ICS-GNN: Lightweight Interactive Community Search via Graph Neural Network

Jun Gao
Key Laboratory of High Confidence Software Technologies, CS, Peking University, China
gaojun@pku.edu.cn

Jiazun Chen
Key Laboratory of High Confidence Software Technologies, CS, Peking University, China
chen_jiazun@foxmail.com

Zhao Li
Alibaba Group, Hangzhou, China
lizhao.lz@alibaba-inc.com

Ji Zhang
Zhejiang Lab, China
ji.zhang@zhejianglab.com

ABSTRACT

Searching a community containing a given query vertex in an online social network enjoys wide applications like recommendation, team organization, etc. When applied to real-life networks, the existing approaches face two major limitations. First, they usually take two steps, *i.e.,* crawling a large part of the network first and then finding the community next, but the entire network is usually too big and most of the data are not interesting to end users. Second, the existing methods utilize hand-crafted rules to measure community membership, while it is very difficult to define effective rules as the communities are flexible for different query vertices. In this paper, we propose an Interactive Community Search method based on Graph Neural Network (shortened by ICS-GNN) to locate the target community over a subgraph collected on the fly from an online network. Specifically, we recast the community membership problem as a vertex classification problem using GNN, which captures similarities between the graph vertices and the query vertex by combining content and structural features seamlessly and flexibly under the guide of users' labeling. We then introduce a k-sized Maximum-GNN-scores (shortened by *kMG*) community to describe the target community. We next discover the target community iteratively and interactively. In each iteration, we build a candidate subgraph using the crawled pages with the guide of the query vertex and labeled vertices, infer the vertex scores with a GNN model trained on the subgraph, and discover the *kMG* community which will be evaluated by end users to acquire more feedback. Besides, two optimization strategies are proposed to combine ranking loss into the GNN model and search more space in the target community location. We conduct the experiments in both offline and online real-life data sets, and demonstrate that ICS-GNN can produce effective communities with low overhead in communication, computation, and user labeling.

PVLDB Reference Format:
Jun Gao, Jiazun Chen, Zhao Li, and Ji Zhang. ICS-GNN: Lightweight Interactive Community Search via Graph Neural Network. PVLDB, 14(6): 1006 - 1018, 2021.
doi:10.14778/3447689.3447704

Proceedings of the VLDB Endowment, Vol. 14, No. 6 ISSN 2150-8097.
doi:10.14778/3447689.3447704

1 INTRODUCTION

Community search [4] aims to find communities containing the given query vertex, and the discovered communities can be used as an effective candidate set in the applications such as item/friend recommendations, fraudulent group discovery, etc. Although the problem is well studied, there are no widely accepted definitions of community in the literature. Most of researchers assume that the vertices in the community share structural and content similarities. They propose community models such as k-core [20], k-truss [9], k-clique [2], etc, from the viewpoint of structural constraints, or attributed community queries [3, 10] which attempt to combine the structural and content features.

Although significant progress is achieved, the current methods still face challenges when applied to real-life social networks. First, nearly all these methods assume that the data have already been crawled, and they only perform analysis on those collected data. However, we cannot separate the data crawling and community search clearly. As there are massive active accounts and messages each day in the online network, the crawler will find a large number of irrelevant pages as a result if the collecting policy is not controlled, which incurs high and unnecessary resource consumptions in storage, network transfer, and computation in the following community search. The focused search methods [16, 18] are alternative choices. But they are too coarse for the community search as they guide crawling using a classifier trained only on local features such as the content of Web pages.

Second, the community search is flexible in nature, and it is nearly impossible to produce high-quality community directly using the predefined community rules. Also, it is not trivial for existing methods to refine communities incrementally to achieve the goal of community search. Compared with community detection which finds all communities in a graph sharing general patterns [4], community search relies on the given query vertex to locate the communities with specific patterns. Some communities have the dense structural relationships, which can be captured by the existing community search models [2, 9, 20], but it is challenging to locate communities with weak structural relationships and high content similarities. For example, users in the same company may roughly take a hierarchical form in an online network with relatively sparse structural relationships but similar users' content features. In addition, it incurs heavy burdens if existing rule-based

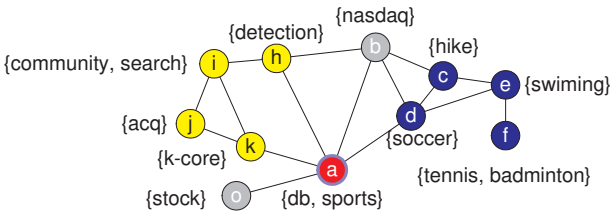


Figure 1: Sample Graph.

methods are employed to find the community progressively. Generally, based on each result presented, users may need to adjust k in the structural constraints [2, 9, 20], select representative attributes, and balance the weights between content and structural features [3, 10], as there is usually tension between the content and structural requirements on communities. However, the number of candidate communities changes significantly even if k is increased or decreased by 1. The rule adjustment poses more challenges if we consider massive content keywords and their complex relationships.

We illustrate these challenges using a toy graph example in Figure 1. The keywords annotated on vertices represent their content. Suppose a is the query vertex. The vertices belong to the same community share the same color. We can see that there are two major communities around a, one for research in yellow, and the other one for sports in blue. However, neither of them can be easily captured by the current structure or keyword based communities. Take the research community (with vertices a, h, i, j, k) as an example. It cannot meet k-core or k-truss requirement when k is larger than 2. The sparsity of relationships may be due to the initial subgraph collected by an incremental crawler. In addition, we observe that there are no shared keywords on different vertices in the research community. For example, the keywords “k-core" and “acq" are totally different, although both are closely related to the community search.

This paper attempts to address the challenges above from the viewpoint of ICS *(interactive community search)*. In order to avoid the unnecessary cost in crawling a large part of an online network, we allow the Web crawling and community search to be interleaved, which are progressed iteratively, and guided by the feedback to end users. We expect that the target community becomes increasingly more clear than it in the previous iterations with more feedback. Still take Figure 1 as an example. If an end user selects vertex k as a member in the target community, the crawler will extend the search from k, and enrich the candidate subgraph from the underlying graph. In addition, rather than using a fixed rule to combine the structural and content features, we choose machine learning methods to determine which vertices should be in the target community. In the example, the community membership rules are learned from the data to combine the content and structural features, by using the query vertex a and the vertex k as positive instances, and vertex d as a negative instance in a classification problem,

We argue that our method is lightweight thanks to the following reasons. First, the Web crawler only acquires the potentially useful data from the Web, which lowers the cost in communication as well as in the following community search. Second, end users only label whether vertices should be in the community or not, which is much easier than adjusting parameters in rules. Third, the method can leverage the pre-trained keyword representations which have already captured relationships from massive data.

To the best of our knowledge, we are the first to study the problem of ICS. The method proposed combines techniques from database, WWW, and machine learning fields. Community search is a well-studied problem in the database area. The interactive crawling for communities can be viewed as a significant extension of focused search in an online social network. The machine learning methods, particularly GNN (Graph Neural Network) [7, 24], can provide a unified way to combine the structure and the content features. The contributions of this paper are summarized as follows:

(1) We model the community as a k-sized subgraph with maximum GNN scores (shortened by the *kMG* community), which combines content and structural features by leveraging a GNN model, and incorporates users' feedback easily. We show that the location of *kMG* community is NP-hard. (Section 2)
(2) We design an approach, called ICS-GNN, to search community iteratively. In each iteration, the candidate subgraph is built with BFS (breadth-first-search) and a partial edge enhancement strategy, a GNN model is trained on the subgraph to infer the vertices' probabilities in the community, and a *kMG* community is located by a vertex-swapping based method. (Section 3)
(3) We further propose optimization strategies of ICS-GNN including, i) incorporation of a ranking based loss function into the GNN model to simplify the labeling tasks, ii) a greedy method based on global relative benefits to locate the *kMG* community to fit various data distributions. (Section 4)
(4) We conduct extensive experiments on offline (with ground-truth communities) and online datasets to show the effectiveness and efficiency of ICS-GNN. (Section 5)

The remainder of this paper is organized as follows. We review preliminaries and formulate the problem in Section 2. Then, we describe the basic ICS-GNN method in Section 3, and devise its optimization strategies in Section 4. Section 5 reports experimental results in the offline and online setting. We review related works in Section 6, and conclude the paper in Section 7.

2 PRELIMINARIES AND PROBLEM FORMULATION

In this section, we first review crawling operations over a social network and its data model, and introduce graph neural network. Then, we give our community definition, and formulate the problem in this paper.

2.1 Open Social Network

This paper aims at the community search in an open social network, in which the messages posted by u can be accessed by others, and therefore can be processed by applications. In addition, the popular platforms, like Sina Weibo [1], Twitter, [2] etc, facilitate third-party applications to visit users' data if allowed. The basic APIs used in

[1] http://open.weibo.com/wiki/
[2] https://apiwiki.twitter.com/

our paper can be summarized in Table 1. Note that the messages and the relationships of a user u are organized in multiple pages, and then the page index is needed in APIs.

Table 1: Web APIs in Social Network

API	Meanings
GetProfile(u)	get the profile of user u.
GetPost(u, i)	get messages posted by u in the i-th page.
GetFollower(u, i)	get followers of u in the i-th page.
GetFan(u, i)	get fans of u in the i-th page.

Note that APIs in Table 1 are abstract. We can find similar APIs in different social networks. Even if the social networks do not provide these functions, we can achieve similar results by accessing the Web page first and extracting the target items next. Without loss of generality, we can use these operation APIs, and build a subgraph from the underlying social network on-the-fly.

For an online social network G, it can be modeled as $G = (V, E, F)$. The vertex set V represents users in the network. The edge set E represents the relationships between users. The structure of G can be represented by an adjacent matrix $A = \{0, 1\}^{|V|\times|V|}$, where $A[i, j]$ indicates whether the i-th vertex and the j-th vertex is connected ($A[i, j] = 1$) or not ($A[i, j] = 0$). F is the content features of vertices. $F = \mathbb{R}^{|V|\times d}$ is a content feature matrix, where $F[i]$ is a d-sized feature vector for the i-th vertex.

2.2 Graph Neural Network

GNN [7, 24] is proposed to learn high-dimensional representation (embedding) of vertices by capturing both content features and structure relationships. GNN achieves this goal via encoding content and structural features into a function, and optimizes the function with the guide of (supervised or un-supervised) training signals. Different from the image data where a pixel has a fixed number of neighbors with fixed positions, a vertex's neighbors and their positions in a general graph are arbitrary. Such features pose high challenges in designing functions in the neural network.

From the viewpoint of a vertex u with its features $F(u)$ and its hidden embedding $H(u)$, GNN involves two major functions namely the aggregate function $f(.)$ and the update function $g(.)$ [24]. The aggregate function $f(.)$ collects embeddings from neighbors with different weights. Since the number of the neighbors is flexible, and there is no fixed order in neighbors, $f(.)$ is required to be a permutation invariant. That is, the results of $f(.)$ should be the same even if the neighbors are processed in different orders. Usually, the aggregation functions, like *sum*, *mean*, can be used in the message collection. The update function $g(.)$ transforms the current embedding into a new one.

Different GNN variants have been proposed, with their differences in assigning the weights of the neighbors and aggregating information from the neighbors. For example, GCN [12] is a simplified version of the spectral based GNN model, and assigns the weights of neighbors related to their degrees. GraphSage [6] performs a sampling operation over the neighbors. GAT [21] uses an attention mechanism in determining the weights of neighbors.

GNN model works as follows. The embeddings for vertices are initialized by $H^0 = g(F, W_0)$ on the content features, where W_0 are parameters in functions. Then, the embeddings are updated from those of neighbors recursively. The embeddings H^{i+1} in the $(i+1)$-th layer can be computed as $H^{i+1} = g(H^i, f(A, H^i, W_1^i), W_0^i)$, where A is the adjacent matrix, and W_0^i and W_1^i are parameters. Under the supervised setting, the loss function measures the differences between the labeled results and the predicted ones. The parameters in the model are learned with strategies like the gradient-descent to minimize the loss.

This paper recasts the community membership as a supervised classification problem [1]. End users label vertices in the target community (including the query vertex) with 1, and label the vertices which are not in the community with 0. The labeling operations on vertices can be performed iteratively on the currently-discovered community. The model can be abstracted by $P = GNN(A, F, W)$, where A, F and W are for adjacent matrix, features, and learnable parameters, respectively. $P = \mathbb{R}^{|V|}$, and $P[i]$ stands for the membership probability of the i-th vertex. We also use $P[u]$ for the GNN score of the vertex u. Obviously, the larger $P[u]$ is, the higher chance that u is in the community. The cross-entropy loss function is used to measure the probability distribution differences between the labeled and the predicted results. The graph $G = (V, E, F)$ can be enriched into $G = (V, E, F, P)$ with the inferred GNN scores on the vertices obtained upon the convergence of GNN training.

2.3 k-Sized Community of Maximum GNN Scores

We give our definition of the target community in the context of the interactive search. Based on the enriched subgraph with GNN scores $G = (V, E, F, P)$, we define the community as follows:

Definition 1. ***k-Sized Community with Maximum GNN Scores.*** *Let $G = (V, E, F, P)$, $q \in V$ be the query vertex, and k be the community size. The k-sized community with maximum GNN scores, called kMG community, is an induced subgraph $G_c = (V_c, E_c, F_c, P_c)$ of G with the following requirements:*

(1) *The query vertex $q \in V_c$, and G_c is connected;*
(2) *$|V_c| = k$;*
(3) *The sum of GNN scores $\sum_{u\in V_c} P[u]$ is maximum.*

In the first condition, we require that the query vertex is in the target community, as we focus on community search rather than community detection. In addition, we make a relaxation in the structural relationships of the target community. Instead of posing rigid structural constraints like k-core [20], k-clique [2], we only require that the community G_c is connected. The relaxation is due to the following reasons: i) The initial subgraph in the interactive community search may not be dense due to the controlled crawling policy; ii) In some cases, there are no densely connected relationships inside a real-life community even if the data is fully collected.

We enforce a restriction on the size of the community in the second condition. It is easy for end users to adjust k according to the quality of the discovered results. In addition, some applications require a restriction on the community size. For example, there are at most k persons in an organized team, and there are at most k vertices for clear visualization in tools. Note that the previous

definitions cannot control the size of community incrementally and precisely.

The third condition is about the similarities of the vertices inside the community. Under the guide of labeled vertices, the GNN model learns the rules to infer the vertex membership probabilities in a community using the content and structural features, capturing the similarities between vertices and the positive vertices (with label 1).

Note that the discovered community can be refined progressively. When an end user is not satisfied with the currently-discovered community, he can perform labeling to tell which vertices should (not) be in the target community. Then, the crawler will search around the labeled positive vertices to enhance the candidate subgraph, and the GNN model is re-trained over the new candidate subgraph. With more feedback and more features, end users have more chances to get the desired target community.

We show that the decision version of location of *kMG* community is NP-hard, that is, to check whether there exists a k-sized subgraph with its sum of GNN scores larger than a threshold. First, it takes a polynomial time to verify the discovered results. Second, we make a reduction from an NP-hard problem, the Knapsack problem [11], to the decision problem of *kMG*. A Knapsack problem is selecting a sub-set of items $I' \subseteq I$ under the budget b and the sum of $i_v (i \in I')$ larger than t, given an item set I with each item $i \in I$ having its cost i_c (an integer) and value i_v. For a Knapsack problem, we construct a tree T as follows. For each $i \in I$, we build an i_c-length path from the root of T to a leaf vertex l_v, with GNN score i_v on l_v and 0 on other vertices in the path. The root vertex is selected as the query vertex. Obviously, an item sub-set $I' \subseteq I$ containing k items and the sum of values larger than t exists iff there exists a $(k+1)MG$ community (including the query vertex) in T with the sum of GNN scores larger than t.

2.4 Problem Formulation

We study the interactive community search from an online social network. Given a query vertex q, we construct a candidate subgraph containing q, train a GNN model to infer probabilities of vertices in the community, and locate the *kMG* community in the subgraph. Such a process repeats with the incorporation of the user's feedback. We expect that the proposed method can achieve higher effectiveness and efficiency, while lower human labeling burden.

The effectiveness means that the vertices in the discovered community are meaningful to the end users. We perform the case studies in the online network. Besides, we conduct tests in the offline networks. The effectiveness can be measured against the ground-truth communities or some simulated communities based on vertices' labels, which is discussed in the experimental part.

The efficiency measures the time cost of the method. The interactive community search involves different phases in multiple rounds. The Web crawling task is mainly related with the network transfer speed. The labeling task on the currently-discovered community may wait for human operations. We then focus on the efficiency of the GNN model training and the location of *kMG* community.

The labeling cost is another important factor. Supervised GNN models need the labeled data, while labeling task is a burden to end users. Then we expect to lower the number of labeled instances. The way of labeling is another concern. For example, end user may take more time to determine explicit labels, but may find it easier to rank two instances in terms of their membership of community. Thus, we need to incorporate different kinds of feedback in the same framework.

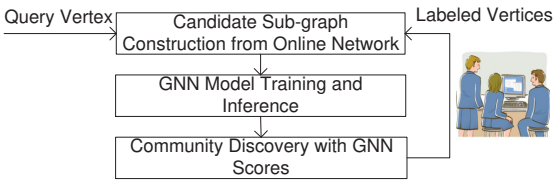


Figure 2: The Framework of ICS-GNN

3 ICS-GNN: GNN BASED INTERACTIVE COMMUNITY SEARCH

In this section, we first illustrate the framework of ICS-GNN, and then focus on different components of ICS-GNN. Finally, we perform the analysis of the algorithm.

3.1 Framework

We illustrate the basic framework of ICS-GNN in Figure 2. It involves multiple rounds of the community search. In each round, ICS-GNN attempts to locate a candidate subgraph using a query vertex and positively labeled vertices from a large online social network. The challenge here is to collect candidate vertices, as well as to provide sufficient relationships among them. The next component is to build a GNN model on the candidate subgraph. The messages posted by the users are transformed into the content features of the vertices. The label of a vertex u indicates whether u should be in the community or not. GNN model is first trained on the candidate subgraph, and then is used to infer the probabilities for all vertices in the subgraph. The final step is to locate the *kMG* community based on the GNN scores on vertices, which requires approximate methods due to the problem complexity. End users may perform labeling over the located community, which will guide the crawling and model training in the next round. The symbols frequently used in this paper are listed in Table 2.

Table 2: Frequently Used Symbols and Meanings

Symbols	Meanings
$G = (V, E, F)$	Underlying graph.
$G_s = (V_s, E_s, F_s, P_s)$	Candidate subgraph with features and inferred GNN scores.
$G_c = (V_c, E_c, P_c)$	Located community.
q	The query vertex.
S, S_p	S for labeled vertices, $S_p \subseteq S$ for positive vertices (including the query vertex).
$u[i_c], u[i_e], u[i_p]$	Vertex u's index of pages for messages, edges, and partial edges.
l_c, l_e, l_p	# pages for messages, edges and partial edges in each crawling.

3.2 Candidate Sub-Graph Construction

The candidate subgraph construction is to locate the potentially useful vertices and their relationships from a large social network, which can be implemented by crawling around the query vertex and the labeled positive vertices rather than expanding from them into deep levels. The candidate subgraph can be viewed as an initial coarse community.

We discuss design choices in the candidate subgraph construction. The first issue is the crawling strategy. It is straightforward to construct the initial subgraph using 1-hop neighbors from positively labeled vertices (including the query vertex) by BFS as they are more likely to be included in the community than distant vertices. We should note that the initial subgraph constructed can only form a tree-like graph with limited structural relationships. However, if we allow further searching from 1-hop neighbors, we will encounter exponentially increased number of vertices, and most of them may be unrelated with the target community. In order to handle this issue, we take a partial edge enhancement strategy for the 1-hop neighbors. That is, we still allow BFS searching from 1-hop neighbors but only include their edges with existing vertices, and exclude the newly encountered vertices from the subgraph.

Algorithm 1: Candidate Sub-Graph Construction

Input: Labeled positive vertices S_p, previous candidate subgraph G'_s.
Output: Candidate subgraph $G_s = (V_s, E_s, F_s)$.

1 Initialize $G_s = (V_s, E_s, F_s)$ by loading G'_s from disk;
2 **for** *vertex u in* V_s **do**
3 **if** $u \in S_p$ **then**
4 **for** *i from* $u[i_e]$ *to* $u[i_e] + l_e$ **do**
5 Invoke *getFollows(u, i)*;
6 Extract newly-visited vertices and edges, and add them into V_s and E_s respectively;
7 $u[i_e] \leftarrow u[i_e] + l_e$;
8 **if** $u \notin S_p$ **then**
9 **for** *i from* $u[i_p]$ *to* $u[i_p] + l_p$ **do**
10 Invoke *getFollows(u, i)*;
11 Extract edges from u to V_s, and add edges to E_s;
12 $u[i_p] \leftarrow u[i_p] + l_p$;
13 **for** *i from* $u[i_c]$ *to* $u[i_c] + l_c$ **do**
14 Invoke *getPost(u, i)*;
15 Add messages to features $F[u]$;
16 Invoke *getProfile(u)* if not done;
17 Save $G_s = (V_s, E_s, F_s)$ into the disk.

Second, we need to record the status of vertices to support the incremental crawling strategy. The vertices in online social networks have a various number of neighbors. It is not feasible to reach all neighbors in one time crawling when the neighbors are massive. Things are similar when the messages posted by a user occupy many pages. We then make a restriction on the number of pages scanned in each crawling. That is, we introduce $u[i_c], u[i_e], u[i_p]$ to record the indices of starting pages in finding messages, edges, and partial edges, respectively. In each crawling, we scan the predefined l_c, l_e, l_p pages for these purposes correspondingly. These starting indices are then updated and stored into the disk (file or database). Thus, a limited number of Web pages are scanned and processed each time the crawler is invoked. Such a crawling strategy can also be viewed as a sampling from the large underlying graph, which plays an important role in reducing the training cost in GNN while preserving its effectiveness [6].

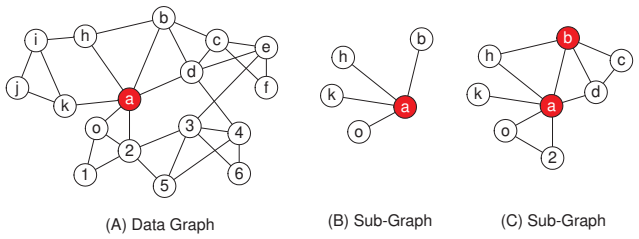


Figure 3: Candidate Sub-Graph Construction

We present Algorithm 1 to build the candidate subgraph. The candidate subgraph is loaded for the incremental update in Line 1. We distinguish the labeled positive vertices from others as they perform different operations. For each labeled vertex u, we locate l_e pages from the starting page indexed by $u[i_e]$ to find new vertices and edges in Line 4 to 7. Note that we only invoke the function *getFollow* while *getFan* is omitted due to the space. For each unlabeled vertex u, only the partial edges between u and the existing vertices are added into the graph. The incremental crawling for the content pages for all vertices are performed from Line 13 to 15.

We illustrate a running example in Figure 3. Given a data graph in Figure 3(a) and the query vertex a, we take a BFS from a, and build the candidate subgraph in Figure 3(b). Suppose at that time, end users make a positive label on vertex b. In the next round of crawling, we continue the BFS from a with the stored status, and start a new BFS from b, during which c is encountered and added. Note that we also start a new BFS from the non-labeled vertices, such as vertex 2. However, we only build an edge between vertex 2 and existing vertices, such as o, but do not add newly encountered vertices, like vertex 5, into the subgraph.

3.3 GNN Model Training and Inference on the Candidate Subgraph

Based on the candidate subgraph constructed, we build a GNN model to measure the probabilities of vertices belonging to the community. Before doing that, we need to build the content features of vertices and choose a loss function in the GNN model.

We first convert messages with varied-length for different vertices into fixed-length features, and at the same time handle the issue of different keywords having similar meanings. Let u be a vertex in the candidate subgraph. $F(u)$ contains the messages posted by u, and for each message $m \in F(u)$, m contains multiple keywords. To build the content feature of u requires the representation of each keyword and the combination of all representations. One way is to learn the keyword representations directly on the collected messages using the model like word2vec [13, 17]. However, learning from scratch may not yield high-quality representations due to insufficient training instances. In this paper, we locate the

keyword representation from pre-trained embeddings, which have been learned from massive data sets and can measure the relationships between different keywords precisely.

Inspired by a simple but efficient method used by GraphSage on Reddit [6], we aggregate the content features of u by averaging $emb(t)$ for each keyword t in all messages posted by u, where $emb(t)$ is the embedding in a pre-trained embedding set. We can take other sophisticated aggregation functions, but they are not the focus of this paper. The profile information of u, such as the name, location, and age, can also be concatenated with the content features to produce rich content features.

The community search needs to measure the probabilities of all vertices in the subgraph, which can be modeled as a 0/1 classification problem. With the labeled vertices, the GNN model is trained on the subgraph to yield P, a $|V_s|$-sized vector for community membership probabilities of all vertices. Let $P[u]$ indicate the probability of a labeled vertex u. We use the cross-entropy as the loss function of GNN (Equation 1), where $u.y$ is for its labeled result. To minimize the loss, we need to update the hidden embedding of u and propagate the updates to its neighbors, which then impacts the prediction results of other vertices.

$$Loss_l = \sum_{u \in S} -u.y * \log(P[u]) - (1 - u.y) * \log(1 - P[u]) \quad (1)$$

Algorithm 2 illustrates the GNN model training and GNN score inference. We preprocess the messages and convert them into the fix-sized vectors. We then build the model with an adjacent matrix A_s, features F_s and parameters W. We compute the loss function in Equation 1 using all labeled (positive and negative) vertices. When the training converges, we use the trained model to infer GNN scores on other vertices, which serve as a basis for locating the *kMG* community.

Algorithm 2: GNN Model Training and Inference

Input: Candidate subgraph $G_s = (V_s, E_s, F_s)$, labeled vertex set S, pre-trained embedding set D.
Output: Candidate subgraph with GNN Scores $G_s = (V_s, E_s, F_s, P_s)$.

1 **for** *vertex u in* V_s **do**
2 Build $u's$ content features from messages using D;
3 Build a GNN mode $\mathcal{M}$ with $P_s = GNN(A_s, F_s, W)$ using the loss function in Equation 1;
4 Train $\mathcal{M}$ until convergence;
5 Apply $\mathcal{M}$ to infer GNN scores P_s on vertices, and obtain $G_s = (V_s, E_s, F_s, P_s)$.

We take an incremental crawling strategy to build candidate subgraphs, but train the model fully on each candidate subgraph. This is because that by controlling the size of the candidate subgraph, the time cost in model training is not the bottleneck, compared with the time required by human labeling and the page crawling. In addition, the model trained fully usually has better performance to support community search than the model trained incrementally. Due to the same reason, we do not consider the inductive models such as GraphSage [6], although they are capable of handling the newly added vertices.

3.4 Search for K-Sized Community with Maximum GNN scores

Due to the problem complexity, we design approximate methods to find the *kMG* community from the subgraph graph with GNN scores. The community certainly contains the query vertex. Intuitively, the vertices near the query vertex have more chances to be included in the community. The community then can be initialized by a BFS from the query vertex until k vertices are encountered. In the following, we swap the vertices with the lower scores inside the current community for the vertices with higher scores outside, while at the same time preserving the connectivity of the community.

Algorithm 3: Search for *kMG* Community

Input: Candidate subgraph $G_s = (V_s, E_s, F_s, P_s)$, query vertex q.
Output: Community $G_c = (V_c, E_c, P_c)$.

1 $u[test] \leftarrow$ *False* for each vertex $u \in V_s$;
2 Initialize $G_c = (V_c, E_c, P_c)$ with an empty graph;
3 **for** *vertex u encountered in a BFS in* G_s *from* q **do**
4 $u[par] \leftarrow$ the parent vertex of u in the BFS;
5 Add u to V_c if $|V_c| < k$ with $u[test] \leftarrow$ *True*;
6 **for** *vertex* $u \in V_c$ **do**
7 Find a vertex $v \in N(u)$ and $v \notin V_c$ and $v[test]$ is *False*;
8 Find a vertex $c \in V_c$ with $P[c]$ smallest in V_C and $c \neq u$;
9 **if** $P[v] > P[c]$ & $\nexists d \in V_c(d[par] = c)$ **then**
10 $V_c \leftarrow V_c - \{c\} + \{v\}$;
11 $v[test] \leftarrow$ *True*;
12 Return an induced graph G_c with V_c.

Algorithm 3 describes the community search method. We introduce $u[test]$ indicating whether u has been considered swapping or not in Line 1. Each vertex can be considered one time, as the sum of GNN scores increases monotonically along with the vertex swapping. We start a BFS from the query vertex, and initialize the vertex set in the community with the first k vertices encountered from Line 2 to 5. We record the parent vertex $u[par]$ during the BFS which will be used to preserve the connectivity in the following.

The vertex swapping is implemented from Line 6 to 11 iteratively. For each vertex u in the community, we search a vertex pair (v, c), where v is in the neighbors $N(u)$ of u outside the community, and c is the vertex in the community with the smallest GNN score $P[c]$. The community is updated by adding v and removing c, if the swapping can increase the sum of GNN scores in G_c and does not break the connectivity in G_c (indicted by the fact that c is not the parent of a vertex in G_c). We keep swapping until no proper vertex pair can be found.

Figure 4 illustrates the basic idea of the search for *kMG* community. Suppose that the query vertex is a, and an end user makes a positive label on k and a negative label on d. Thus, the probabilities of vertices in research community are relatively higher than

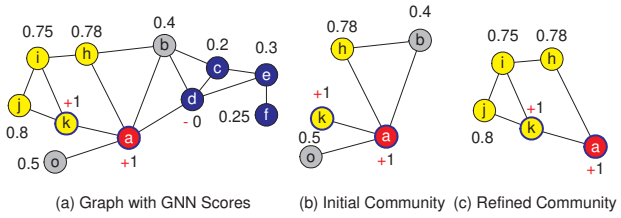


Figure 4: Location of *kMG* (k=5) Community

those on other vertices. The community with 5 vertices is initialized by BFS from the query vertex a in Figure 4(b). Then we swap the vertices with the lowest GNN scores out of the community. For example, j is a neighbor of k, $P[b] = 0.4$ is lower than $P[j] = 0.8$, and then we remove b and add j into community. Finally, we obtain the desired research community in Figure 4(c).

3.5 Algorithm Analysis

We discuss the time/space cost of the ICS-GNN, the quality of the discovered community, and the number of Web API calls. Besides the symbols in Table 2, we use l be the number of layers of GNN model, and n_e be the number of users in each relationship page.

Time complexity of ICS-GNN. Each round of ICS-GNN involves subgraph construction, GNN training and inferring, and the location of *kMG* community. In the first step, each labeled positive vertex in S_p scans l_e pages for edges, each other vertex scans l_p pages for partial edges, and all vertices scan l_c pages for messages. These operations take at most $O(|S_p|l_e n_e + |V_s|l_p n_e + |V_s|l_c)$. In the GNN training, the message propagation between vertices takes $O(|V_s|^2 l)$, where l is the number of layers. For the community discovery, the initial community is constructed via BFS with $O(|E_s|)$ time cost. In the vertex swapping, each vertex is considered at most one time, during which the vertex with the lowest probability in the community is located. The vertex swapping then takes $O(|V_s||V_c|)$ time cost. In summary, the total time complexity is $O(|V_s|l_p n_e + |V_s|^2 l)$, when l_p equals l_e.

Space complexity of ICS-GNN. In the candidate subgraph construction, each positively labeled vertex u will search l_e pages for new vertices. The number of new vertices is $O(|S_p|l_e n_e)$. The other vertices enrich the edge set E_s which is bounded by $O(|V_s|^2)$. All vertices will scan l_c pages for messages. Thus, the candidate subgraph takes $O(|S_p|l_e n_e + |V_s|^2 + |V_s|l_c)$ space cost. For the GNN training, we need to store parameters between layers and the embeddings for all vertices. It requires at most $O(d^2 l)$ to store the parameter matrix and $O(d|V_s|)$ for the space of vertex embeddings, where d is the feature dimension for each vertex. The annotation of attributes on vertices does not impact the space cost complexity in the community location, which is still $O(|E_s|)$. In summary, the total space requirement is $O(|S_p|l_e n_e + |V_s|^2 + |V_s|l_c + d^2 l + d|V_s|)$. Usually, it can be reduced to $O(|S_p|l_e n_e + |V_s|^2 + |V_s|l_c)$, when the size of the feature dimension is less than the total number of vertices, and l is 2.

Quality of the discovered community G_c. G_c is connected, as the initial community G_c via a BFS in Algorithm 3 is connected, and the following swapping operations are allowed only when G_c is still connected. GNN model can combine the structural and content features. Usually, larger GNN scores indicate denser cohesiveness in structural and content features in the community. The swapping operations in Algorithm 3 are carried out only when the sum of GNN scores in G_c increases.

Number of Web API calls. The number of Web API calls in each candidate subgraph construction is in linear to $O(|V_s|)$. From Algorithm 1, each positively labeled vertex searches l_e pages for new vertices and edges, and each other vertex scans l_p pages for partial edges. The number of Web API calls in this part is at most $O(|V_s|\max(l_e, l_p))$. For any vertex u, we search l_c pages for messages. In all, the number of Web API calls is $O(|V_s|(\max(l_e, l_p)+l_c))$ in the worst case. Compared with the exponential number of API calls without control, the crawling policy is friendly to the online social network.

4 OPTIMIZATION

In this section, we allow end users to rank two vertices as feedback besides making explicit labels. In addition, we discuss the *kMG* community location when the query vertex is not the core of the target community.

4.1 Ranking Loss in ICS-GNN

The GNN model in this paper is trained with the labeled vertices indicating whether the vertices should be in the community or not. In some cases, end users may be not certain that one vertex must be in the community. Instead, it is easy for end uses to make a ranking over two candidates, that is, to express the statement like "compared with user A, user B should be more likely in the community".

Such kinds of users' feedback can be captured by the margin ranking loss [23]. Suppose end users assign a set of ranking vertex pairs in the form of $R = \{(v_0, v_1)\}$, where v_0 should have a higher GNN score than v_1. The loss function can be formulated in Equation 2. It means that the loss equals 0 and no parameter adjustment is needed, when the predicted result is correct (the GNN score of v_0 is larger than that of v_1). However, the loss function will guide the parameter optimization if $P[v_0] < P[v_1]$. $0 \le m \le 1$ is the margin to allow a tolerance of the error in the ranking. A higher ranking confidence of end users corresponds to a lower m.

$$Loss_r = \sum_{(v_0,v_1)\in R} \max(0, P[v_1] - P[v_0] - m) \tag{2}$$

By combining Equation 1 and 2, the overall loss function takes the form of Equation 3. λ is a hyperparameter to adjust weights between the label loss and the ranking loss.

$$Loss_a = loss_l + \lambda loss_r \tag{3}$$

Another reason to introduce the ranking loss into the interactive community search lies in its ability to incorporate implicit feedback of end users. When the target community is generated and presented to end users, end users may take more operations (like click, view, search) on some vertices than others. Once collected, these behavior hints can compose a ranking order on vertex pairs, which guides GNN model to yield higher scores on these vertices that are interesting to end users. The margin can be with a relatively large value, as the captured feedback is implicit and imprecise.

4.2 Relative Benefit Based Greedy Community Search

The search method for the *kMG* community in Subsection 3.4 works well when the query vertex is in the core of discovered community. However, the method faces difficulty in handling the cases when the query vertex is a boundary vertex of the community. For example, suppose that a vertex v is labeled positive but it is far from the query vertex. v may be not included in the final community, if v is not connected directly with any vertex in intermediate communities and thus has no chance to be swapped into the final community by Algorithm 3.

Algorithm 4: Relative Benefit Based Greedy Community Search

Input: Candidate subgraph $G_s = (V_s, E_s, F_s, P_s)$, the query vertex q.
Output: Community $G_c = (V_c, E_c, P_c)$.

1 Add q into V_c;
2 **while** $|V_c| < k$ **do**
3 Initialize an vertex queue Q with all vertices V_c;
4 $i \leftarrow 0$;
5 **while** $i <$ *the length of* Q **do**
6 $u \leftarrow Q[i]$;
7 **for** *each neighbor* $v \in N(u)$ *and* $v \notin Q$ **do**
8 $v[par] \leftarrow u$;
9 Locate the shortest path $path(v, G_c)$ from v to G_c with $[par]$ links;
10 Compute the relative benefit of v using $path(v, G_c)$;
11 Add v into Q;
12 $i \leftarrow i + 1$;
13 Locate the vertex $v_m \notin V_c$ with the largest relative benefit;
14 Add vertices in the shortest path $path(v_m, G_c)$ into V_c;
15 Return the induced subgraph G_c with vertex set V_c;

Here, we do not assume that the query vertex is the center of the community, but enable the community to follow its distribution. In order to do so, we introduce a global measure of all vertices to decide whether the vertex should be in the community. As the query vertex may not be the center of the community, the measurement needs to be adjusted each time a vertex is selected into the community. Let $G_c = (V_c, E_c, P_c)$ be an intermediate community. For a vertex $v \notin V_c$, v finds the shortest path $path(v, G_c)$ from v to any vertex in the G_c. We have to add all vertices in $path(v, G_c)$ to keep connectivity if we want to add v into V_c. Thus, the relative benefit of v can be defined by $\frac{\sum_{u \in path(v,G_c)} P[u]}{len(path(v,G_c))}$, which reveals the benefit and cost if v is selected. We select the vertex with the largest relative benefit into the community.

The location of shortest paths from any vertices outside G_c to G_c takes a cost of $O(|V_s||E_s|)$, and then the computation of relative benefit in all iterations takes $O(k|V_s||E_s|)$ cost in determining k vertices in the final community, if the shortest paths are located separately. In order to handle the issue, we start a BFS from all vertices in V_c to the remaining vertices from Line 4 to 12 in Algorithm 4 in each iteration, and locate the shortest path $path(v, G_c)$ from any vertex v outside G_c to G_c with the aid of the parent vertices recorded in the BFS. With the above strategy, the time cost in computing relative benefit is reduced to $O(k|E_s|)$.

We discuss the case of a labeled vertex v far from the query vertex q using the rules above. Initially, the community G_c contains q only, and the relative benefit of v is low since the length of $p(v, G_c)$ is large. The relative benefit of v is recomputed after each vertex is selected into the community. When the intermediate community is extended toward v, the path length can be reduced and then the relative benefit increases, which enables v to have more chances to be selected.

5 EXPERIMENTAL EVALUATION

In this section, we conduct extensive experiments to evaluate ICS-GNN in terms of effectiveness and efficiency.

5.1 Experiment Setup

We perform our experiments on both offline and online networks.

Table 3: The Statistics of the offline graphs

DataSet	$\|V\|$	$\|E\|$	Scenario
Cora	2,708	5,278	Paper citation relationships
CiteSeer	3,327	4,676	Paper citation relationships
PubMed	17K	44K	Paper citation relationships
Reddit	232K	114M	Post co-commented relationships
Fb-Egonet	4,039	88K	User friendship relationships
DBLP	317K	1M	User co-author relationships
Amazon	334K	925K	Product co-purchased relationships

Offline Setting. We use 7 real-life graphs in our experiments with their statistics in Table 3. These graphs are divided into 3 representative groups. The graphs in the first group have content features/vertex class labels but without ground-truth communities. We select *Cora*, *CiteSeer* and *PubMed*, which are widely used in GNN model performance study (they can be downloaded from project [5]). These graphs describe research papers and their citation relationships. The vertex class labels reveal the research areas/topics that papers belong to, from which we simulate communities. We can select a vertex u as the query vertex, and a subgraph around u can then be constructed. The vertices in the subgraph are labeled with 1 if they share the same class as u, or are labeled with 0 otherwise. Intuitively, the vertices with the label of 1 are for the paper community in the same research area.

Graphs in the second group have content features and ground-truth communities. We select Reddit [6] and Fb-Egonet (Facebook Egonet) [10, 22] in this group. Reddit graph is collected from an online discussion forum, where vertices refer to posts, and an edge between two post vertices exists if a user comments on both posts. The vertex labels reveal the communities that posts belong to. Facebook Egonets are collected in [22] and used in [10]. Each Egonet f_x contains an induced subgraph including a query vertex x and its neighbors. The vertices with the similar interest form social circles or communities in Egonets.

Graphs in the third group have ground-truth communities but without content attributes. These graphs can be found in SNAP project [22]. We select DBLP and Amazon graphs in this group due to the space limitation. As ICS-GNN can search communities by considering both content and structural features, we generate content features using Gaussian distribution with a mean value of 0 and a standard deviation of 1. Note that some previous methods [10] also generate content features on these graphs.

There are few works in interactive community search, and the extension of existing community search methods [3, 10] to an interactive scenario is not trivial. End users have to adjust the parameters in rules in each round. Take LocATC (Attributed Truss Community) as an example. End users may need to refine the parameters in (k, d)-truss, and select the representative attributes in each round to locate the desired community. Thus, we study the performance of one-time community search of ICS-GNN in an offline data set, and choose LocATC as our competitor.

The community search in the offline graphs can be simulated as follows: We first randomly select a query vertex q in the graph, and perform a BFS from u to build a candidate subgraph. The vertex community membership comes from the ground-truth or label-generated communities, according to different graphs. We select the training data according to a training ratio p_t. That is, we randomly select a labeled positive vertex set S_p (including the query vertex u) containing $|V_s|p_t/2$ vertices with the label of 1, and randomly select the same number of negative examples with the label of 0. Next, we train a GNN model on the subgraph using the labeled vertices, and infer the GNN scores for other vertices. Finally, we locate the kMG community G_c (the size of the community actually equals $k+|S_p|$ including labeled positive vertices) using different methods, and all vertices in the discovered community have a predicted value of 1.

We compute the precision of the discovered community G_c as $|V_c' - S_p|/|V_c - S_p|$ to evaluate different methods, where $V_c' - S_p \subseteq V_c - S_p$ contains all newly discovered vertices in the community but excludes the labeled positive vertices. Usually, a larger subgraph with more structural and content features indicates a more stable result and a higher precision. The size of the community also impacts the precision, since the number of vertices with label 1 may be smaller than the size of the community. Then, a larger community may lead to a lower precision. Note that it is easy for end users to adjust the size of community in ICS-GNN according to the discovered results.

We implement all the algorithms in ICS-GNN using Python 3.7, and run experiments on a machine having two Intel(R) Xeon(R) Gold 2.30GHz CPU, 256GB memory, and 2 GeForce RTX 2080 Ti Graphics Cards (GPU), with Ubuntu 18.04 installed. We choose Pytorch Geometric implementation [5] of GCN [12] in the GNN training. GNN and GCN are used interchangeably in the following.

Online Setting. We take Sina Weibo in our online test. The Web crawler follows the Scrapy framework [3], and the candidate subgraph is constructed using Algorithm 1 with a given query vertex. The content embeddings for users are initialized based on the Gaussian distribution first. We then pre-process the message posts using pre-trained 300 dimension embeddings in Fasttext [4]. Since both Chinese and English sentences may exist in the message posts, the total size of each user's content features is 600. We do not include the profile of users in this test. The communities are located using the method following the offline settings above, and the located community is visualized by Gephi [5].

[3]https://scrapy.org/
[4]https://fasttext.cc/

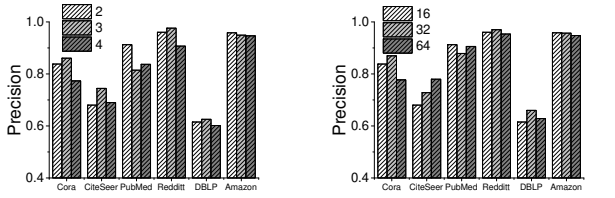
Table 4: Default Values for Parameters

Parameter	Values	**Parameter**	Values
Size of subgraph	400	Training Epochs	200
Size of community	30	Drop out	0.5
Training ratio	2%	Learning-rate	0.01
λ in Equation 3	1	Layers in GCN	[16,2]

We list the default values of parameters in Table 4. The left column contains the default values of the parameters in ICS-GNN, while the right column contains those in GCN training. These parameter values are used in the following tests unless stated otherwise.

5.2 Experimental Results on Offline Networks

In this subsection, we first determine the hyper-parameters of the GCN model. With the selected parameters, we study the effectiveness of incorporating the ranking loss into GNN model, and show the effectiveness and efficiency of the community search methods. Finally, we compare with the competitor in term of effectiveness.



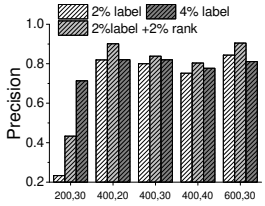
(a) Varying of # of Layers (b) Varying of size of Hidden Layers

Figure 5: Studies of Hyper-parameters of GCN

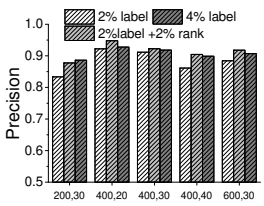
Figure 5 shows the results of different hyper-parameters in the one-time community search using GCN model on different data sets. We first test the GCN models with different layers (2, 3, 4) with the size of layer 16. It is observed that more layers cannot result in significant advantages, at least using the current version of the GCN. We can reach a similar conclusion from the related works [8]. That is, without the extra optimization, the increase of layers in GCN will oversmooth the vertex embeddings, and weaken their classification abilities. Thus, we select 2-layer GCN models as the default. We then test the results with different sizes (64, 32, 16) of the hidden layer, and find that we achieve similar results using these parameters. We then take 16 as the default size of hidden layers to lower the training cost.

We simulate the incorporation of ranking loss into the interactive setting using 2-round searching as follows: We train a model $\mathcal{M}_0$ on a subgraph with 2% labeled vertices (including the positive and negative examples) in the first round. We then infer the vertex

[5]https://gehpi.org/



(a) Ranking loss on Cora



(b) Ranking Loss on PubMed

Figure 6: Effectiveness of Ranking Loss

membership probabilities for other vertices. As an end user may quickly find these vertices with obvious errors, we simulate the process by locating false positive vertices with high GNN scores and false negative vertices with low GNN scores, and composing 2% vertex pairs in ranking loss for the model $\mathcal{M}_1$ in the second round. We also compare a model $\mathcal{M}_2$ using 4% training vertices in the first round.

Figure 6 shows the effectiveness of the ranking loss on two data sets with varying settings. The data presented as *"200, 30"* in the x-axis is for the size of subgraph 200 and the size of community 30. From the results, we can see that the ranking loss function works well in nearly all settings. It boosts the performance of $\mathcal{M}_0$ significantly, and achieves similar or even better results than $\mathcal{M}_2$, although the ranking task is relatively easy compared with the explicit labeling.

We implement 4 methods to locate the community. We are interested in whether the *BFS* with only structural information can produce a community with a high precision. The method based on the BFS with the vertex swapping, denoted by *BFS-S*, is described in Algorithm 3. The method in Algorithm 4 is named as *Greedy-G*. We also implement a simplified version *Greedy-T* that shares a similar idea to *Greedy-G*, but computes the relative benefit using a path to the query vertex directly, which can be accelerated by the cached results of one-time BFS from the query vertex.

Figure 7(a)-(f) report the precision results on community discovered on different networks. The symbols of *"30,400, 4%"* in the figures describe the size of the community, the size of the subgraph, and the training ratio in GNN, respectively. We can see that *Greedy-G* achieves the best performance compared with other methods in the same setting in general. *BFS* alone does not yield the precise community, showing that only structural features are insufficient. A small subgraph always indicates incomplete features, which impacts the effectiveness of GNN models and the following results. The increase of the training ratio can improve the performance of GNN model on the one hand, but may lower the precision of community on the other hand, when the number of vertices with the label of 1 is smaller than the size of the community.

Figure 7(g) illustrates the training cost of the GNN model on different sizes of the subgraphs from various networks. We can see that the absolute time of the model training on all test graphs is less than 1.5 seconds with the aid of GPU. Note that we train the GNN model on the subgraphs collected from the underlying graph, and then the size of the underlying graph has no impact on the model training.

Figure 7(h) shows the efficiency of our community discovery methods. *Greedy-G* computes the relative benefit for all vertices multiple times, each of which a BFS from the intermediate community to the remaining vertices is required. By contrast, *Greedy-T* also computes the relative benefit for all vertices multiple times, but relies on the same cached BFS results from the query vertex to the remaining vertices. Such an optimization strategy shows its advantage only in dense graphs such as Reddit. We also note that the absolute execution time of *Greedy-G* is acceptable due to that we limit the size of the underlying subgraph. By considering the previous results on community precision, we choose *Greedy-G* as our default community discovery method.

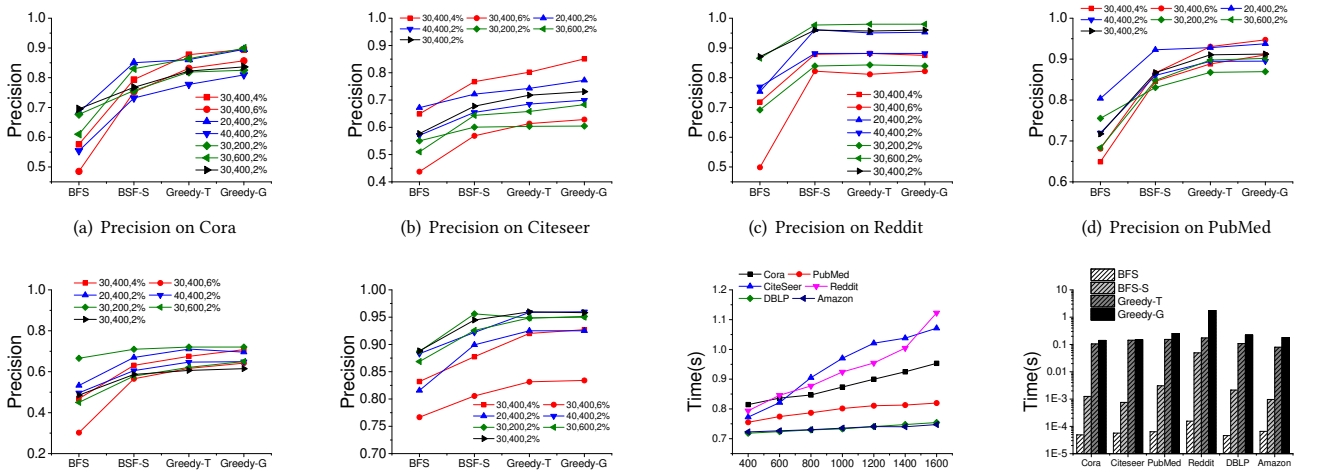
Figure 8 studies the community's precision achieved by the increase of iterations. At each time, one positive and one negative vertex are labeled. As expected, GNN model becomes more effective with more labeled vertices, which results in more precise communities generally. Note that the precision does not necessarily increase with more iterations from Figure 8(a), as the limited labeled vertices cannot ensure stable results. The symbols such as *"30, 1"* in Figure 8(b) refer to the community size and the number of newly labeled positive (negative) vertices in each iteration. We select Amazon graph to test the impacts of these two factors along with the number of iterations on the community precision, and find the similar trends.

Figure 9 shows the performances of our method and LocATC method [10] (codes provided by authors) over Fb-Egonets. In order to make the comparison fair, we follow the same experimental setting in [10]. For each community (with its size larger than 8), LocATC selects 8 vertices randomly as the query vertices. Correspondingly, ICS-GNN uses 4 positive and 4 negative vertices. LocATC chooses two attributes which are common in the current community but rare in other communities, while ICS-GNN does not need to select attributes. After LocATC yields a k-sized community, ICS-GNN utilizes the same value of k in the *kMG* community search, from which we compute the precision, recall, and F1 over discovered communities [10] but excluding positively labeled vertices. As labeled vertices are randomly selected, we average scores after 20 rounds of community search for both methods. We illustrate F1, P (Precision), and R(Recall) in each Egonet using the two methods, and find that ICS-GNN achieves better F1 scores in 9 out of 10 Egonets. The scores of LocATC are lower than those reported [10], as we focus on community with size larger than 8 and exclude the labeled vertices in score computation.

5.3 Experimental Results on Online Network

We use Sina Weibo, the largest Weibo system in China, as our online context. Prof Xuemin Lin is a well-known researcher in the database field, and we select Xuemin as the query vertex. We set 2 for all page limitations, including l_c , l_e, l_e in Table 2 in the incremental crawler. The size of community is 40.

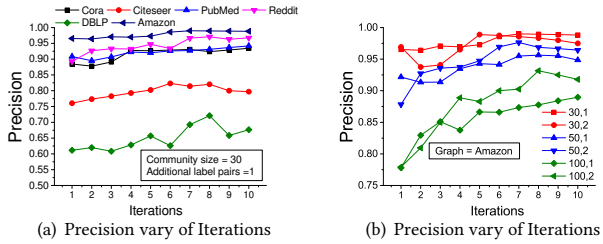
Figure 10 (a) shows the first round community discovered over the subgraph containing about 200 vertices, after 3 rounds of incremental crawling. We then train a GNN model with positive examples (Bin Cui, and Xiaoyong Du), and negative examples (Ping Lang, the coach of a volleyball team). Guided by the examples, most members in the initial community are also researchers, including



(a) Precision on Cora (b) Precision on Citeseer (c) Precision on Reddit (d) Precision on PubMed

(e) Precision on DBLP (f) Precision on Amazon (g) Efficiency on GNN Training (h) Efficiency on Community Search

Figure 7: Effectiveness and Efficiency of Location of the *kMG* Community



(a) Precision vary of Iterations (b) Precision vary of Iterations

Figure 8: Impact of Iterations in Interactive Search

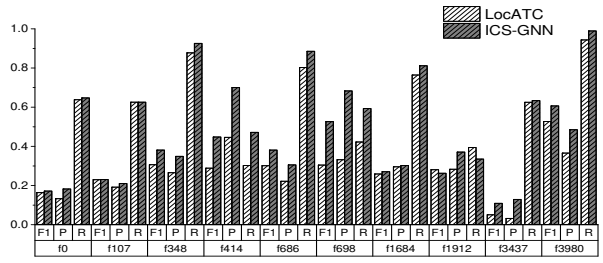


Figure 9: F1, Precision and Recall Scores on Fb-Egonets

Xiaofang Zhou, Hengtao Shen, etc. The size of the circle is related to the degree of the vertices in the final community.

Figure 10 (b) illustrates the second round of sports community. Based on the subgraph in Figure 10 (a), we swap the positive and negative examples (Xueming is still a positive example), and invoke another 3 rounds of incremental crawling from the volleyball coach (Ping Lang). We retrain the GNN model to find the sports community. We can see that Xueming now is not the core of discovered community, as most researchers have no direct relationships with the players in the currently collected data. As discussed above, our method supports the community search with various forms, like the tree form in the organization.

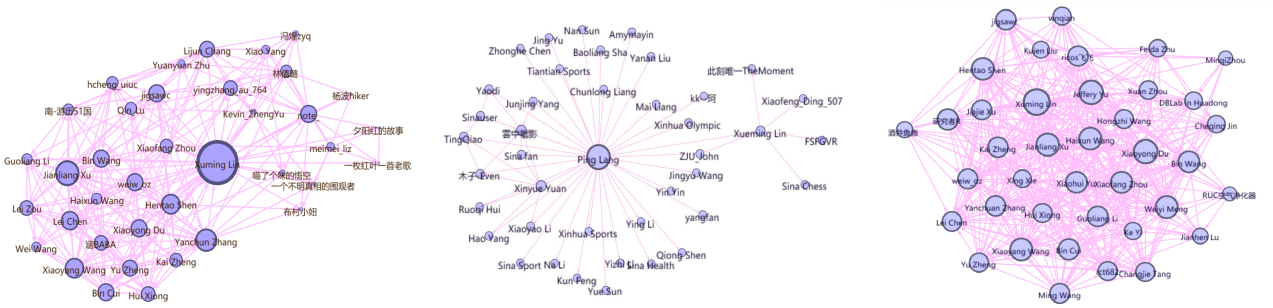
Figure 10 (c) shows the second round of research community. Based on the subgraph in Figure 10 (a), we make another 3 rounds of incremental crawling to get more pages from the labeled positive vertices, Bin Cui and Xiaoyong Du. We perform the community search in the new subgraph, and can see that the newly discovered community becomes dense in structure. Professors like Jeffery Yu, Haixun Wang, Jiangliang Xu are active in the community discovered.

Summary. From the above experimental results, we can draw the following conclusions: i) ICS-GNN can effectively capture the content and structural features by GNN, and yield a high-quality community with the limited labeling efforts. ii) The margin ranking loss is effective in the context of interactive community search. iii) The greedy community discovery method with the global relative benefit can produce communities fitting various distributions.

6 RELATED WORK

We review the progress of related works, including the community search, the focused crawler, GNN models and their impacts on the community search.

Community Search. Most existing methods assume that the community should satisfy both content similarity and structural cohesiveness. In order to capture the structural constraints, different k-* related measurements [2, 9, 14, 20] are proposed from the perspectives of the degree, the number of triangles, the maximum diameter, etc, in the community. Usually, the communities meeting these structural constraints achieve good precision. However, it is not easy for end users to set a proper k. A higher k fails to produce any communities, while a smaller k results in massive candidate communities. Other works [3, 10] measure the communities by combining both structural and content features using rules. As discussed above, the interactive adjustment of these rules is not trivial due to complex relationships among features, as well as tension between content and structural requirements.



(a) 1-st round Research Community (b) 2-nd round Sports Community (c) 2-nd round Research Community

Figure 10: Case Study in Sina Weibo. We translate Chinese names into English when we know the persons, and keep their original forms in other cases.

Focused Crawler. Focused crawlers [16, 18] are designed to collect relevant Web pages with a classifier trained on the local information (links and content of Web page), which determines the next link to be searched. Although interactive community search shares a similar purpose as focused crawlers, we use the GNN to train a model on the entire subgraph instead of local information in one vertex. In addition, focused crawler collects a set of pages, while the interactive community search finds meaningful subgraphs.

GNN for Community. GNN encodes the content and structural features into the vertex embeddings, which are used in downstreaming tasks like classification. Various GNN variants and models [6, 12, 21] are proposed with their differences in aggregate functions, update functions, and transductive/inductive settings, etc. Besides the works discussed before, Addgraph [23] utilizes a margin loss function to handle imprecise labeled examples. LightGCN [8] studies the impacts of different factors like deep layers on the model performance, and proposes a simple but effective model. In this work, we exploit the similar ranking based loss function to provide end users with a simple labeling mechanism. For the hyperparameters used in GCN, we choose 2-layer GCN as default, as more layers cannot improve the precision of community obviously if no extra optimization is adopted.

The roles of deep learning in community detection have been studied, and the challenges and opportunities are summarized in [15]. In the work of LGCN [1], the vertex's community membership is modeled as a vertex classification problem, and a combined GNN model on the original graph and its line graph is proposed to predict probabilities on each vertex. The overlapping community detection method is proposed [19] by incorporating Bernoulli Poisson probabilistic model into the loss function in GNNs. Similar to these works, ICS-GNN relies on GNN model to infer community membership probabilities on vertices. The differences between our work and the existing ones are that we abstract a *kMG* model for communities, and locate the *kMG* community incrementally and interactively rather than yielding the probabilities on vertices only.

7 CONCLUSIONS

This paper proposes ICS-GNN to support the interactive community search in an online social network. We first model the community as a k-sized subgraph with maximum GNN scores, which combines content and structural features by leveraging a GNN model, and incorporates users' feedback easily. Second, we design an interactive framework with incremental crawling and community search to progressively refine the results and avoid unnecessary communication and computation cost. Third, we introduce various optimization strategies to reduce the labeling burden of end users, and a greedy method to locate communities with flexible forms. Offline and online experimental results illustrate the effectiveness and efficiency of ICS-GNN.

ACKNOWLEDGEMENT

We would like to thank Dr. Yixiang Fang for sharing ACQ codes, and thank Dr. Xin Huang for sharing their executable LocATC codes. We would like to thank the comments from anonymous reviewers. This work was partially supported by NSFC under Grant No. 61832001, Alibaba-PKU joint program, Zhejiang Lab under Grant No. 2019KB0AB06, and Zhejiang Provincial Natural Science Foundation (LZ21F030001).

REFERENCES

[1] Zhengdao Chen, Lisha Li, and Joan Bruna. 2019. Supervised Community Detection with Line Graph Neural Networks. In *ICLR*.
[2] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *SIGMOD*. 277–288.
[3] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective Community Search for Large Attributed Graphs. *Proc. VLDB Endow.* 9, 12 (2016), 1233–1244.
[4] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.
[5] Matthias Fey and Jan Eric Lenssen. 2019. *Fast Graph Representation Learning with PyTorch Geometric.* http://arxiv.org/pdf/1903.02428v3:PDF
[6] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[7] Jessica B. Hamrick, Kelsey R. Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Josh Tenenbaum, and Peter W. Battaglia. 2018. Relational inductive bias for physical construction in humans and machines. In *CogSci*.
[8] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.
[9] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.
[10] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-Driven Community Search. *Proc. VLDB Endow.* 10, 9 (2017), 949–960.
[11] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations*. 85–103.

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[13] Bofang Li, Aleksandr Drozd, Yuhe Guo, Tao Liu, Satoshi Matsuoka, and Xiaoyong Du. 2019. Scaling Word2Vec on Big Corpus. *Data Sci. Eng.* 2, 4 (2019), 157–175.
[14] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (2015), 509–520.
[15] Fanzhen Liu, Shan Xue, Jia Wu, Chuan Zhou, Wenbin Hu, Cécile Paris, Surya Nepal, Jian Yang, and Philip S. Yu. 2020. Deep Learning for Community Detection: Progress, Challenges and Opportunities. In *IJCAI*. 4981–4987.
[16] Robert Meusel, Peter Mika, and Roi Blanco. 2014. Focused Crawling for Structured Data. In *CIKM*. 1039–1048.
[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.
[18] Kien Pham, Aécio S. R. Santos, and Juliana Freire. 2019. Bootstrapping Domain-Specific Content Discovery on the Web. In *WWW*. 1476–1486.
[19] Oleksandr Shchur and Stephan Günnemann. 2019. Overlapping Community Detection with Graph Neural Networks. *CoRR* (2019). http://arxiv.org/abs/1909.12201
[20] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*. 939–948.
[21] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[22] Jaewon Yang and Jure Leskovec. 2012. Defining and Evaluating Network Communities Based on Ground-Truth. In *ICDM*. 745–754.
[23] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. 2019. AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN. In *IJCAI*. 4419–4425.
[24] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR* (2018). http://arxiv.org/abs/1812.08434