mathsf{re}(G)$. Each candidate $u \in V(G) \setminus \mathsf{re}(G)$ is sampled with a sampling probability $\mathsf{prob}(u)$, i.e., the normalized criticalness of u (Eq. (4)):

$$\mathsf{prob}(u) = \frac{\mathsf{Cr}(u)}{\Sigma_{v \in V \setminus \mathsf{re}(G)} \mathsf{Cr}(v)} \tag{4}$$

The biased sampling motivates the neural network model to learn well on nodes of high criticalness. There are alternative biased sampling and fine-tuning approaches, which are beyond the scope of this paper. The significance of reducing nodes $\mathsf{re}(G)$ is two folds. First, it reduces the sampling space in the training process, which improves the effectiveness and efficiency of training. Second, it also prunes $\mathsf{re}(G)$ from the solution space since the neural network only needs to inference the probability $p(u)$ for $u \in V \setminus \mathsf{re}(G)$ in Eq. (2).

To this end, the learned function $f(u, S, G; \Theta)$ improves the greedy algorithm by leveraging the critical combinations of node sets, which are sampled from the graph based on the node criticalness. In other words, we use observed sets to fit the function of joint criticalness for any node sets. The biased sampling and sample space reduction enable the observation to concentrate on high potential critical users. To implement $f(u, S, G; \Theta)$, below we introduce a graph neural network model, which finally handles combinatorial complexity and structural complexity in an end-to-end framework.

4.3 The Proposed Learning Framework: *SCGCN*

To learn $f(u, S, G; \Theta)$ by samples of node sets on k-core, we adopt a deep neural network architecture *DeepSets* [6], which operates on sets and captures the structure of permutation invariant functions. For a set S, each element in S is transformed into an element-wise representation. Then, all the element-wise representations are summed up by a pooling function, e.g., classic aggregation sum, generating a set-level representation. Finally, the add-up representation is processed by other neural networks. There are two main challenges when applying this architecture to our problem:

- Leveraging the graph topology as much as possible to perform element-wise embedding on the graph.
- Designing a flexible pooling function that encodes the importance of different nodes simultaneously.

In the literature, graph representation learning has recently shifted from hand-crafted kernel method [7] to unsupervised proximity preserving [8], [9] and graph neural networks approaches [10], [11]. In this vein, we propose a new model called *SCGCN*. In *SCGCN*, we adopt the GCN for node embedding, generating the element-wise representation for node appearance/absence in the sampled set. For the pooling function, we use a self-attention mechanism [12], [13] to aggregate the node embedding to a fixed-length graph representation as the set-level representation, which also encodes the importance of different nodes to the graph representation. The overall architecture of our learning model is shown in Fig. 5.

Given the adjacency matrix $A \in \mathbb{R}^{n \times n}$ of a k-core with n nodes. A pre-processing step firstly computes the normalized adjacency matrix $\hat{A}$ as shown in Eq. (5), where I_n is the $n \times n$ identity matrix and $D = diag(d)$ for $d(i) = \Sigma_i (A + I_n)_{ij}$.

$$\hat{A} = D^{-\frac{1}{2}}(A + I_n)D^{-\frac{1}{2}} \tag{5}$$

The formula in Eq. (6) defines the two-layer GCN [11]. The matrix W_i is the weights of the i-th layer, where $W_0 \in \mathbb{R}^{\phi \times \rho}$ and $W_1 \in \mathbb{R}^{\rho \times m}$, respectively.

$$H = \hat{A}\ \mathsf{ReLU}(\hat{A}XW_0)W_1 \tag{6}$$

As indicated in [14], a graph convolution layer can be regarded as a special Laplacian smoothing on the node feature for generating new features. Taking the first layer as an example, intuitively, each node aggregates the feature $X \in \mathbb{R}^{n \times \phi}$ passed from its neighborhoods. Then, the aggregated feature passes a linear transformer with parameter W_0, and activated by an element-wise rectified linear unit ReLU, thereby generating a new feature of ρ-dim for each node.

As shown in Fig. 5, we first utilize the two-layer graph convolution to smooth the node features over the graph topology. Then the generated node representations are added up by a self-attentive mechanism [12]. Let $H \in \mathbb{R}^{n \times m}$ be the node representation generated by the two-layer GCN. The self-attentive mechanism takes the whole node representation as input, and output the annotation matrix A_{tt} as the attention weights (Eq. (7)).

$$A_{tt} = \mathsf{softmax}(W_{s2}\mathsf{tanh}(W_{s1}H^T)) \tag{7}$$

Here, $W_{s1} \in \mathbb{R}^{d \times m}$ and $W_{s2} \in \mathbb{R}^{r \times d}$ are two weight matrices. The weight W_{s1} transforms the node representation linearly from m-dim space to d-dim space, which is activated by the non-linear tanh. Subsequently, the weight W_{s2} learns the importance of each node on the k-core, in r aspects. It serves as there are r experts rating the importance of nodes in independent perspectives. Then, the softmax function is applied along the second dimension of its input to ensure the computed attention weights of each expert sum up to 1. This structured attention (Eq. (7)) can be regarded as a two-layer multilayer perceptrons (MLP) without bias. The MPL has d hidden units and the weights of the two layers are W_{s1} and W_{s2}, respectively.

By multiplying the annotation matrix A_{tt} with the node representation H as Eq. (8), a final graph representation $E \in \mathbb{R}^{r \times m}$ is computed. Eq. (8) acts as a pooling layer, which sums up the embedding of each node weighted by the attention learned in Eq. (7). It is worth noting that E is a fixed-length representation controlled only by the hyper-parameters r and m.

$$E = A_{tt}H \tag{8}$$

For a sampled node set S, we assign the input node feature H_0 as the identity vector representation of S, i.e., $H_0(i) = 1$ if the i-th node is in S, otherwise $H_0(i) = 0$. Auxiliary node features like degree, the core number of each node can be concatenated to H_0. Processed by the two-layer GCN and the attention layer, we obtain a set-level embedding matrix $E \in \mathbb{R}^{r \times m}$. After flatting E to a vector e, we feed e to an MLP with sigmoid activation to estimate the probabilities $f(u, S, G; \Theta)$ for the given node u:

TABLE 3
Datasets

Graphs	|V|	|E|	Description
Facebook [15]	4,039	88,234	social net
Brightkite [16]	58,228	214,078	friendship net
WormNet [17]	16,347	762,822	gene net
DBLP [15]	317,080	1,049,866	co-authorship net
Wiki-Talk [16]	519,403	1,604,100	communication net
Google+ [15]	107,614	13,673,453	social net

$$f(u, S, G; \Theta) = \hat{p}_\Theta(u|S) = \mathrm{MLP}(e)[u], \tag{9}$$

where the output of the MLP is a c-dim vector, i.e., the estimation of Eq. (2), where $c = |V \setminus \mathsf{re}(G)|$.

In the training phase, given a vectorized sampled partial solution, $f(u, S, G; \Theta)$ is trained by the loss in Eq. (3). While in the prediction phase, we replace $f(u, U, G)$ with the trained $f(u, S, G; \Theta)$ and generate the critical user set U w.r.t. Algorithm 1. We analyze the time complexity of *SCGCN*. For training, the time complexity is $O(sb_s(l|E| + rm^2d|V| + rmhc))$. Here, s and b_s are the number of training steps and the mini-batch size, respectively, and sb_s is the number of used samples in total. For each sample, the complexity consists of three parts, the graph convolution layers ($l|E|$), the self-attention layers ($rm^2d|V|$) and the MLP ($rmhc$). Here, l is the number of convolutional layers, $|V|$ and $|E|$ are the number of nodes and edges. m is the output embedding dimension of graph convolutions and r, d are the hyper-parameters of the self-attention. rm, as the size of the output of self-attention, is the input size of a two-layer MLP with h hidden units and c predicted classes. Therefore, the final training complexity is $O(sb_s(l|E| + rm^2d|V|))$ for short and the testing complexity is $O(b(l|E| + rm^2d|V|))$.

5 Experimental Study

In this section, we present our experimental studies. First, we give the setting of the testing, including the datasets, the models and training configurations, and the baseline algorithms. Then, we conduct experiments to investigate the following facets: (1) Compare the proposed model with the existing baseline algorithms on real graphs (Section 5.2). (2) Analyze the effectiveness of *SCGCN* on varying k (Section 5.3). (3) Investigate the generalization capability of models trained by different node set sizes (Section 5.4). (4) Conduct a case study on the DBLP co-author network (Section 5.5). (5) Compare the running time of *SCGCN* with the baseline algorithms (Section 5.6).

5.1 Experiment Setup

Datasets: We use six real graphs collected from Stanford SNAP [15], KONECT [16] and Network Data Repository [17]: *Facebook* is the Facebook friendship network. *Brightkite* is a location-based social network, where a pair of nodes, i.e., users, has an edge if there is a friendship. *WormNet* is the gene network where the nodes denote genes, and edges denote gene functional associations. *DBLP* is the co-authorship network of research papers in the field of computer science. *Wiki-Talk* is the communication network

TABLE 4
The hyper-parameter configuration

hyper-parameters	Values
learning rate	$10^{-3} \sim 10^{-4}$
mini-batch size	$16 \sim 128$
training steps	$50 \sim 5 \times 10^3$
weight decay (L2 penalty)	$10^{-3} \sim 10^{-5}$
dropout probability	$0.1 \sim 0.9$
# GCN hidden units m	{64, 128, 256, 512}
# attention layer hidden units r, d	{32, 64, 128, 256}
# MLP hidden units	{64, 128, 256, 512}

of Wikipedia where nodes represent users, and an edge represents a message sent between two users. *Google+* is the Google+ social network. Table 3 summarizes the information of these real graphs. We extract multiple k-cores of the graphs in Table 3, where k is starting from around half of the maximum core number of the graph, respectively. These cores have $300 \sim 3000$ nodes. In general, we aim to find around $0.5\% \sim 2\%$ of the nodes in the k-core as the critical users U.

Implementation and Setting: The learning framework is built on *PyTorch* [18] with Python 3, while the node sets sampling is implemented by parallelized C++ binding [19]. For sampling one set, we first sample a set size in the range of $[1, b-1]$ uniformly, then we sample nodes of this size by the probability prob in Eq. (4). We use Adam [20] optimizer with a learning rate decay to train our model. Table 4 shows the hyper-parameters configuration in the training. The models are learned with these parameters tuned in the corresponding experienced range. Here, we use a two-layer MLP to encode the set-level representation and generate the final output. The hidden layer is activated by ReLU and the output layer uses the softmax to predict the probabilistic distribution (Eq. (2)). Both training and prediction are performed on a single Tesla P40.

Baseline Approaches: We compare our model *SCGCN* with the following approaches: (1) *Greedy*: the greedy algorithm proposed in [1] (Algorithm 1). This algorithm greedily selects b nodes, where in each iteration, it selects one node with the maximum criticalness, regarding the current graph. The time complexity of this algorithm is $O(bnm)$, where m is the cost of computing the criticalness for one node. (2) *Degree*: the baseline method in [1]. In each iteration, it greedily removes a node with the maximum degree and adds it to the solution. Meanwhile, the degree of the influenced nodes is updated. (3) *PageRank*: An algorithm that selects b nodes greedily. In each iteration, it selects a node with the maximum PageRank score [21], removing it and its followers. And the PageRank value is recomputed for the current k-core. The PageRank is computed by power iteration with the dumping factor of 0.85. (4) *Eigencentrality*: Similar to *PageRank*, instead we use the eigenvector centrality [22] to measure the importance of the node. The eigenvector centrality is computed by power iteration and the edge weights are assigned uniformly. All the approaches above as well as *SCGCN* are greedy strategies (Algorithm 1) with different heuristics $f(u, U, G)$, i.e., criticalness, degree, PageRank, eigenvector centrality, $f(u, S, G; \Theta)$. In one iteration, if multiple node candidates have the same $f(u, U, G)$ value, we break the tie by choosing the one with minimum node id.

5.2 Effectiveness on Real Graphs

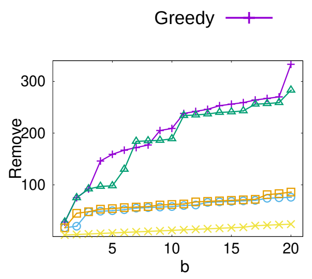
First and foremost, we test the effectiveness of our approach on various k-cores, which is the ability to drag away followers as the selected critical users are removed.

Overall Results and Analysis. Fig. 6 shows the overall results over various k-core of the six real graphs. These figures show the growth of removed followers as the selected critical users are removed gradually. In general, our approach *SCGCN* can outperform its competitors in most cases. Regarding the trend of community collapsion with the leave of critical users, from a macro perspective, the collapsion process behaves as stable collapsing with jumping points. The stable collapsing derives from the local quantitative removing of one critical user, while the jumping points derive from the accumulative effect of previous collapsing, which bring qualitative influence on the communities. The complexity of graph topology, as well as the potential jumping points, makes the problem more complicated in specific cases. *Greedy* performs well if the number of removals is always linear to the budget size b. However, due to the existence of these nodes that change the structure of the community qualitatively, the existing approaches cannot detect the jumping points as early as possible, and miss some critical users. Under this circumstance, our learning-based approach *SCGCN* can find the jumping points effectively to collapse more followers than the existing solutions.

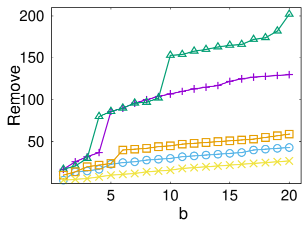
As a closer observation, we further elaborate on the results shown in Facebook. In Fig. 6(b), *SCGCN* finds a jumping point at $b = 4$, slightly earlier than *Greedy*. In the following, at $b = 10$, it finds another jumping point, leading to a 43% improvement of *Greedy*. Finally, at $b = 20$, it achieves a 55% improvement of *Greedy*. For the 40-core of Facebook (Fig. 6(c)), *SCGCN* finds a jumping point much earlier than *Greedy*, finally outperforms *Greedy* 300%. For the 50-core of Facebook (Fig. 6(d)), *SCGCN* finds a small jumping point at $b = 16$, further improving the number of collapsed followers by 48% for *Greedy*.

Another interesting observation is that *SCGCN* can collapse the whole community with fewer selected critical users. For example, in the 50-core of Brightkite (Fig. 6(h)), the whole community is collapsed by 3 critical users in *Degree*, *PageRank*, *Eigencentrality* and our *SCGCN*, while *Greedy* generates a solution of 5 nodes for this complete collapse. And similar phenomenons are also shown in WormNet (Fig. 6(l)) and Wiki-Talk (Fig. 6(t)). In most cases, *SCGCN* successfully finds the jumping points earlier than other methods. Furthermore, we observe that before finding the jumping points, *SCGCN* always selects an alternative candidate to remove, which could has slightly fewer local quantitative removing followers. But this selection leads to the better final results, such as Fig. 6(b), Fig. 6(k), Fig. 6(o) and Fig. 6(w). It indicates that *SCGCN* successfully captures the long-term accumulative qualitative collapsing and overcomes the short-term deficiency. It is the main reason that *SCGCN* can outperform its competitors.

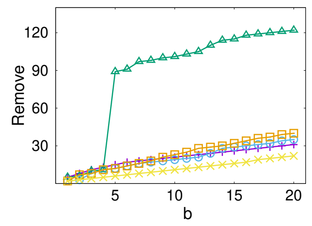
In addition, Fig. 6 suggests that *PageRank* and *Eigencentrality* perform similarly to *Degree* in most cases. In fact,



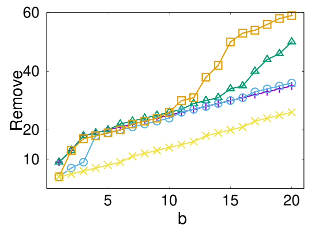
(a) Facebook $k = 20$



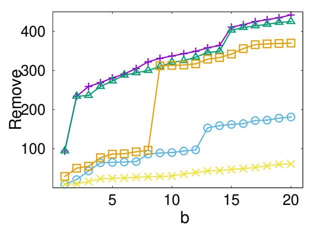
(b) Facebook $k = 30$



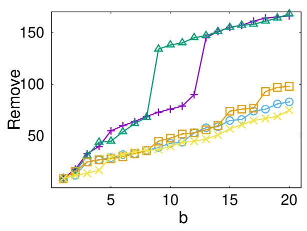
(c) Facebook $k = 40$



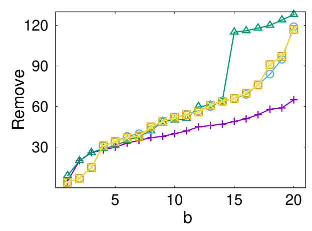
(d) Facebook $k = 50$



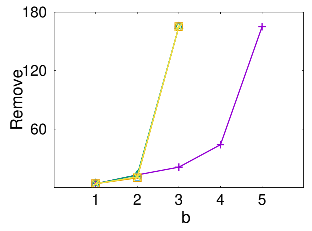
(e) Brightkite $k = 20$



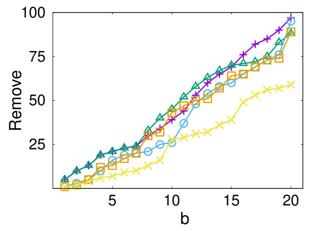
(f) Brightkite $k = 30$



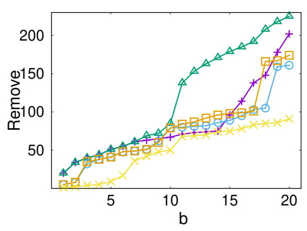
(g) Brightkite $k = 40$



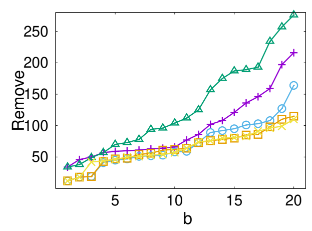
(h) Brightkite $k = 50$



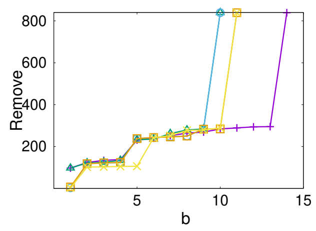
(i) WormNet $k = 130$



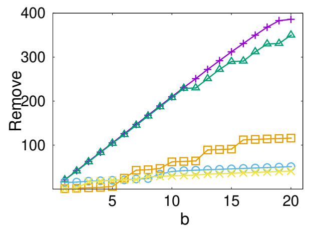
(j) WormNet $k = 140$



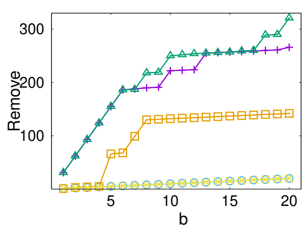
(k) WormNet $k = 150$



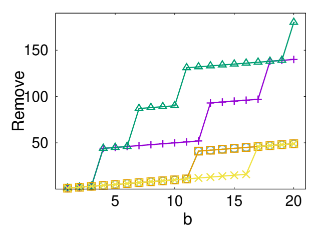
(l) WormNet $k = 160$



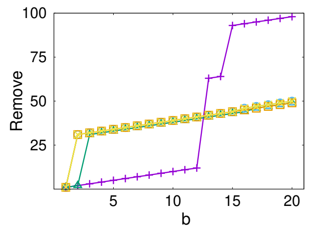
(m) DBLP $k = 20$



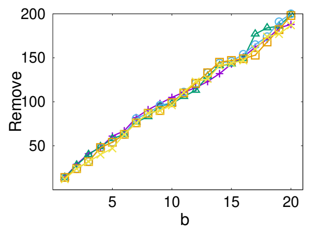
(n) DBLP $k = 30$



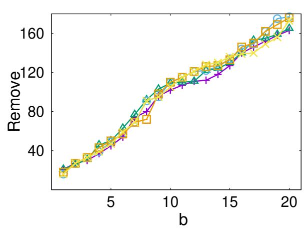
(o) DBLP $k = 40$



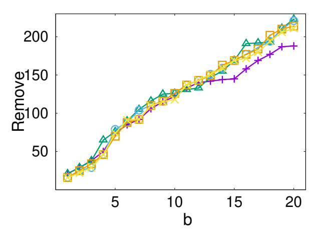
(p) DBLP $k = 50$



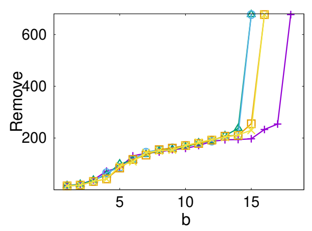
(q) Wiki-Talk $k = 110$



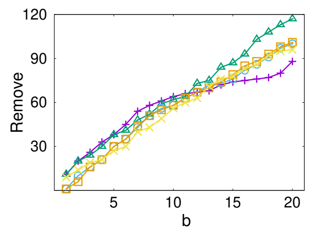
(r) Wiki-Talk $k = 120$



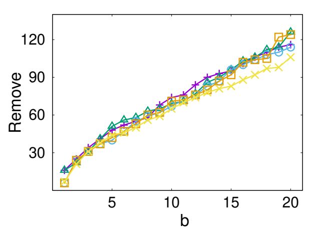
(s) Wiki-Talk $k = 130$



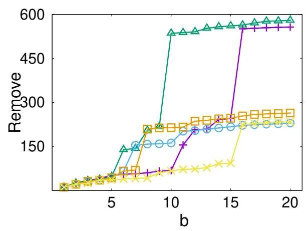
(t) Wiki-Talk $k = 140$



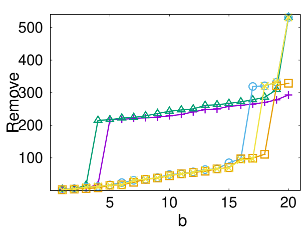
(u) Google+ $k = 90$



(v) Google+ $k = 100$



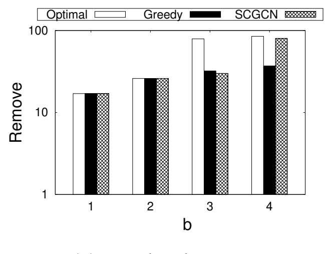
(w) Google+ $k = 110$



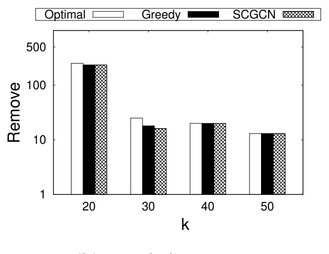
(x) Google+ $k = 120$

Fig. 6. Varying k

all of the three measurements evaluate node importance based on its neighborhood. The difference is that degree centrality counts the 1-hop neighbor while the PageRank and eigenvector centrality count the walks of infinite length. Compared with eigenvector centrality, PageRank imposes weights on the node neighborhood, i.e., the transition



(a) Facebook $k = 30$



(b) Brightkite $b = 2$

Fig. 7. Compare With the Optimal Solution

TABLE 5
The Statistics of *SCGCN* and *Greedy* Results on Facebook

Approach	$k = 20$		$k = 30$		$k = 40$		$k = 50$	
	mean	var	mean	var	mean	var	mean	var
SCGCN	0.37	0.03	0.53	0.03	0.61	0.01	0.58	0.01
Greedy	0.44	0.04	0.58	0.04	0.73	0.02	0.75	0.02

probabilities. Those make *PageRank* achieve more robust performance than *Degree* and *Eigencentrality* under most circumstances, especially for Fig. 6(b), Fig. 6(d), Fig. 6(n) and Fig. 6(j). The performance of the three baselines, as well as *Greedy* and *SCGCN* are comparable in some cases, e.g., Fig. 6(q), Fig. 6(r) and Fig. 6(v), in which we speculate these communities have relatively simple intrinsic structures, leading to a linear node collapsion regarding b. Thereby, these approaches may be able to find near-optimal solutions.

Empirical Gap to Optimal Solution. To further investigate the effectiveness of *SCGCN*, Fig. 7 compares the results of *SCGCN*, *Greedy* with the optimal solution on two relatively small networks by varying b on Facebook (Fig. 7(a)) and varying k on Brightkite (Fig. 7(b)). The optimal solutions are achieved by exhaustively search for all the combinations, where the time complexity is $O(n^b m)$. Here, m denotes the cost of computing the criticalness of a node set with b nodes and in the worst case, $m = O(|E|)$. Our C++ implementation of this exhaustive search takes 180 seconds for $b = 3$ and 69,000 seconds for $b = 4$ on the Facebook 30-core, respectively. When $b = 1$, the solutions of *Greedy* are exactly optimal, and *SCGCN* can also find the optimal solution for all the test cases (Fig. 6). As b increases, the removal gap between optimal and *Greedy* tends to become larger, which brings improvement space for *SCGCN*. As Fig. 7(a) shows, *SCGCN* achieves a near-optimal removal at $b = 4$ via an early found jumping point. Before this improvement, *SCGCN* tends to selects an alternative or even slightly worse solution than *Greedy*, as $b = 3$ in Fig. 7(a). That is a main reason that usually, *SCGCN* cannot outperform *Greedy* in selecting the first few critical uses. Note the potential function $f(u, U, G)$ of *Greedy* is proved to be non-submodular, which means the performance gap between *Greedy* and the optimum is already hard to analyze.

5.3 Variation on k

From Fig. 6, we can observe that for different cores, the pattern of the collapsing process is very different, even for the same graph. For cores of smaller k, there are no obvious jumping points in the collapsing process, as Fig. 6(a), Fig 6(m), Fig 6(e) and Fig. 6(u) shown. Here, the number of collapsed followers is growing stably as b increases, with considerable speed. In contrast, for cores of larger k, for instance, Fig. 6(b), Fig. 6(n), Fig. 6(g) and Fig. 6(w), collapsing is mainly caused by remarkable jumping points instead of local removing of one critical user. This phenomenon is consistent with our intuition, i.e., the denser the core, the harder to collapse it. In other words, it is more difficult for a single removal to break the structure of a dense k-core since it tends to form a stable structure, e.g., clique. To validate this intuition, we present the local clustering coefficient [23] distribution of Facebook cores in Fig. 8. The local clustering coefficient (CC), as defined in Eq. (10), computes the rate of the number of closed triangles to the number of triplets that a node forms, which quantifies how close its neighbors tend to form a clique.

$$\mathsf{CC}(u) = \frac{2|(v,k) : (v,k) \in E(G), v, k \in \mathsf{nbr}(u,G)|}{\deg(u,G)(\deg(u,G)-1)} \tag{10}$$

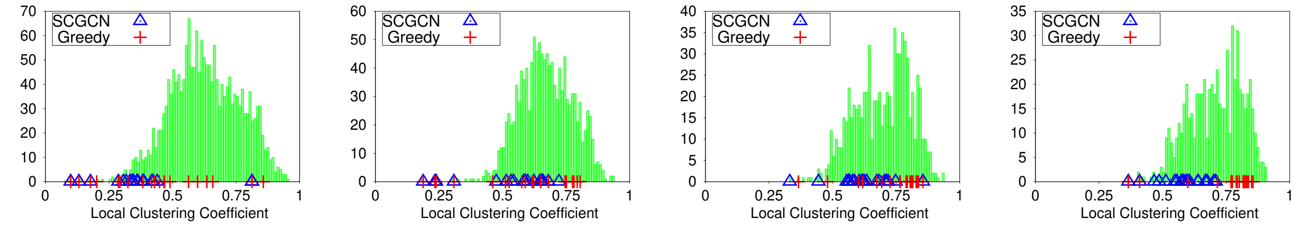
In Fig. 8, we divide the domain of CC, $[0, 1]$, to 50 segmentations and count the frequency of nodes falling in the segmentations. We can observe that from sparser to denser core, the histogram shifts from smaller CC to larger CC.

Moreover, we mark the CC of the critical users selected by our model *SCGCN* and *Greedy* in Fig. 8, respectively. Meanwhile, the mean (mean) and variance (var) of the CC are given in Table 5. It is worth mentioning that the solution of *SCGCN* has an overall smaller CC than that of *Greedy* on Facebook 30, 40, and 50 cores. Recall that smaller CC indicates the neighbors of the node have a weak ability to cluster together, so that removing the node would cause violation of the k-core to a greater extent.

5.4 Variation on Training Set Size

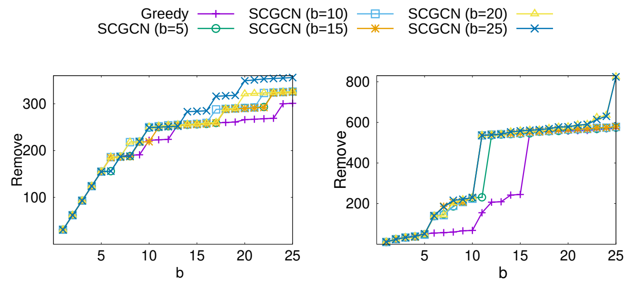
In this section, we investigate the generalization capability of models trained by node sets of different sizes. The larger node sets are sampled, the larger combinations the model is to learn. As the size of node set is uniformly sampled between $[1, b-1]$, we train 5 models by setting b to $\{5, 10, 15, 20, 25\}$, respectively and generating the solutions of 25 critical users. Except for the sampled node set size, the 5 models for one community are trained by the same hyper-parameters.

Fig. 9 shows the result of the 5 models on two communities, the 30-core of DBLP (Fig. 9(a)) and the 110-core of Google+ (Fig. 9(b)). According to Fig 9, it is encouraging to find that *SCGCN* has the generalization ability as b varies during training. In other words, the model trained by a give b can be reused to generate good solutions for larger b in the prediction phase. It implies that *SCGCN* learns the information from small node combinations and generalizes it to larger combinations. This generalization ability alleviates the combination complexity of the problem thus *SCGCN* can outperform *Greedy* for a larger budget b in prediction, even if the training budget is small. On the other hand, training by larger node sets is favorable to further improve the generalization ability, especially for solving a larger set of critical users. As shown in Fig 9(a), *SCGCN* ($b = 25$) achieves the best performance after $b = 13$ in prediction. For example, to remove 25 critical users, *SCGCN* ($b = 25$) collapses 356 users in total, while other models collapse 325 or 326 users.



(a) $k = 20$ (b) $k = 30$ (c) $k = 40$ (d) $k = 50$

Fig. 8. Facebook Local Clustering Coefficient Distribution



(a) DBLP $k = 30$ (b) Google+ $k = 110$

Fig. 9. Varying Training Set Size

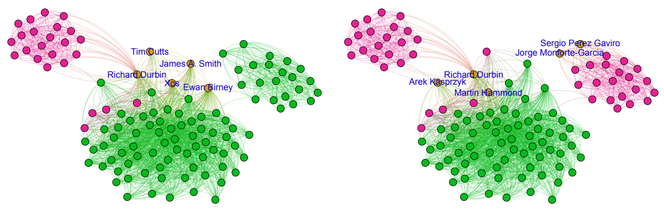
Another interesting observation is that in Fig. 9(b), both *SCGCN* $(b = 25)$ and *SCGCN* $(b = 20)$ detect a jumping point as the 25-th critical user, improving the number of collapsed users by about 44% compared with other models and *Greedy*. We conjecture that *SCGCN* $(b = 20)$ can detect the final jumping point due to the simple inherent structure in *Google+*.

5.5 A Case Study on DBLP

To demonstrate the effectiveness of *SCGCN* in social network analysis, we conduct a case study on the co-author network extracted from DBLP. In the co-author network, each node represents a researcher and the edge between them indicates that they have collaborated in more than 5 papers. Figure 10 depicts the followers of the critical users in the *Greedy* and our *SCGCN* approach. For a clear presentation, we just show one connected component of the whole network, and there are 119 researchers in the community.

When b is set as 1, both *Greedy* and *SCGCN* approach return "Richard Durbin", from the University of Cambridge. He has published many works in the area of bioinformatics and human genetics. The total number of citations is more than 170,000, and he has 20 followers. Notice that when b is 1, both *Greedy* and *SCGCN* find the optimal solution, as "Richard Durbin" has the most followers compared with all the other nodes.

Also, we set b as 5 and explore the critical users again. In this scenario, *Greedy* returns 5 nodes with a total of 29 followers, which covers 24.37% of the whole network. On the other hand, our *SCGCN* approach finds 5 nodes with 50 followers, and it covers nearly half of the whole community, with 72.41% more followers, compared with *Greedy*. The reason for the improvement is that, for *Greedy*, after detecting "Richard Durbin", the number of followers for each node in one iteration does not vary significantly. Thus the greedy algorithm selects the critical user from the node set with the same number of followers, and it cannot find



(a) *Greedy* (b) *SCGCN*

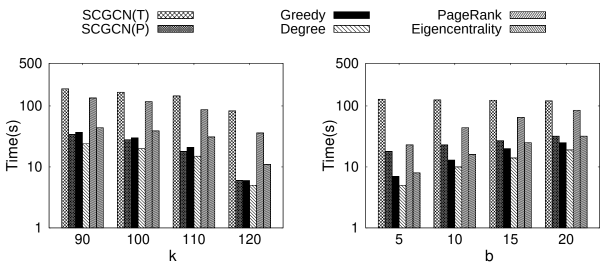
Fig. 10. The visualization of *Greedy* and *SCGCN* for DBLP dataset with $k = 20$ and $b = 5$. The yellow indicates the selected critical users U. The red and the green indicate corresponding removed followers and the remaining nodes, respectively.

the influence with multiple nodes as a combination. In our *SCGCN* approach, as we sample more node combinations, the results are significantly better than *Greedy*, and the critical users include "Richard Durbin","Arek Kasprzyk", etc. The critical users we found are representatives in the corresponding areas, and their papers have more than 212,911 citations. This result validates the potential of *SCGCN* in real-world applications.

5.6 Efficiency of Training and Prediction

Finally, we investigate the running time of the training and prediction of *SCGCN*, comparing with the 4 baseline algorithms. An *SCGCN* uses python *NetworkX* [24] to perform core collapsing in the prediction phase while we implement the 4 baseline algorithms by *NetworkX*. Fig. 11 shows the running time of *SCGCN* in training (*SCGCN* (T)) and prediction (*SCGCN* (P)), and that of the baselines, where the time of loading the graph and computing the initial k-core is excluded. For the training, the batch size is set to 32 and the number of steps is 2000, where we find *SCGCN* has already achieved its best performance over the graphs in Table 3. Due to the parallelized C++ binding for training data sampling and the GPU acceleration for model training, training can be finished in 200 seconds for the cores of Google+ and Wiki-Talk. In Fig. 11, the running time of the 4 baselines as well as the prediction of *SCGCN* are of the same magnitude, where *Degree* is the cheapest and *Eigencentrality* is the most expensive approach. With the acceptable training cost and the competitive prediction time, *SCGCN* is a promising option to find high-quality solutions when sufficient computation resources are available.

Moreover, an interesting result is that training by sampling on smaller budget size is not faster than that of larger budget size, as shown in Fig. 11(b). Consider the time complexity of sampling one partial solution S and computing the label (Eq. (2)), which is $O(|S|m_0 + nm_1)$, where m_0 and m_1 are two coefficients denoting the cost of computing



(a) Google+ $b = 20$ (b) Wiki-Talk $k = 120$

Fig. 11. Comparison on Running Time

the criticalness for one node on the original k-core $\mathsf{C_k}(G)$ and the partially collapsed k-core $\mathsf{C_k}(G_S)$, respectively. For one thing, as $n \gg |S|$, the term $O(nm_1)$ will dominate the complexity. For the other, the relationship between m_0 and m_1 is nondeterministic regarding different budget sizes $|S|$. Therefore, the training time for smaller b is not shorter but could be even longer than that for larger b.

6 Related Works

Critical Entity Exploration in Dense Subgraph: For the related works, [1] proposes the greedy algorithm to find critical users in social networks. The criticalness is based on the number of nodes removed according to the degree constraints. [2] proposes the measurement of the core structure based on the removal of edges, and [4] investigates the minimization of k-core with edge manipulation. [25] proposes a game theory based approach to find the critical edges. These works aim to find the critical structure of a dense subgraph via removing edges, but cannot apply to find the critical entities directly. [26] investigates the parameterized complexity for the collapsed k-core problem for different k and b values. It also proposes two algorithms for $k = 1$ and $k = 2$, respectively, but leaves the theory and algorithms for general cases as open problems.

For the dense subgraph, k-core is introduced in [27] and has been used in network analysis [28], event detection [29], keyword extraction [30], etc. These studies use the core structure to model the dense subgraph in the corresponding networks. Another variance based on k-core is anchored k-core problem [31]. The problem is to find b vertices, such that there exists k-core if these b vertices are not removed. This is different from the criticalness in our problem, as we aim to find the vertices with the most followers.

Neural Networks for Combinatorial Optimization: Recently, neural networks have been adopted to solve the combinatorial optimization problem. To solve the travel salesman problem (TSP), [32] uses reinforcement learning (RL) to train a Pointer Network [33] by learning from a set of problem instances and solutions. In addition, Dai et. al. [34] propose a framework to learn a greedy algorithm over graphs, in solving minimum vertex cover (MVC), maximum cut and TSP. This approach learns a graph representation and an evaluation function as a greedy heuristic, which is trained by deep Q-learning [35] jointly. [36] builds a GCN based classifier with a collection of graphs and solutions to solve MVC, maximal independent set and maximal clique. To reduct the search space of combinatorial optimization algorithms, [37] trains a binary classifier to predict the probability that a node is in the maximum clique, by leveraging node-level features. However, these approaches cannot be directly applied to solve our problem, detecting critical entities over a single large real-world graph. They focus on building a general model by learning from a distribution of graphs and corresponding solutions in an inductive way. In contrast, our model, *SCGCN* learns over a target domain, i.e., a given graph, via graph representation learning.

For machine learning-based solutions, the first and foremost step is graph representation learning. A recent survey introduces extensive studies [38], [39], including early low-rank matrix decomposition approaches [40], encoding the neighborhood relationship by truncated random walks [8], [9], and preserving high-order proximity and macroscopic structures [41], [42]. Graph convolutional networks (GCN) defines convolution filters on graph Laplacian [10], which has desirable properties as local translational invariance, permutation invariance and size independence. [11] proposes a localized first-order approximation of spectral graph convolution. [43] reviews a large pool of emerging GCN variations.

While we rely on existing deep learning paradigms, our main differences are: (1) we use GCN and attention mechanism to model the combination structures among potential critical entities, in solving the NP-hard problem. (2) we consider both the graph representation from the learning procedure, as well as the deterministic algorithm procedure to explore critical entities.

7 Conclusions

In this paper, we focus on detecting critical entities in social network communities, in a novel, machine learning-based perspective. As it is an NP-hard problem, we first analyze the deficiency of the existing greedy algorithm in depth. Distinguished from the pre-defined heuristics, we design a new model: *SCGCN* to learn the underlying combinatorial criticalness over nodes sets, which are possible partial solutions generated by sampling on the communities. Furthermore, considering the graph structure, we use a deterministic pruning strategy to reduce the sampling space as well as the model inference space. Compare with the greedy algorithm, our learning-based approach improves the quality of the solution significantly, with acceptable training cost and competitive prediction cost. As a brand new attempt to solve the NP-hard problem by learning approach, our study also reveals that neural networks have the ability to deal with combinational optimization problem by encoding hidden features of the combination structures.

Acknowledgement

This work is supported by the Research Grants Council of Hong Kong, China under No. 14203618, No. 14202919 and No. 14205520. Zhiwei Zhang is supported by National Key R&D Program of China (Grant No. 2020YFB1707902), NSFC (Grant No. 62072035) and Open Research Projects of Zhejiang Lab (Grant No. 2020KE0AB04).

References

[1] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Finding critical users for social network engagement: The collapsed k-core problem," in *Proc. of AAAI'17*.

[2] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan, "Measuring and improving the core resilience of networks," in *Proc. of WWW'18*, 2018.

[3] F. Zhang, L. Yuan, Y. Zhang, L. Qin, X. Lin, and A. Zhou, "Discovering strong communities with user engagement and tie strength," in *Proc. of DASFAA'18*, 2018.

[4] W. Zhu, C. Chen, X. Wang, and X. Lin, "K-core minimization: An edge manipulation approach," in *Proc. of CIKM'18*.

[5] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[6] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. of NIPS'17*, 2017.

[7] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[8] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proc. of KDD'14*, 2014.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. of KDD'16*, 2016.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. of ICLR'14*, 2014.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. of ICLR'17*, 2017.

[12] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *Proc. of WWW'19*, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. of NIPS'17*, 2017.

[14] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. of AAAI'18*, 2018.

[15] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[16] "KONECT (the koblenz network collection)," http://konect.uni-koblenz.de.

[17] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: http://networkrepository.com

[18] "Pytorch," https://github.com/pytorch/pytorch.

[19] "pybind11-Seamless operability between C++11 and Python," https://github.com/pybind/pybind11.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of ICLR'15*, 2015.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[22] P. Bonacich, "Power and centrality: A family of measures," *American journal of sociology*, vol. 92, no. 5, pp. 1170–1182, 1987.

[23] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[24] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Tech. Rep., 2008.

[25] S. Medya, T. Ma, A. Silva, and A. K. Singh, "K-core minimization: A game theoretic approach," *CoRR*, vol. abs/1901.02166, 2019.

[26] J. Luo, H. Molter, and O. Suchý, "A parameterized complexity view on collapsing k-cores," in *Proc. of IPEC'18*, 2018.

[27] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269 – 287, 1983.

[28] A. Adiga and A. K. S. Vullikanti, "How robust is the core of a network?" in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 541–556.

[29] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis, "Degeneracy-based real-time sub-event detection in twitter stream," 2015.

[30] A. J.-P. Tixier, F. D. Malliaros, and M. Vazirgiannis, "A graph degeneracy-based approach to keyword extraction," in *EMNLP*, 2016.

[31] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, "Preventing unraveling in social networks: The anchored k-core problem," in *Automata, Languages, and Programming*, A. Czumaj, K. Mehlhorn, A. Pitts, and R. Wattenhofer, Eds., 2012, pp. 440–451.

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

[33] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. of NIPS'15*, 2015.

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

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[36] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. of NIPS'18*.

[37] J. Lauri and S. Dutta, "Fine-grained search space classification for hard enumeration variants of subset problems," in *Proc. of AAAI'19*.

[38] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, 2017.

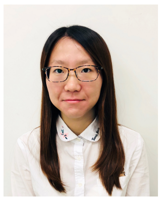
[39] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. on Knowledge and Data Engineering*, 2018.

[40] T. M. V. Le and H. W. Lauw, "Probabilistic latent document network embedding," in *Proc. of ICDM'14*.

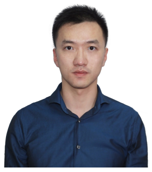
[41] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: large-scale information network embedding," in *Proc. of WWW'15*, 2015.

[42] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proc. of AAAI'17*, 2017.

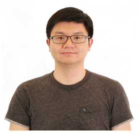
[43] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *CoRR*, vol. abs/1901.00596, 2019.



Kangfei Zhao is a Postdoc Fellow in Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. She obtained Ph.D. in The Chinese University of Hong Kong, under the supervision of Prof. Jeffrey Xu Yu in 2019 and received her B.E. degree in University of Science and Technology of China in 2014. Her research interests focus on graph analytics/processing systems, graph data management, in-database machine learning.



Zhiwei Zhang received the B.E. degree in computer science and technology from Renmin University of China in 2010, and the Ph.D. degree in systems engineering and engineering management from the Chinese University of Hong Kong in 2014. He is currently a professor at Beijing Institute of Technology, and was a research assistant professor at Hong Kong Baptist University from 2015 to 2019. His research interests include graph algorithms, distributed systems and blockchain.



Yu Rong is a Senior researcher of Machine Learning Center in Tencent AI Lab. He obtained the Ph.D. degree from The Chinese University of Hong Kong in 2016 and joined Tencent AI Lab in June 2017. His main research interests include social network analysis, deep graph learning, and large-scale graph systems. He has published several papers on data mining, machine learning top conferences KDD, WWW, NIPS, CVPR, ICCV. He has served as a reviewer for KDD, WWW, CIKM, WSDM, SDM and other journals such as VLDBJ and TKDE.

Jeffrey Xu Yu held teaching positions in the Institute of Information Sciences and Electronics, University of Tsukuba, Japan, and the Department of Computer Science, Australian National University, Australia. Currently, he is a Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong, Hong Kong.

Junzhou Huang is an Associate Professor in the Computer Science and Engineering department at the University of Texas at Arlington. He received the B.E. degree from Huazhong University of Science and Technology, Wuhan, China, the M.S. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, and the Ph.D. degree in Computer Science at Rutgers, The State University of New Jersey. His major research interests include machine learning, computer vision and imaging informatics.