COCLEP: Contrastive Learning-based Semi-Supervised Community Search

Ling Li[1,2], Siqiang Luo[2*], Yuhai Zhao[1*], Caihua Shan[3], Zhengkui Wang[4], Lu Qin[5]
[1]Northeastern University, China, [2]Nanyang Technological University, Singapore, [3]Microsoft Research Asia,
[4]Singapore Institute of Technology, Singapore, [5]University of Technology Sydney, Sydney
lilingneu@gmail.com, siqiang.luo@ntu.edu.sg, zhaoyuhai@mail.neu.edu.cn,
caihuashan@microsoft.com, zhengkui.wang@singaporetech.edu.sg, lu.qin@uts.edu.au

***Abstract*—Community search is a fundamental graph processing task that aims to find a community containing the given query node. Recent studies show that machine learning (ML)-based community search can return higher-quality communities than the classic methods such as k-core and k-truss. However, the state-of-the-art ML-based models require a large number of labeled data (i.e., nodes in ground-truth communities) for training that are difficult to obtain in real applications, and incur unaffordable memory costs or query time for large datasets. To address these issues, in this paper, we present the community search based on contrastive learning with partition, namely COCLEP, which only requires a few labels and is both memory and query efficient. In particular, given a small collection of query nodes and a few (e.g., three) corresponding ground-truth community nodes for each query, COCLEP learns a query-dependent model through the proposed graph neural network and the designed label-aware contrastive learner. The former perceives query node information, low-order neighborhood information, and high-order hypergraph structure information, the latter contrasts low-order intra-view, high-order intra-view, and low-high-order inter-view representations of the nodes. Further, we theoretically prove that COCLEP can be scalable to large datasets with the min-cut over the graph. To the best of our knowledge, this is the first attempt to adopt contrastive learning for community search task that is nontrivial. Extensive experiments on real-world datasets show that COCLEP simultaneously achieves better community effectiveness and comparably high query efficiency while using fewer labels compared with the-state-of-the-art approaches and is scalable for large datasets.**

I. Introduction

Community search aims to extract a cohesive subgraph (or a community) containing a query node, and is widely used in various applications, such as event organization, friend recommendation, fraud detection, targeted advertisement [1], [2]. Much research has been devoted on community search algorithms (e.g., [3]–[8]).

The conventional community search methods often define a community based on an explicit structural requirement (e.g., k-core [3], [7], k-truss [4], [6], [9], and k-ecc [5], [10]), and try to capture different aspects of the cohesiveness of the community. For instance, a k-core is a dense subgraph such that each node connects to at least k other nodes. k-truss requires that each edge in the subgraph is contained in at least k-2 triangles. k-ecc is a subgraph such that after removing any $k-1$ edges, it is still connected. However, such a predefined structure can be inflexible in real applications, as the community structures may vary significantly in different types of networks.

* Corresponding Author

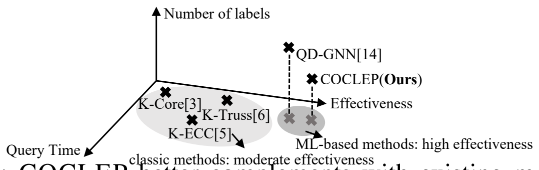


Fig. 1: COCLEP better complements with existing methods in terms of community effectiveness, query efficiency and the number of labels required.

Therefore, the recent research works have proposed the machine learning (ML)-based community search methods, i.e., ICS-GNN [11] and QD-GNN [12] on the attributed graphs that are flexible to handle any community structures, to overcome the limitations of conventional methods. Given some ground-truth communities, these approaches aim to learn a model that finds a suitable community for any given query node. For instance, ICS-GNN trains a model for each query task with the given positive and negative labels. QD-GNN uses the training dataset that contains the query nodes and the corresponding ground-truth community to train a model. Though existing ML-based community search methods have shown very promising results on the task, they still suffer from several limitations. (1) *They heavily rely on a large number of labeled data to train the model*. Particularly, QD-GNN requires to know the whole community (hence, many labels) for the query nodes in the training dataset. ICS-GNN needs to know both positive and negative pairs for *every node* in the graph, where the positive pairs refer to two nodes in the same community. Unfortunately, in many real applications, such a large number of labeled data may not be available. (2) *They incur significant memory costs or query time for large datasets.* Particularly, QD-GNN needs to train the model from the whole graph, which easily leads to out of memory for large graphs. Although ICS-GNN is scale for large network w.r.t. the memory usage, it suffers from slow query processing, as it needs to retrain the model for each query node. Therefore, this calls for a memory and query efficient ML-based method that does not require the large number of labeled data for community search.

In this paper, we make the first attempt to adopt the contrastive learning in the community search tasks. Contrastive learning has been widely used in other tasks for graph representation learning [13]–[15] as it has shown a great potential of using no or only a few labels for model training. But it has never been explored in community search. In order to take advantage of contrastive learning, we foresee three main challenges of introducing contrastive learning into the community search tasks.

(1) *How to encode the nodes and community information?* Community search is a query-driven task that aims to discover the community containing the query node. However, the existing contrastive learning methods usually learn representations for *all nodes equally*, instead of being *query-driven*. The problem arises in how to encode the query node and the corresponding labeled nodes. Furthermore, positive pairs and negative pairs in classic contrastive learning do not carry the community information. Hence, we need to expand the scope of designing positive pairs and negative pairs to mediate the community information.

(2) *How to generate an augmentation view for community search in a graph?* Contrastive learning includes an augmentation view as a crucial component. The augmentation view is a *modified graph* transformed from the original graph without affecting the semantics for downstream tasks. It can be seen as the supervision information for model training. Existing methods for graph data augmentation include random node dropping, edge perturbation, attribute masking, and subgraph sampling [14]. However, these methods are at risk of damaging the semantics of the generated graph for community search. For instance, the inherent community properties (e.g., density) may be heavily changed. Additionally, these methods do not carry neighborhood information, which is helpful for community information capturing. For example, most neighbors of a node are located in the community containing this node. Therefore, we need new designs of the augmentation view for community search.

(3) *How to alleviate the memory and time costs?* Existing contrastive learning methods train a model from the entire graph, which leads to high memory and time costs. However, for the community search task, the nodes in the community are usually located around the query node. The nodes located far from the query nodes will not contribute to the model training. Thus, training based on an entire graph is often not necessary. Exploring efficient training methods can significantly reduce memory and time costs.

Our Approach. To address these challenges, we propose COCLEP (**Co**mmunity search based on **C**ontrastive **Le**arning with **P**artition) with several novel designs. Before proposing the COCLEP, we develop the COCLE, which is the **Co**mmunity search based on **C**ontrastive **Le**arning without partition. First, we propose a new attention-driven graph neural network to encode the nodes into low-dimensional vectors. Being effective in integrating the neighboring information into a node and fusing the query node information and structure information, the attention-driven graph neural network brings benefits in learning the set of community nodes that share similar features to the query node. Second, to capture the community information, the label-aware contrastive learner, including intra-view and inter-view learning is proposed. Third, we design the augmentation view in contrastive learning as a hypergraph, which is then encoded by the attention-driven hypergraph neural network to capture the higher-order structure of the neighborhood information. Finally, to reduce the model training memory cost, COCLEP partitions the graph first and rejoins the community results in each partition. We theoretically analyze why the partition is reasonable. Our analysis also shows that the training memory and time costs are geared to the partition size instead of the whole graph size.

It is important to note that we do not aim to completely replace classic non-ML-based community search models with ML-based methods. Our main purpose is to explore the potential novelty of the ML-based community search, which complements the decade-old but still powerful classic methods and, arguably, continues the new research direction for ML-based community search opened by recent studies [11], [12]. Figure 1 shows a general comparison between the classic methods and ML-based methods, including COCLEP. One important benefit of the classic methods is that it requires no ground-truth knowledge (i.e., labels), whereas the ML-based methods (e.g., QD-GNN and ICS-GNN) excel in community effectiveness at the expense of using ground-truth communities. When ground-truth communities are available (e.g., see many datasets in [16]), the ML-based methods can be preferred for their high effectiveness. Further, we argue that the burden on obtaining the ground-truth labels can be further lifted by our COCLEP, which requires much fewer labels (e.g., 3) while maintaining high community effectiveness. In summary, COCLEP takes the advantages of both the conventional methods (no prior-knowledge required) and the ML-based methods (high effectiveness).

Contributions. Our contributions are summarized as follows:
- We present a contrastive learning-based semi-supervised model for community search in a graph, namely COCLEP, which is both memory and query efficient requiring very few labeled data. To the best of our knowledge, this is the first work to introduce contrastive learning in community search.
- We propose the COCLE that includes a new label-aware contrastive learner, where a hypergraph-based augmentation method and a contrastive learning strategy that integrates low-order intra-view learning, high-order intra-view learning and low-high-order inter-view learning are proposed. Further, we design a new attention-driven graph neural network as well as its combination with graph partitioning (COCLEP).
- We conduct extensive experiments on real networks and the results demonstrate that our proposed models achieve significantly better performance w.r.t. the query efficiency, training time, community effectiveness, and the requirement of labeled data.

II. Related work

Community Search. Community search has been extensively studied in recent years. It aims to find densely connected subgraphs containing query nodes. To capture the cohesive structure, different models have been proposed, including k-core [3], [7], [17], k-truss [4], [6], [9], k-ecc [5], [10]. See [1] for a survey. Besides simple graphs, community search is also studied in more complex graphs like geo-social networks [18]–[22], attributed networks [23]–[25], heterogeneous information networks [26], [27], bipartite graphs [28]. However, such predefined structure-based community search methods are inflexible. Recently, some ML-based methods have been proposed to solve this limitation. Gao et al. [11] first studied community search based on graph neural networks. They see the community search problem as a binary classification problem. But their model is a query-specific model. Once a new query comes in, they need to relearn a new model for the new query, which is time-consuming. Besides, for each query, they need the labeled data with incremental provided. In reality, it is impossible for users to easily provide labels for all queries. Jiang et al. [12] proposed QD-GNN and AQD-GNN to solve community search problem on simple graphs and attributed graphs, respectively. They also see the community search problem as a binary classification problem.

Contrastive Learning. Contrastive learning has been widely used in computer vision [29], [30], natural language [31], [32] and the graph domain [13]–[15], [33]–[35]. It sees the data themselves as supervision information without labels to make similar samples closer and dissimilar samples far from each other. To generate similar samples, different data augmentation methods have been proposed. For example, Qiu et al. [13] use the random walk within the same k-hop subgraphs to generate similar samples. You et al. [14] study different graph augmentation methods, such as node dropping, edge perturbation, attribute masking, and random walk induces subgraph. Zhu et al. [33] proposed adaptive augmentation by considering the important nodes in the graph. All these methods do not consider the label information, causing similar samples in the graph being seen as dissimilar samples, further influencing the performance. To solve this problem, some works [35], [36] proposed supervised contrastive learning with labeled data. But all the existing approaches are focused on node representation learning, link prediction, and node classification tasks. They cannot be directly applied to the community search task since it is a query-dependent task. Recently, Wang et al. [37] proposed a contrastive loss that directly optimizes alignment and uniformity of the node features and leads to comparable or better performance.

III. Problem Definition

We focus on an undirected graph $G(V,E)$ with node set V and edge set E. Let $n = \mid V \mid$ and $m = \mid E \mid$ be the number of nodes and edges respectively. Given a node $u{\in}V$, the neighbor set of u is $N(u)$=$\{v|(u,v){\in}E$ or $(v,u){\in}E\}$. The adjacency matrix of G is denoted as $\boldsymbol{A}{\in}\{(0,1)^{n\times n}\}$, $\boldsymbol{A}_{i,j}=1$ iff $(v_i,v_j){\in}E$.

Community Search: Given a graph G and a query node q, community search aims to find a cohesively connected subgraph containing q. Here, we use the set of nodes C_q in the subgraph to denote the community to which q belongs.

Based on the definition of community search, we formulate our ML-based community search problem: given a graph G and a set of query tasks with (community) labeled nodes, i.e., $\mathcal{D}$=$\{(q_1,pos(q_1)),\ (q_2,pos(q_2)),\ \ldots\}$, where q_i is the query node and $pos(q_i)\subseteq C_{q_i}$ is the set of positive samples with respect to query node q_i. We aim to build a machine learning model that can learn a function $\mathcal{F}(q,v){\in}\boldsymbol{\mathcal{R}}$ that outputs the probability of node $v{\in}V$ belonging to C_q. Note that we aim to learn a model that benefits various queries in the same graph. The underlying assumption is that there exists one common structural pattern that composes the community for different queries in a graph. Given a new query node q_{new}, once the probabilities $\mathcal{F}(q_{new},v)$ for all nodes are obtained, we can find the community $C_{q_{new}}$ that is composed by the nodes whose probability are higher than the given threshold p.

TABLE I: Frequently used notations.

Notation	Meaning
$G(V,E)$	the input graph.
$G_{aug}(V_{aug},E_{aug})$	the augmentation graph of graph G.
n, m	the number of nodes and edges in G.
$N(u)$	the set of neighbors of u in G.
$C_q(G)$	the community containing q in graph G.
$pos(q,G)$	the nodes in ground-truth community C_q.
L	total number of GNN layers.
$\boldsymbol{h}_u^{(l)}$, $\boldsymbol{z}_u^{(l)}$	the representation of node u in G and G_{aug} in the l-th layer, respectively.
δ	the activation function.
$\mathcal{D}$	the training dataset.
$\hat{C}_{q_i}^b$	the boundary of one partition.

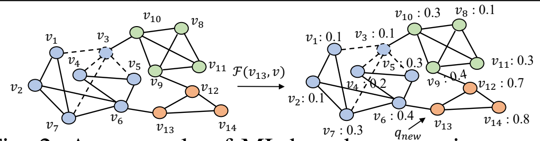
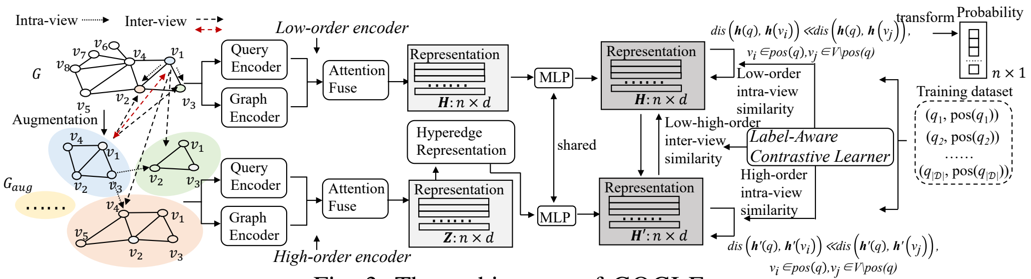


Fig. 2: An example of ML-based community search.

Example. *Figure 2 shows an example for ML-based community search. Given the graph G, the nodes of a ground-truth community are marked by different colors. For example,* C_{v_7}=$\{v_1,v_2,v_3,v_4,v_5,v_6\}$*. Then the set* $pos(v_7)$ *can be* $\{v_1,v_2,v_3\}$ *which is a subset of* C_{v_7}*; the set* $pos(v_{10})$ *can be* $\{v_8,v_9,v_{11}\}$*. The training dataset* $\mathcal{D}$ *is composed by these pairs of tasks, i.e.,* $\mathcal{D}$=$\{(v_7,(v_1,v_2,v_3)),(v_{10},(v_8,v_9,v_{11})),\ldots\}$*. With* $\mathcal{D}$*, we aim to learn the function* $\mathcal{F}$ *that could infer the community for any query node* q_{new}*. Suppose* q_{new}=v_{13}*, as Figure 2 shown, the probability of all nodes are reported through* $\mathcal{F}$*. Given a probability threshold p=0.5,* $C_{v_{13}}$=$\{v_{12},\ v_{14}\}$ *because only* v_{12} *and* v_{14} *have probabilities that are larger than* 0.5.

IV. COCLE: Contrastive Learning-Based Community Search

Figure 3 gives the overall architecture of COCLE, which mainly includes three components, i.e., node feature encoder, high-order augmentation encoder, and label-aware contrastive learner. Given a graph G and the training dataset $\mathcal{D}$=$\{(q_1,pos(q_1)),\ (q_2,pos(q_2)),\ \ldots\}$, the node feature encoder is used to obtain the representation of each node in G by fusing the query node information and the lower-order neighboring information of the node. The high-order augmentation encoder captures the high-order community information of the nodes from an augmentation hypergraph. The label-aware contrastive learner obtains the final representations of the nodes with a carefully designed learning strategy that


Intra-view
Inter-view
Low-order encoder
Query Encoder
Graph Encoder
Attention Fuse
Representation
$H: n \times d$
MLP
shared
Hyperedge Representation
$Z: n \times d$
$H': n \times d$
High-order encoder
Augmentation
G
G_{aug}
$dis(h(q), h(v_i)) \ll dis(h(q), h(v_j))$,
$v_i \in pos(q), v_j \in V \setminus pos(q)$
$dis(h'(q), h'(v_i)) \ll dis(h'(q), h'(v_j))$,
$v_i \in pos(q), v_j \in V \setminus pos(q)$
Low-order intra-view similarity
Low-high-order inter-view similarity
High-order intra-view similarity
Label-Aware Contrastive Learner
transform
Probability
$n \times 1$
Training dataset
$(q_1, pos(q_1))$
$(q_2, pos(q_2))$
……
$(q_{|\mathbb{D}|}, pos(q_{|\mathbb{D}|}))$
Fig. 3: The architecture of COCLE.

enlarges the contrastive scope and incorporates both high-order and low-order information. Once the model is trained, we can get a function $\mathcal{F}(q, v)$ that maps each node v in G to a probability that v belongs to the community C_q w.r.t the query node q. When a new query q_{new} comes, q_{new} and all nodes in G are encoded by COCLE and the representations of all nodes in G are transformed into the probabilities. All nodes whose probabilities are above a given threshold are output as the community of q_{new}.

To be effective, COCLE considers the challenges (1) and (2) mentioned in Section I. In particular, the challenge (2) of the augmentation view generation for community search is addressed in the high-order augmentation encoder by constructing a hypergraph. Since nodes in the same neighborhood form hyperedges and no nodes or edges are randomly removed, the community information is captured without the relationships among nodes damaged. For the challenge (1) of node encoding with community information, the constructed hypergraph and the community structure are utilized in the label-aware contrastive learner. As a result, the community information is integrated into the node encoding and the contrastive scope is expanded to intra-view and inter-view from both low-order and high-order views. Moreover, an attention-based graph convolutional network is used in the node feature encoder so that the node encoding is query-driven.

Constrative learning is the core of COCLE. As shown in Figure 3, it includes intra-view learning and inter-view learning. To facilitate the understanding of COCLE, we describe both in detail below.

(1) *Intra-view learning*. In one view (i.e., G), given the label information (q, u) where $u \in pos(q)$, and their respective representations $\boldsymbol{h}(q)$ and $\boldsymbol{h}(u)$, it aims at maximizing the mutual information between $\boldsymbol{h}(q)$ and $\boldsymbol{h}(u)$. For instance, in Figure 3, in view G, given q=v_1, $pos(v_1)$=$\{v_2, v_3\}$, the intra-view learning makes the distance among these representations of nodes v_1, v_2, v_3 small. The intra-view learning takes into account the community information of query nodes, which is important to boost the effectiveness of the community search. In addition to the *low-order intra-view* learning in the original graph G, to utilize the high-order information, we propose *high-order intra-view* learning. It uses the similarity of the neighborhood structure of nodes to capture the high-order information. For example, in view G_{aug}, 1-hop of node v_1 and node v_2 have similar neighborhood structures.

(2) *Inter-view learning*. Let $\boldsymbol{h}(u)$ and $\boldsymbol{h}'(u)$ be the representations of node u in two views G and G_{aug}, respectively. Note that here, $\boldsymbol{h}(u)$ and $\boldsymbol{h}'(u)$ are obtained by a shared encoder. It aims at extracting information across these two views by exploring two representations, $\boldsymbol{h}(u)$ and $\boldsymbol{h}'(u)$. According to the general framework of contrastive learning, the representations $\boldsymbol{h}(q_i)$ and $\boldsymbol{h}'(q_i)$ of the query node q_i compose the positive pairs. For example, in Figure 3, given q=v_1, the blue nodes v_1 in G and the 1-hop set $\{v_1, v_2, v_3, v_4\}$ of v_1 in G_{aug} compose the positive pairs. If directly applying the previous contrastive learning method, the community label information across the views is ignored, outputting representations that do not carry the community information. Different from the classic contrastive learning, we extend the scope of positive pairs by including (p, q), where $q \in pos(p)$. Particularly, we also aim to reduce the distance between $\boldsymbol{h}(p)$ and $\boldsymbol{h}'(q)$. For instance, in Figure 3, given q=v_1, $pos(v_1)$=$\{v_2, v_3\}$, the blue node v_1 in G and green nodes $\{v_1, v_2, v_3\}$ and orange nodes $\{v_1, v_2, v_3, v_4, v_5\}$ in G_{aug} compose the inter-view learning, and vice versa.

V. Components of COCLE

A. Node Feature Encoder

As community search is a query-dependent task, for different queries, there will be different initial node features and the model will output specific representations. Since query node information and structure information have different influences for community search in different datasets, to effectively utilize the two information, we propose using an attention-based model to dynamically learn how to fuse the two information and generate the node representations. Graph Convolutional Network (GCN) [38] has shown great potential in encoding structure information. We first use GCN to encode query node and structure for the original graph G, respectively. Then we use the graph convolution operator to fuse them by learning the attention coefficients of the query node and structure.

In a GCN with L layers, its l-th layer receives the features from the previous layer and updates it by aggregating features from the neighborhood. The propagation rule is as follows:

$$\boldsymbol{H}^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(l)}\boldsymbol{\Theta}^{(l)}). \quad (1)$$

Here, $\hat{\boldsymbol{A}}$ is the adjacency matrix with self-connections, i.e., $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$. $\hat{\boldsymbol{D}}$ is the degree diagonal matrix with $\hat{\boldsymbol{D}}_{ii} = \sum_j \hat{\boldsymbol{A}}_{ij}$. $\boldsymbol{\Theta}^{(l)} \in \boldsymbol{\mathcal{R}}^{d(l) \times d(l+1)}$ as the trainable weight matrix at the l-th layer, $d(l)$ is the dimension of l-th layer and $\sigma(\cdot)$ is the activation function. $\boldsymbol{H}^{(l)} \in \boldsymbol{\mathcal{R}}^{n \times d(l)}$ is the hidden node representation learned by GCN in the l-th layer. $\boldsymbol{H}^0$ is the initial representation of all nodes.

Selection of $\boldsymbol{H}_0$. To encode the query node, we use the following definition to initialize the node features, $\boldsymbol{H}_q^{(0)}$=$\boldsymbol{X}_q$. Matrix $\boldsymbol{X}_q \in \boldsymbol{\mathcal{R}}^{n \times 1}$ is composed by $\boldsymbol{x}_q[u]$, $\forall u \in V$.

$$\boldsymbol{x}_q[u] = \begin{cases} 1, & u = q, \\ 0, & u \neq q. \end{cases} \quad (2)$$

To encode the structure information of the graph, we use $\boldsymbol{H}_s^{(0)}$=$\boldsymbol{X}_s$, where $\boldsymbol{X}_s \in \boldsymbol{\mathcal{R}}^{n\times d}$ is the feature matrix. In this paper, we focus on simple undirected graphs without features. The core number could capture the cohesiveness structure of the graph, and communities are usually composed of cohesively connected nodes. We use the normalized core number of all nodes to compose the feature matrix $\boldsymbol{X}_s$, where the core number of a node u is the maximum k that a k-core contains u [23]. If the graph has the original features, we can simply concatenate the original features and the normalized core number as the initial feature matrix (i.e. $\boldsymbol{H}_s^{(0)}$ = $[\boldsymbol{F}||\boldsymbol{X}_s]$, $\boldsymbol{F}$ is the original feature matrix). Then these two encoders can be defined as

$$\boldsymbol{H}_q^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_q^{(l)}\Theta_q^{(l)}),$$

$$\boldsymbol{H}_s^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_s^{(l)}\Theta_s^{(l)}).$$

In each layer of these two encoders, given $\boldsymbol{h}_q^{(l)}(u)$ and $\boldsymbol{h}_s^{(l)}(u)$, we use Eq.(3) to learn the attentions and fuse them. $\boldsymbol{a}_q^{(l)}$ and $\boldsymbol{a}_s^{(l)}$ are the trainable attention vector.

$$\begin{aligned}
&\boldsymbol{h}_f^{(l)}(u) = \tau_q^{(l)}(u)\boldsymbol{h}_q^{(l)}(u) + \tau_s^{(l)}(u)\boldsymbol{h}_s^{(l)}(u) \\
&\tau_q^{(l)}(u) = \frac{exp((\boldsymbol{a}_q^{(l)})^T\boldsymbol{h}_q^{(l)}(u))}{exp((\boldsymbol{a}_q^{(l)})^T\boldsymbol{h}_q^{(l)}(u)) + exp((\boldsymbol{a}_s^{(l)})^T\boldsymbol{h}_s^{(l)}(u))} \\
&\tau_s^{(l)}(u) = \frac{exp((\boldsymbol{a}_s^{(l)})^T\boldsymbol{h}_s^{(l)}(u))}{exp((\boldsymbol{a}_q^{(l)})^T\boldsymbol{h}_q^{(l)}(u)) + exp((\boldsymbol{a}_s^{(l)})^T\boldsymbol{h}_s^{(l)}(u))}.
\end{aligned} \qquad (3)$$

In each layer, $\boldsymbol{h}_q^{(l)}(u)$ and $\boldsymbol{h}_s^{(l)}(u)$ may have different influence, we fuse them in each layer and use Eq.(4) to get the fused representations, where $\bar{\boldsymbol{H}}_q^{(0)} = \boldsymbol{X}_q\boldsymbol{W}_q + \boldsymbol{b}_q$, $\bar{\boldsymbol{H}}_s^{(0)} = \boldsymbol{X}_s\boldsymbol{W}_s + \boldsymbol{b}_s$. $\boldsymbol{W}_q \in \boldsymbol{\mathcal{R}}^{1\times d(0)}$, $\boldsymbol{b}_q \in \boldsymbol{\mathcal{R}}^{n\times d(0)}$, $\boldsymbol{W}_s \in \boldsymbol{\mathcal{R}}^{d\times d(0)}$, $\boldsymbol{b}_s \in \boldsymbol{\mathcal{R}}^{n\times d(0)}$ are used to map $\boldsymbol{X}_q$ and $\boldsymbol{X}_s$ with same dimension. By doing so, we use the l-layer $\boldsymbol{H}_f^{(l)}$ as the final representation. It is also termed as $\boldsymbol{H}$.

$$\begin{aligned}
&\boldsymbol{H}_f^{(l+1)} = \sigma(\boldsymbol{\tau}_q^{(l+1)}\boldsymbol{H}_q^{(l+1)} + \boldsymbol{\tau}_s^{(l+1)}\boldsymbol{H}_s^{(l+1)} + \hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_f^{(l)}\Theta_f^{(l)}) \\
&\boldsymbol{H}_f^{(0)} = \boldsymbol{\tau}_q^{(0)}\bar{\boldsymbol{H}}_q^{(0)} + \boldsymbol{\tau}_s^{(0)}\bar{\boldsymbol{H}}_s^{(0)}.
\end{aligned} \qquad (4)$$

Following the existing contrastive learning framework [35], after obtaining the node level representation $\boldsymbol{H}$, we use MLP (Multilayer Perceptron) with one hidden layer to get the vector space for contrastive learning. Particularly, we use the following transformation:

$$\boldsymbol{H} = \sigma(\boldsymbol{H}\boldsymbol{\theta}_1 + \boldsymbol{b}_1)\boldsymbol{\theta}_2 + \boldsymbol{b}_2 \qquad (5)$$

where $\sigma(\cdot)$ denotes the activation function, e.g., ReLU. $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are the weight parameter matrices. $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are the bias.

B. High-order Augmentation Encoder

Existing augmentation methods are not suitable for community search for the following reasons:

(1) The relationships among nodes are important. Randomly removing nodes or edges will lead to great changes in the relationships. As shown in Figure 2, if we remove v_3 from the graph, the density of the whole graph is greatly reduced. If we want to find the community containing v_7 in this graph, nodes $\{v_4, v_5, v_6\}$ may not be in the community as these nodes and v_7 have lower density. But in the original graph, these nodes may belong to the community C_{v_7} since nodes $\{v_7, v_1, v_2, v_3, v_4, v_5, v_6\}$ compose a dense structure. This will further affect the performance of the learned model.

(2) More importantly, existing graph augmentation methods usually do not integrate the high-order graph structure. Similar to the aforementioned graph convolutional network, existing methods aggregate for each node the node features of all its 1-hop neighbors. Since the neighborship is only the relationship between a pair of nodes, the aggregation in this way can only capture the low-order information between nodes. Instead, the hypergraph-based convolutional network aggregates the related hyperedge features for each node. Each hyperedge corresponds to a subset of nodes, and essentially represents the overall relationship of all nodes within the subset. Thus hypergraph can model the high-order relationships.

To capture the high-order structure information, we propose a hypergraph-based method that utilizes the idea of neighborhood structure similarity. The underlying assumption is that the nodes within the same community have similar subgraph context induced by the same community. To achieve this goal, we see the neighborhood-induced subgraphs as hyperedges, exploiting the idea of hypergraph neural network (HGNN) to learn the representation of subgraphs. It is important to note that not all the nodes in the neighborhood of a node u have equal importance as some nodes in the neighborhood may not be in the community containing u. Thus, we propose using an attention-based model to fuse the neighborhood information.

A hypergraph is represented by $G_{\mathcal{H}}(V_{\mathcal{H}}, E_{\mathcal{H}}, \boldsymbol{W}_{\mathcal{H}})$, where $V_{\mathcal{H}}$ and $E_{\mathcal{H}}$ are the sets of nodes and hyperedges, respectively, and $\boldsymbol{W}_{\mathcal{H}} \in \boldsymbol{\mathcal{R}}^{|E_{\mathcal{H}}|\times|E_{\mathcal{H}}|}$ is a diagonal matrix with the hyperedge weight. Each hyperedge $e \in E_{\mathcal{H}}$ is a non-empty subset of $V_{\mathcal{H}}$. Generally, hypergraph $G_{\mathcal{H}}$ can be represented by an incidence matrix $\boldsymbol{\mathcal{H}} \in \boldsymbol{\mathcal{R}}^{|V_{\mathcal{H}}|\times|E_{\mathcal{H}}|}$, where $\boldsymbol{\mathcal{H}}_{i,j}$=1 if $v_i \in e_j$. Then, the node degree and hyperedge degree are defined as:

$$\boldsymbol{D}_{ii}^{v} = \sum_{j=1}^{|E_{\mathcal{H}}|}\boldsymbol{W}_{\mathcal{H}}(j,j)\boldsymbol{\mathcal{H}}_{i,j}, \qquad \boldsymbol{D}_{jj}^{e} = \sum_{i=1}^{|V_{\mathcal{H}}|}\boldsymbol{\mathcal{H}}_{i,j}. \qquad (6)$$

The diagonal matrices of node and hyperedge degrees are denoted as $\boldsymbol{D}_v$, $\boldsymbol{D}_e$, respectively.

To construct the hypergraph, we employ a simple yet effective approach, based on a common observation in community search that the nodes in the community are generally densely connected and located around the query node [4], [24], i.e., nodes in the community of q are not far away from the query node q. In particular, for each node in graph G, we search for the r-hop neighboring nodes rooted at every node, with these nodes corresponding to a hyperedge. As such, there will be a total of $|V|$ hyperedges. We use e_u to denote the hyperedge that is constructed for node u.

To encode the nodes in the constructed hypergraph, we use the existing hypergraph neural network [39], which has shown better performance than GCN when encoding high-order relationships in graphs. Given a L-layer HGNN, its l-th layer receives the features from the previous layer and updates it by aggregating features from the neighborhood. Here, the neighbor of a node u is the node that is located in the same hyperedges as u. The hypergraph convolution is defined as:

$$\boldsymbol{Z}^{(l+1)} = \sigma(\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}^{(l)}\hat{\boldsymbol{\Theta}}^{(l)}), \qquad (7)$$

where $\boldsymbol{Z}^{(l)}$ is the hidden representation in the l-th layer, $\sigma(\cdot)$ is the activation function. $\hat{\boldsymbol{\Theta}}^{(l)}$ is a trainable parameter matrix. We also use two HGNN-based encoders to encode the query node and graph structure, i.e., $\boldsymbol{Z}_q^{(l+1)} = \sigma(\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}_q^{(l)}\hat{\boldsymbol{\Theta}}_q^{(l)})$, $\boldsymbol{Z}_s^{(l+1)} = \sigma(\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}_s^{(l)}\hat{\boldsymbol{\Theta}}_s^{(l)})$, where $\boldsymbol{Z}_q^{(0)}$=$\boldsymbol{X}_q$, $\boldsymbol{Z}_s^{(0)}$=$\boldsymbol{X}_s$. Similarly, we also use above mentioned attention-based method to fuse $\boldsymbol{Z}_q^{(l)}$ and $\boldsymbol{Z}_s^{(l)}$ and get $\boldsymbol{Z}_f^{(l)}$.

Note that we aim to capture the information of different high-order structures, and each hyperedge corresponds to a subgraph. Hence, we need a readout function (it is a function to get the feature representation of the whole graph by aggregating the node features.) to aggregate the node representations in each hyperedge. We use node similarity attention to measure the weight of nodes in each hyperedge. Then, the hyperedge readout function $\boldsymbol{\mathcal{P}}(\cdot)$ is defined as follows:

$$\boldsymbol{\mathcal{P}}(e_u) = \sum_{v \in e_u} \beta_v \cdot \boldsymbol{z}(v), \tag{8}$$

where e_u is the hyperedge that is constructed for node u, and $\boldsymbol{z}(u)$ is the representation of node u in the last layer of HGNN, namely $\boldsymbol{Z}_f^{(l)}$. β_v weighs the importance of each node in the hyperedge, which is calculated as follows:

$$\beta_v = \frac{exp(sim(\boldsymbol{z}(u), \boldsymbol{z}(v)))}{\sum_{w \in e_u} exp(sim(\boldsymbol{z}(u), \boldsymbol{z}(w)))}, \tag{9}$$

where $sim(u, v)$ denotes the cosine similarity between two representation vectors u and v, which is defined as follows:

$$sim(\boldsymbol{z}(u), \boldsymbol{z}(v)) = \frac{\boldsymbol{z}(u)^T \boldsymbol{z}(v)}{\|\boldsymbol{z}(u)\| \|\boldsymbol{z}(v)\|}. \tag{10}$$

The augmentation represetation matrix $\boldsymbol{H}'$ is composed by all hyperedge representation $\boldsymbol{\mathcal{P}}(e_u)$, $\forall u \in V$.

Similar to that mentioned earlier, after obtaining $\boldsymbol{H}'$, we use MLP (Multilayer Perceptron) to get the vector space for contrastive learning. Note that the MLP used here has the same parameters as the one mentioned above.

C. Label-Aware Contrastive Learner

To capture the information contained in the ground-truth community, we propose a label-aware contrastive learner, incorporating the community label information into the contrastive loss. In addition to the community information in each view, the information contained in the data itself (i.e., the similarity between the same nodes and similarity between the query node and its positive samples in different views) is also helpful for model training. We also add such information from different views into the contrastive loss. In a nutshell, we redesign the scope of positive pairs and negative pairs and the contrastive loss to conduct the contrastive learning, which includes low-order intra-view, high-order intra-view and low-high-order inter-view learning.

Given a query task with positive samples $(q_i, pos(q_i))$, we see the query node q_i and each node from the positive samples as the positive pair, while the node q_i and other nodes compose the negative pairs. Then the intra-view contrastive loss in the original graph view is defined as follows:

$$\boldsymbol{\mathcal{L}}_{intra}(q_i, G) = \frac{1}{|pos(q_i)|} \sum_{u \in pos(q_i)} I(\boldsymbol{h}(q_i), \boldsymbol{h}(u)), \tag{11}$$

where

$$I(\boldsymbol{h}(q_i), \boldsymbol{h}(u)) = -\log \frac{exp(sim(\boldsymbol{h}(q_i), \boldsymbol{h}(u))/\tau)}{\sum_{v \in V} exp(sim(\boldsymbol{h}(q_i), \boldsymbol{h}(v))/\tau)}. \tag{12}$$

$\boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}(u))$ measures the representation similarity between query node q_i and its positive sample $u \in pos(q_i)$ in one view. The $sim(\cdot)$ denotes the cosine similarity. τ is the temperature parameter. If τ is set to small, the model more focuses on separating the negative samples that are most similar to the query node [40]. Different from the previous contrastive loss, which focuses on maximizing mutual information between representations of the same node, we mainly concentrate on the relationships between the query node and its positive samples. For the hypergraph view, we use a similar equation, but the representations are replaced by hyperedge representations.

Given two views, we see the representation of node q_i in one view and the representation of node u ($u \in pos(q_i)$) from another view as the positive pair, while the representation of node q_i in one view and the representation of node v ($u \notin pos(q_i)$) from another view compose the negative pair. Then the inter-view contrastive learning is defined as follows:

$$\boldsymbol{\mathcal{L}}_{inter}(q_i) = \frac{1}{2}\frac{1}{|pos(q_i)|} \sum_{u \in pos(q_i)} \left[\boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}'(u) + \boldsymbol{I}(\boldsymbol{h}'(q_i), \boldsymbol{h}(u))\right], \tag{13}$$

where $\boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}'(u))$ or $\boldsymbol{I}(\boldsymbol{h}'(q_i), \boldsymbol{h}(u))$ measures the representation similarity between query node q_i in one view and its positive node u in another view. They are defined as follows:

$$\begin{aligned} \boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}'(u)) &= -\log \frac{exp(sim(\boldsymbol{h}(q_i), \boldsymbol{h}'(u))/\tau)}{\sum_{v \in V} exp(sim(\boldsymbol{h}(q_i), \boldsymbol{h}'(v))/\tau)}, \\ \boldsymbol{I}(\boldsymbol{h}'(q_i), \boldsymbol{h}(u)) &= -\log \frac{exp(sim(\boldsymbol{h}'(q_i), \boldsymbol{h}(u))/\tau)}{\sum_{v \in V} exp(sim(\boldsymbol{h}'(q_i), \boldsymbol{h}(v))/\tau)}. \end{aligned} \tag{14}$$

Given the training dataset $\mathcal{D}$, the overall contrastive learning loss is given as follows:

$$\begin{aligned} \boldsymbol{\mathcal{L}}_s &= \sum_{(q_i, pos(q_i)) \in \mathcal{D}} \boldsymbol{\mathcal{L}}_s(q_i, pos(q_i)) \\ &= \sum_{(q_i, pos(q_i)) \in \mathcal{D}} \frac{1}{2}\left(\boldsymbol{\mathcal{L}}_{intra}(q_i, G) + \boldsymbol{\mathcal{L}}_{intra}(q_i, G_{aug})\right) + \lambda \boldsymbol{\mathcal{L}}_{inter}(q_i), \end{aligned} \tag{15}$$

where $\boldsymbol{\mathcal{L}}_{intra}(q_i, G)$ and $\boldsymbol{\mathcal{L}}_{intra}(q_i, G_{aug})$ denote the intra-view contrastive loss in the original graph G and the augmentation graph G_{aug}, respectively. $\boldsymbol{\mathcal{L}}_{inter}(q_i)$ is the inter-view contrastive loss. λ denotes the weight of inter-view contrastive loss. Since the representations of node q_i from two views are also helpful for training, we see these two representations of node q_i as positive pairs, the representation of node q_i in one view and the representations of other nodes in another view as negative pairs. Then we add an unsupervised comparative loss, which is defined as follows:

$$\boldsymbol{\mathcal{L}}_u = \sum_{(q_i \in \mathcal{D})} \boldsymbol{\mathcal{L}}_u(q_i) = \sum_{(q_i \in \mathcal{D})} \frac{1}{2}\left[\boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}'(q_i)) + \boldsymbol{I}(\boldsymbol{h}'(q_i), \boldsymbol{h}(q_i))\right], \tag{16}$$

where $\boldsymbol{I}(\boldsymbol{h}(q_i), \boldsymbol{h}'(q_i))$ and $\boldsymbol{I}(\boldsymbol{h}'(q_i), \boldsymbol{h}(q_i))$ measure the representation similarity of the query node q_i between two views.

Then the final loss function is defined as follows:

$$\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{L}}_s + \alpha \boldsymbol{\mathcal{L}}_u, \tag{17}$$

where α is weight of unsupervised loss. We note that α and λ are hyperparameters inherent to machine learning models, and we have evaluated their sensitivities in the experiments.

D. Algorithms

The COCLE training process is shown in Algorithm 1. In the loop, it first uses the proposed model to encode the nodes (Lines 6-13). Then, the hyperedge representations are obtained by $\boldsymbol{\mathcal{P}}(\cdot)$ (Line 14). After that, two sets of representations are transformed by a shared MLP (Line 15). Finally, the loss is computed (Lines 16-17) and parameters are updated (Line 18).

Algorithm 1: The COCLE training

```
Input: Graph $G$, training set $\mathcal{D}$, parameters $\tau$, $\lambda$, $\alpha$, $r$, $L$.
Output: parameters of model COCLE.
1  Build augmentation graph $G_{aug}$ with $r$-hop neighbors;
2  while not converge do
3      $\boldsymbol{\mathcal{L}}$=0;
4      for $(q_i, pos(q_i)) \in \mathcal{D}$ do
5          $l$=0;
6          while $l < L$ do
7              $\boldsymbol{H}_q^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_q^{(l)}\boldsymbol{\Theta}_q^{(l)})$;
8              $\boldsymbol{H}_s^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_s^{(l)}\boldsymbol{\Theta}_s^{(l)})$;
9              $\boldsymbol{H}_f^{(l+1)} = \sigma(\boldsymbol{\tau}_q^{(l+1)}\boldsymbol{H}_q^{(l+1)} + \boldsymbol{\tau}_s^{(l+1)}\boldsymbol{H}_s^{(l+1)} +$
                   $\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_f^{(l)}\boldsymbol{\Theta}_f^{(l)})$;
10             $\boldsymbol{Z}_q^{(l+1)} = \sigma(\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}_q^{(l)}\hat{\boldsymbol{\Theta}}_q^{(l)})$;
11             $\boldsymbol{Z}_s^{(l+1)} = \sigma(\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}_s^{(l)}\hat{\boldsymbol{\Theta}}_s^{(l)})$;
12             $\boldsymbol{Z}_f^{(l+1)} = \sigma(\hat{\boldsymbol{\tau}}_q^{(l+1)}\boldsymbol{Z}_q^{(l+1)} + \hat{\boldsymbol{\tau}}_s^{(l+1)}\boldsymbol{Z}_s^{(l+1)} +$
                   $\boldsymbol{D}_v^{-1}\boldsymbol{\mathcal{H}}\boldsymbol{W}_{\mathcal{H}}\boldsymbol{D}_e^{-1}\boldsymbol{\mathcal{H}}^T\boldsymbol{Z}_f^{(l)}\hat{\boldsymbol{\Theta}}_f^{(l)})$;
13             $l$=$l$+1;
14         $\boldsymbol{H} \leftarrow \boldsymbol{H}_f^{(L)}$,$\boldsymbol{H}' \leftarrow \boldsymbol{\mathcal{P}}(\boldsymbol{Z}_f^{(L)})$;
15         $\boldsymbol{H} = \sigma(\boldsymbol{H}\boldsymbol{\theta}_1 + \boldsymbol{b}_1)\boldsymbol{\theta}_2 + \boldsymbol{b}_2, \boldsymbol{H}' =$
             $\sigma(\boldsymbol{H}'\boldsymbol{\theta}_1 + \boldsymbol{b}_1)\boldsymbol{\theta}_2 + \boldsymbol{b}_2$;
16         compute the loss $\boldsymbol{\mathcal{L}}(q_i)$=$\boldsymbol{\mathcal{L}}_s(q_i)$+$\alpha\boldsymbol{\mathcal{L}}_u(q_i)$;
17         $\boldsymbol{\mathcal{L}}$ = $\boldsymbol{\mathcal{L}}$+$\boldsymbol{\mathcal{L}}(q_i)$;
18     Update parameters of model COCLE by using gradient
         descent to minimize $\boldsymbol{\mathcal{L}}$;
19 return parameters of model COCLE;
```

Algorithm 2: Community Search

```
Input: COCLE model, query node $q$, threshold $p$, $L$.
Output: Community $C_q$.
1  $l$=0;
2  while $l < L$ do
3      $\boldsymbol{H}_q^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_q^{(l)}\boldsymbol{\Theta}_q^{(l)})$;
4      $\boldsymbol{H}_s^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_s^{(l)}\boldsymbol{\Theta}_s^{(l)})$;
5      $\boldsymbol{H}_f^{(l+1)} = \sigma(\boldsymbol{\tau}_q^{(l+1)}\boldsymbol{H}_q^{(l+1)} + \boldsymbol{\tau}_s^{(l+1)}\boldsymbol{H}_s^{(l+1)} +$
           $\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}_f^{(l)}\boldsymbol{\Theta}_f^{(l)})$;
6      $l$=$l$+1;
7  $\boldsymbol{H} = \sigma(\boldsymbol{H}_f^{(L)}\boldsymbol{\theta}_1 + \boldsymbol{b}_1)\boldsymbol{\theta}_2 + \boldsymbol{b}_2$;
8  $p(u|q)$=$sigmoid(sim(\boldsymbol{h}(q), \boldsymbol{h}(u)))$, $\forall u \in V$;
9  $C_q$ = $\{u|p(u|q) \geq p\}$;
10 return $C_q$;
```

Given the new query node q, we use the trained model to encode the nodes in the graph. Algorithm 2 describes the community search method. When using the model to encode the nodes in the graph G for query node q (Lines 1-6), the distance of representations between the nodes in C_q and query node q becomes smaller than that of nodes in $V \setminus C_q$ and q. Then, we use the Eq.(18) to obtain the probability of whether a node belongs to community C_q (Line 8):

$$p(u|q) = sigmoid(sim(\boldsymbol{h}(u), \boldsymbol{h}(q))), \tag{18}$$

where $sim(\cdot)$ measures the similarity between query node q and other nodes, and $sigmoid(\cdot)$ is an activation function that maps the similarity into [0,1]. Then community C_q is composed of the nodes whose probability is larger than the given threshold p (Line 9).

VI. COCLEP: Optimization by Graph Partitioning

Although the proposed Algorithm 1 can effectively answer the community search query, it needs at least $O((m+ndL)|\mathcal{D}|)$ space in training process, where m is the number of edges, n is the number of nodes, d is the dimension, L is the number of GNNs layers, and $|D|$ is the training dataset size. It may easily run out of memory on GPU when the graph is very large. Besides, Algorithm 1 needs $O(mL|\mathcal{D}|)$ time to train a model in each iteration. When m is large, the training cost becomes very high. The details for complexity analysis of Algorithm 1 is shown in the last part of this section.

The major challenge for model training on large graphs is that the graph convolution operator needs to iteratively aggregate the neighbors information for all nodes in the graph. Thus, all the node features are needed to store on GPU. When the graph is large, it leads to out of the memory. Since the community is generally densely connected and community nodes locate around the query node, the training based on the entire graph may not be necessary. We can train the model by identifying a moderate-sized subgraph that contains the query node and conducting the training mainly on the subgraph. How to find such a possible subgraph is important as it can influence the effectiveness of the trained model.

Our goal is to find a partition such that the obtained node representations are close to the node representations trained from the original graph. Let G' is the partitioned graph that removes some edges from G, $\boldsymbol{h}(u)$ and $\hat{\boldsymbol{h}}(u)$ denote the representation of node u obtained from the original graph and partitioned graph respectively. We use the L2 norm to measure distance between $\boldsymbol{h}(u)$ and $\hat{\boldsymbol{h}}(u)$, i.e., $dist(\boldsymbol{h}(u), \hat{\boldsymbol{h}}(u)) = \|\boldsymbol{h}(u) - \hat{\boldsymbol{h}}(u)\|_2 = \sqrt{\sum_i(\boldsymbol{h}(u)_i - \hat{\boldsymbol{h}}(u)_i)^2}$, where $\boldsymbol{h}(u)_i$ and $\hat{\boldsymbol{h}}(u)_i$ is the element on the vector $\boldsymbol{h}(u)$ and $\hat{\boldsymbol{h}}(u)$ respectively.

Analysis. We observed that if $\|\boldsymbol{h}(u) - \hat{\boldsymbol{h}}(u)\|_2$ is small for all $u \in V$, $sim(\hat{\boldsymbol{h}}(q), \hat{\boldsymbol{h}}(u))$ is close to $sim(\boldsymbol{h}(q), \boldsymbol{h}(u))$. We refer interested readers to Lemma 1 and Lemma 2 in our technical report [41] for more details. Thus, to guarantee that the result from the partitioned graph is close to that from the original graph, we should minimize the upper bound of $\|\boldsymbol{h}_u^{(l)} - \hat{\boldsymbol{h}}_u^{(l)}\|_2$. Let $B(G')$=$\{u|u \in V, s.t. \exists v \in V, (u,v) \in E(G), (u,v) \notin E(G')\}$, $N(u) = N(u,G) = \{v|v \in V, s.t. \exists (u,v) \in E(G)\} \cup \{u\}$, $N(u)'$=$N(u,G')$-$B(G')$, and $N(u)''$=$N(u,G')$-$N(u)'$. We denote $\sigma(\cdot)$ is the ReLU activation function. In the following, unless otherwise stated, $h_u^{(l)}$ denotes the row vector.

In the l-th layer of the graph convolutional operator, we have

$$\boldsymbol{h}_u^{(l)} = \sigma\left(\sum_{v \in N(u)} \alpha_{v,u}\boldsymbol{h}_v^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right) = \sigma(\bar{\boldsymbol{h}}_u^{(l)} + \boldsymbol{\beta}_u^{(l)}),$$

where $\bar{\boldsymbol{h}}_u^{(l)} = \sum_{v \in N(u)'} \alpha_{v,u}\boldsymbol{h}_v^{(l-1)}\boldsymbol{\Theta}^{(l-1)}$, $\boldsymbol{\beta}_u^{(l)} = \sum_{v \in (N(u)-N(u)')} \alpha_{v,u}\boldsymbol{h}_v^{(l-1)}\boldsymbol{\Theta}^{(l-1)}$, $\alpha_{v,u} = \frac{1}{\sqrt{1+d_v}\sqrt{1+d_u}}$ which is an element of a normalized relation matrix. d_v and d_u are the degrees of node v and node u.

In the partitioned graph G',

$$\hat{\boldsymbol{h}}_u^{(l)} = \sigma\left(\sum_{v \in N(u)' \cup N(u)''} \hat{\alpha}_{v,u}\hat{\boldsymbol{h}}_v^{(l-1)}\boldsymbol{\Theta}^{(l-1)}\right) = \sigma(\tilde{\boldsymbol{h}}_u^{(l)} + \boldsymbol{\gamma}_u^{(l)}),$$

```
Algorithm 3: The COCLEP training
Input: Graph $G$, training dataset $\mathcal{D}$, parameters $\tau$, $\lambda$, $\alpha$ $r$,
       number of partitions $n_c$.
Output: parameters of model COCLEP.
1  $\hat{\mathcal{C}}$={$\hat{C}_1$, $\hat{C}_2$, ..., $\hat{C}_{n_c}$}←partition graph into $n_c$ clusters;
2  for $(q_i, pos(q_i)) \in \mathcal{D}$ do
3      $G_{q_i} \leftarrow G(\hat{C}_{q_i} \cup \hat{C}^b_{q_i})$;
4      Construct the augmentation graph $G_{aug,q_i}$ for $G_{q_i}$ with
        $r$-hop neighbors;
5      $\hat{\mathcal{D}} = \hat{\mathcal{D}} \cup (q_i, pos(q_i), G_{q_i}, G_{aug,q_i})$;
6  while not converge do
7      $\mathcal{L}$=0;
8      for $(q_i, pos(q_i), G_{q_i}, G_{aug,q_i}) \in \hat{\mathcal{D}}$ do
9          Algorithm 1 lines 5-17;
10     Algorithm 1 line 18;
11 return parameters of model COCLEP;
```

where $\tilde{\boldsymbol{h}}_u^{(l)} = \sum_{v \in N(u)'} \alpha_{v,u} \hat{\boldsymbol{h}}_v^{(l-1)} \boldsymbol{\Theta}^{(l-1)}$, $\boldsymbol{\gamma}_u^{(l)} = \sum_{v \in N(u)''} \hat{\alpha}_{v,u} \hat{\boldsymbol{h}}_v^{(l-1)} \boldsymbol{\Theta}^{(l-1)}$, $\hat{\alpha}_{v,u}$ is an element of a normalized relation matrix in the partitioned graph.

Theorem 1: $\|\boldsymbol{h}_u^{(l)} - \hat{\boldsymbol{h}}_u^{(l)}\|_2 \leq C \max_{v \in V} \{\|\boldsymbol{h}_v^{(l-1)} - \hat{\boldsymbol{h}}_v^{(l-1)}\|_F\}$ $\|\boldsymbol{\Theta}^{(l-1)}\|_F + B_1 + B_2$, where $C = \max_{v \in V}\{deg(v)\}$, $B_1 = \sum_{v \in N(u)-N(u)'} \alpha_{v,u} \|\boldsymbol{h}_v^{(l-1)}\|_F \|\boldsymbol{\Theta}^{(l-1)}\|_F$, $B_2 = \sum_{v \in N(u)''}$ $\hat{\alpha}_{v,u} \|\hat{\boldsymbol{h}}_v^{(l-1)}\|_F \|\boldsymbol{\Theta}^{(l-1)}\|_F$. $\|\cdot\|_F$ is the Frobenius norm.

By Theorem 1, the representations are related to the nodes that are cut. If the cut is minimized, the set $N(u) - N(u, G')$ and $N(u)''$ will be small, which will decrease the upper bound of $\|\boldsymbol{h}_u^{(l)} - \hat{\boldsymbol{h}}_u^{(l)}\|_2$. Thus, we can partition the graph by using the min-cut (e.g., METIS [42]). We term our method as COCLEP. Min-cut guarantees that the links inside the clusters are dense and the links among clusters are sparse. According to our experiments, such high-quality partition even helps in locating communities because communities are also densely connected internally. With the partitioned graphs, we can further reduce the unnecessary influence from the irrelevant nodes. As the graph is partitioned into several disjoint subgraphs, each query node in the training dataset will belong to only one cluster. Then, we can perform the model training on the subgraph that contains the specific query node. As such, the final contrastive loss function can be redefined as follows:

$$\mathcal{L}_C = \sum_{(q_i, pos(q_i)) \in \mathcal{D}} (\mathcal{L}_s(q_i, pos(q_i), G_{q_i}) + \alpha \mathcal{L}_u(q_i, pos(q_i), G_{q_i})),$$

where G_{q_i} denotes the subgraph containing query q_i. $\mathcal{L}_s(q_i, pos(q_i), G_{q_i})$ and $\mathcal{L}_u(q_i, pos(q_i), G_{q_i})$ are the semi-supervised contrastive learning loss and unsupervised contrastive learning loss for the given query task $(q_i, pos(q_i))$ in G_{q_i}, respectively.

Subgraph Community Rejoining. As the graph is partitioned into disjoint partitions, nodes that belong to the same community may be divided into different partitions. To solve this problem, for each partition, we add the nodes that are located at the boundary of the partition and are the neighbors of the query node q_i. Let $\hat{C}_{q_i}$ be one of the partitions, where the boundary is defined as $\hat{C}^b_{q_i}$={$u \mid u \in (V \setminus \hat{C}_{q_i}) \cap N(q_i)$}. In other words, the boundary of a partition contains the nodes that can link to other partitions. Then, the subgraph for a query node q_i is the subgraph induced by $\hat{C}_{q_i}$ and $\hat{C}^b_{q_i}$, i.e., $V_{q_i} = \hat{C}_{q_i} \cup \hat{C}^b_{q_i}$. By doing so, if a node u in $\hat{C}^b_{q_i}$ belongs to C_{q_i} with a high probability, we regard node q_i as the query node in the another partition to search for the community. Then, we can include more nodes that originally belonged to the same community but were partitioned into different partitions.

```
Algorithm 4: Partition-based Community Search
Input: COCLEP model, query node $q$, threshold $p$, clusters
       $\hat{\mathcal{C}}$ of $G$.
Output: Community $C_q$.
1  $G_q \leftarrow G(\hat{C}_q \cup \hat{C}^b_q)$;
2  $\boldsymbol{H}$ ←Algorithm 2 lines 1-6;
3  $p(u|q)$=$sigmoid(sim(\boldsymbol{h}_G(q), \boldsymbol{h}_G(u)))$, $\forall u \in V_{G_q}$;
4  $C_q$ = {$u | p(u|q) \geq p$};
5  for $u \in \hat{C}^b_q$ do
6      if $p(u|q) \geq p$ then
7          $G_u \leftarrow G(\hat{C}_u \cup \hat{C}^b_u)$;
8          $\boldsymbol{H_u}$ ←Algorithm 2 lines 1-6;
9          $\boldsymbol{h}_u(q) = \boldsymbol{h}_u(q) + \boldsymbol{h}(q)$;
10         $p(v|q)$=$sigmoid(sim(\boldsymbol{h}_u(q), \boldsymbol{h}_u(v)))$, $\forall v \in V_{G_u}$;
11         $C_q$=$C_q \cup$ {$v | p(v|q) \geq p$};
12 return $C_q$;
```

The partition-based training algorithm and the community search algorithm are presented in Algorithm 3 and Algorithm 4, respectively. Algorithm 3 first partitions the graph into n_c partitions (Line 1). After that, it reconstructs the training dataset by getting the induced subgraph with nodes in $\hat{C}_{q_i}$ and $\hat{C}^b_{q_i}$ and builds the hypergraph (Lines 2-5). Then, it trains the model (Lines 6-10) similar to Algorithm 1. The difference is that it trains the model only based on partitioned graphs. Algorithm 4 shows the pseudo-code for community search using the partitions. It first obtains the community in G_q (Lines 1-4). Then it finds the community in other partitions (Line 5). If $p(u|q) \geq p$, it sees q as the query node in the graph G_u to search for the community C_q (Lines 5-11). To capture the information of node q in the G_q, we consider adding $\boldsymbol{h}(q)$ to $\boldsymbol{h}_u(q)$ (Line 9).

Complexity Analysis. In the following, we analyze the time and space complexity of our proposed algorithms to demonstrate the effectiveness of COCLEP.

Theorem 2: Let d be the dimension of node representations, L be the number of GNNs layers, $|\mathcal{D}|$ be number of query nodes, T be the number of iterations. The time complexity of Algorithm 1 depends on the size of r to augment the hypergraph, where it takes $O(mL|\mathcal{D}|T)$ time when r=1, $O(nm + n^2L|\mathcal{D}|T)$ time when r>1. The space complexity is $O((m + ndL)|\mathcal{D}|)$ when r=1, $O((n^2 + ndL)|\mathcal{D}|)$ when r>1.

Theorem 3: Algorithm 2 takes $O(mL)$ time and $O(m + ndL)$ space, where n is the number of nodes in G, m is the number of edges in G, L is the number of GNNs layers, and d is the dimension of node representations.

Theorem 4: Let n_{q_i} and m_{q_i} be the number of nodes and edges in the partition C_{q_i} that contains the query node q_i. d is the dimension of node representations. L is the number of GNNs layers, and T is the number of iterations. Depending on the size of r to augment the hypergraph, Algorithm 3 takes $O(m + \sum_{q_i \in \mathcal{D}} m_{q_i} LT)$ time when r=1, and $O(m + \sum_{q_i \in \mathcal{D}} n_{q_i} m_{q_i} + n_{q_i}^2 LT)$ time when r>1. The space complexity is $O(m + \sum_{q_i \in \mathcal{D}} m_{q_i} + n_{q_i} dL)$ when r=1 and $O(m + \sum_{q_i \in \mathcal{D}} n_{q_i}^2 + n_{q_i} dL)$ when r>1.

Theorem 5: Algorithm 4 takes $O(\sum_{u \in \hat{C}_q^b \cup \{q\}} m'_u L)$ time and $O(\sum_{u \in \hat{C}_q^b \cup \{q\}} m'_u + n'_u dL)$ space.

Since m_{q_i} and n_{q_i} are much smaller than m and n, Algorithm 3 is faster than Algorithm 1. Note that when the number of partitions n_c is large, the values of n_{q_i} and m_{q_i} become smaller, and the time complexity and space complexity are also lower.

VII. Experiments

This section presents the performance of COCLE and COCLEP in various aspects.

A. Experimental Setting

Datasets. We use eight real-life graphs with ground-truth communities in our experiments. Football can be found in http://www-personal.umich.edu/∼mejn/netdata/. Citeseer and Cora are from [43]. The Facebook dataset is from [44]. Amazon, DBLP, Youtube, and LiveJournal can be found in SNAP (http://snap.stanford.edu/data/). Table II shows the statistics of the datasets, which are ordered by the number of edges, where $|C|$ denotes the number of ground-truth communities in the datasets. Note that for the datasets from SNAP, we use the top-5000 ground-truth communities with the highest quality.

Algorithms. To evaluate the effectiveness and efficiency of COCLE and COCLEP, we compare the existing cohesiveness model-based methods, including K-core [1], EquiTruss [6], K-ECC [5], and CTC [4]. We also compare the learning-based methods ICS-GNN [11] and QD-GNN [12]. The implementation of all baseline algorithms are provided by the authors. We note that the QD-GNN can not be applied to large-scale datasets, we compare the QD-GNN-P, which applies the QD-GNN in our proposed partition-based framework.

For COCLE and COCLEP and all their variants, we set the learning rate to 0.001, the dropout ratio to 0.1, the batch size to 64. The number of layers L is 3. The dimension of hidden features is set to 256. For the r-hop neighbors, we use 1 by default. We implement our methods in PyTorch with PyTorch Geometric, and models are trained by Adam [45]. The L2 penalty for Adam is 0.0005. The number of epochs is set to 200. The parameter τ is 0.2. The α and λ mainly affect the precision and further affect the other metrics. When α and λ are increased, the precision is increased. We increase these two parameters from small to large to find the appropriate parameters. For Citeseer, α and λ are set to 0.0001 and 0.0001, respectively. For Cora, they are set to 0.001 and 0.001. For Youtube, they are set to 100 and 100. For other datasets, they are set to 0.2 and 0.2, respectively. The threshold p for determining whether a node belongs to the community is obtained by searching for the optimal precision or F1-score from the validation set. The search range is [0.1, 0.9] with a 0.05 step. For Football, Citeseer, Cora, Facebook, Amazon, DBLP, Youtube, LiveJournal, the default n_c are 2, 3, 3, 5, 100, 100, 300, 1000, respectively. Both ICS-GNN and QD-GNN need attributes to obtain the initial feature matrix, while we focus on simple graphs without attributes. Thus, we use the normalized core number as the initial feature matrix for all ML-based community search methods. We use the default parameter setting in [11], [12], respectively. Note that ICS-GNN generates a subgraph with 400 nodes and then trains a model on this subgraph for each query node. We run all ML-based community search methods 5 times and report the average results with the standard deviation. We mainly use the DBLP and Youtube to evaluate the parameters influence and the components in our proposed model, most other datasets have the similar results with these two datasets. All experiments were conducted on a machine having 3.50GHz Intel(R) Xeon(R) CPU E5-2637, 128GB memory, and 2 NVIDIA RTX A4000 Graphics Cards (GPU) with 16GB memory.

TABLE II: Dataset statistics.

Data Set	|V|	|E|	|C|
Football	115	613	12
Citeseer	3,327	4,552	6
Cora	2,708	5,278	7
Facebook	3,622	72,964	130
Amazon	334,863	925,872	5,000
DBLP	317,080	1,049,866	5,000
Youtube	1,134,890	2,987,624	5,000
LiveJournal	3,997,962	34,681,189	5,000

Query and Data Generartion. To generate the training datasets, we randomly select a node from the ground-truth communities to generate the query node, and then randomly select 3 nodes from the same community as positive samples. For the validation sets and test sets, we only randomly select a node from the ground-truth communities to generate the query node. But the query tasks in the validation set contain ground-truth communities. To evaluate the effectiveness of models for query nodes in unseen communities, we divide all ground-truth communities into two groups to generate training tasks and test query tasks with a ratio of about 1: 4. Then we randomly select 0.25 of the ground-truth communities in the training set for validation task set generation. The training dataset, validation dataset, and test dataset contain 300, 100, and 500 query tasks, respectively. Since the number of ground-truth communities of Football, Citeseer, and Cora is small, we do not split the ground-truth communities to generate the tasks on these datasets. Note that, for fairness, we try to use the same number of labels to compare. ICS-GNN uses 3 positive and 3 negative samples for each query task. QD-GNN and QD-GNN-p use 3 positive samples and 3 negative samples in the training dataset. Our methods only use 3 positive samples.

Evaluation Metrics. To evaluate the quality of the found results, we employ three metrics: F1-score [46], Normalized Mutual Information (NMI) [47], Jaccard similarity [44]. We report the average score for 500 query tasks.

B. Evaluation of Community Search

1) Effectiveness Evaluation: We show the effectiveness results in Table III. Compared with the non-ML-based models (e.g., K-Core, K-Truss, K-ECC, and CTC), COCLEP consistently outperforms them in terms of all the three metrics in all datasets. This is because the ML-based method can be more flexible in capturing various types of community structures. Compared with ML-based models (e.g., ICS-GNN and QD-GNN), COCLEP, QD-GNN-P and ICS-GNN are more scalable. QD-GNN and COCLE need to train the model from the

TABLE III: Effectiveness on COCLE and COCLEP compared with other methods (in percentage). In training datasets, for each query node, ICS-GNN uses 3 positive samples and 3 negative samples; QD-GNN and QD-GNN-P uses 3 positive samples and 3 negative samples; COCLE and COCLEP use 3 positive samples only.

Metric	Dataset	K-Core	K-Truss	K-ECC	CTC	ICS-GNN	QD-GNN	QD-GNN-P	COCLE	COCLEP
F1	Football	14.88	82.18	33.42	78.79	26.29±0.46	83.86±0.36	82.53±1.04	**85.64±0.46**	84.27±2.15
	Citeseer	15.66	0.95	7.11	0.88	3.94±0.03	19.04±0.00	24.10±0.00	28.27±0.10	**31.86±0.0**
	Cora	20.92	4.32	7.69	1.77	8.86±0.13	23.11±0.69	38.52±0.20	23.00±1.12	**39.51±0.0**
	Facebook	20.31	30.82	32.33	36.15	23.60±0.28	38.48±1.16	37.49±1.17	**40.55±0.32**	39.82±0.80
	Amazon	38.56	84.43	75.48	74.38	25.12±0.40	o.o.m	80.50±1.28	o.o.m	**89.15±0.48**
	DBLP	5.35	63.13	62.49	66.70	25.05±0.48	o.o.m	56.67±1.44	o.o.m	**69.43±0.20**
	Youtube	0.08	12.01	15.92	47.57	15.20±0.07	o.o.m	22.98±2.04	o.o.m	**48.82±0.19**
	LiveJournal	6.39	70.03	7.17	72.15	24.15±0.33	o.o.m	56.78±1.79	o.o.m	**73.24±0.24**
NMI	Football	2.57	75.56	20.25	69.08	6.61±0.43	76.47±0.70	75.31±1.96	**79.11±0.85**	77.35±3.06
	Citeseer	1.63	0.52	0.93	0.50	1.64±0.02	1.14±0.00	**5.39±0.00**	0.24±0.29	5.15±0.0
	Cora	0.98	1.96	2.03	1.13	3.62±0.08	10.55±0.38	**13.97±0.14**	8.33±4.18	13.79±0.0
	Facebook	14.53	23.73	24.08	27.63	14.39±0.23	31.79±0.26	31.58±0.74	**32.37±0.23**	31.80±0.63
	Amazon	37.21	82.17	72.47	71.68	20.13±0.35	o.o.m	76.59±1.46	o.o.m	**87.11±0.55**
	DBLP	5.12	59.58	59.17	63.35	19.76±0.48	o.o.m	52.11±1.41	o.o.m	**64.78±0.26**
	Youtube	0.04	10.92	13.87	43.51	12.48±0.07	o.o.m	20.50±1.88	o.o.m	**46.02±0.19**
	LiveJournal	5.90	66.26	6.83	68.04	19.54±0.32	o.o.m	53.31±1.74	o.o.m	**69.31±0.24**
Jaccard	Football	8.08	77.77	26.06	71.58	15.56±0.34	78.92±0.44	77.85±1.43	**81.31±0.85**	79.23±2.99
	Citeseer	9.12	0.51	4.08	0.44	2.05±0.01	11.26±0.00	15.85±0.00	16.61±0.07	**20.03±0.0**
	Cora	11.99	2.58	4.27	0.90	4.72±0.07	14.72±0.54	25.48±0.20	14.25±0.69	**25.86±0.0**
	Facebook	14.07	22.78	23.00	25.87	14.20±0.19	28.71±0.58	28.05±0.87	**30.65±0.26**	29.86±0.74
	Amazon	35.39	78.61	66.80	66.35	15.18±0.32	o.o.m	70.53±1.83	o.o.m	**84.24±0.69**
	DBLP	4.85	54.82	54.40	58.04	14.68±0.34	o.o.m	45.49±1.56	o.o.m	**59.04±0.35**
	Youtube	0.04	9.36	10.96	37.74	8.39±0.05	o.o.m	15.31±1.54	o.o.m	**38.48±0.20**
	LiveJournal	5.27	60.92	6.36	61.91	14.44±0.24	o.o.m	46.28±1.84	o.o.m	**62.75±0.33**

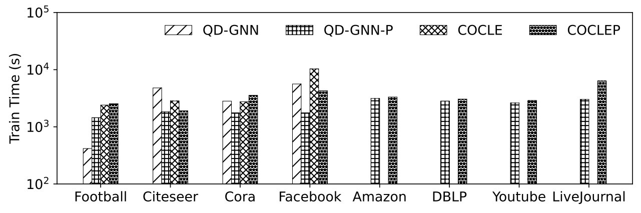


Fig. 4: Training time.

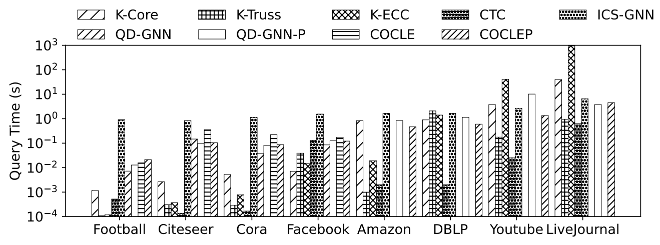


Fig. 5: Query time for all methods.

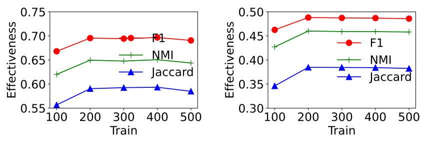
whole graph, leading to out-of-memory errors on large graphs. Thus, the results of QD-GNN and COCLE for larger datasets are not reported and marked as out-of-memory (o.o.m.). On the small datasets, COCLE and COCLEP achieve better performance compared with QD-GNN and QD-GNN-P on Football and Facebook and comparable performance on Cora and Citeseer. On the large datasets, COCLEP consistently outperforms QD-GNN-P. Thus, QD-GNN and QD-GNN-P may not be suitable for semi-supervised learning with few labels known for each query node. The effectiveness of ICS-GNN is relatively lower compared with the other ML-based models, because it is proposed for a different kind of learning process, i.e., it needs to be interactively provided labels and it is proposed for the attributed graphs. Besides, it needs to set the community size to find the community. When the community size is not properly set, the results can be influenced. Comparing COCLE and COCLEP, we surprisingly find that COCLEP can perform even better than COCLE sometimes. One reason is that the graph partitioning also supervises how to form the community in some sense. All in all, we conclude that COCLEP is the most effective method in general. See the technical report [41] for more experimental results.

2) Training Efficiency: Figure 4 shows the training time for the ML-based methods. We do not report the training time for ICS-GNN because it is an interactive method that requires retraining the model whenever there is a new query. In other words, the training process of ICS-GNN is performed repetitively instead of only once. On all datasets except for the Football, Citeseer, Cora, Facebook, QD-GNN and COCLE run out of memory on GPU. We can see that COCLE and COCLEP always achieve the comparable efficiency compared with QD-GNN and QD-GNN-P except for Football. COCLEP and QD-GNN-P train the model faster because their cost is related to local subgraphs. Since the Football is very small, the COCLEP and QD-GNN-P do not show the superiority of partition.

3) Query Efficiency: Figure 5 reports the query time for all methods. Since ICS-GNN needs to train a model for each query node, we sum up its training time and prediction time as the query time. Thus, its query time is not better than that of COCLEP, COCLE, QD-GNN and QD-GNN-P. CTC has the fastest query processing on most datasets because it uses local exploration to find the community and the ground-truth communities usually have small diameters. The query time of COCLEP is comparable to or better than other methods on most datasets.

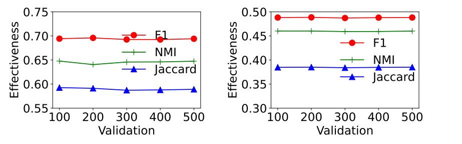
C. Parameter Analysis

1) Varying training set size, validation size, number of the epoch: Figures 6, 7, 8 show the effectiveness with varying the



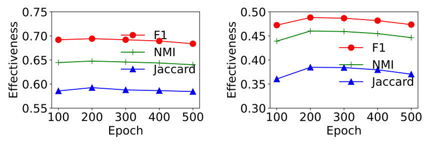
(a) DBLP (b) Youtube

Fig. 6: Effectiveness with varying train size.



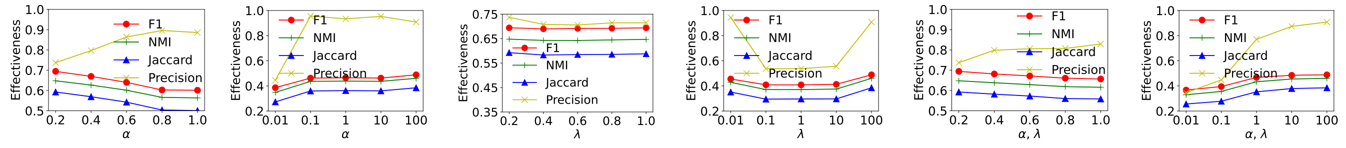
(a) DBLP (b) Youtube

Fig. 7: Effectiveness with varying validation size.



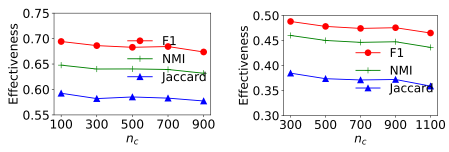
(a) DBLP (b) Youtube

Fig. 8: Effectiveness with varying number of epochs.



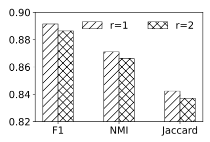
(a) DBLP (b) Youtube (c) DBLP (d) Youtube (e) DBLP (f) Youtube

Fig. 9: Effectiveness with varying α and λ.



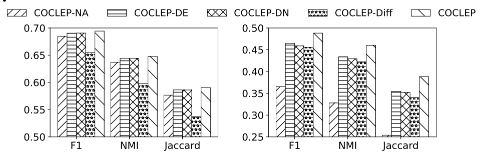
(a) DBLP (b) Youtube

Fig. 10: Effectiveness with varying n_c.



(a) Amazon (b) DBLP

Fig. 11: Effectiveness with varying r.



(a) DBLP (b) Youtube

Fig. 12: Effectiveness with different augmentation methods.

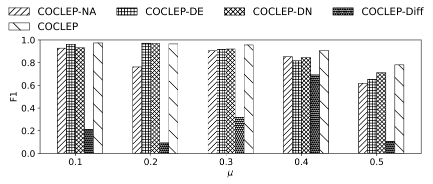
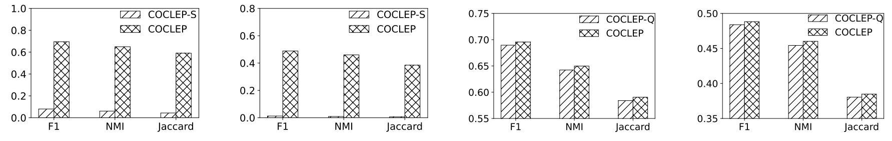


Fig. 13: F1-score of different augmentation methods on synthetic datasets.



(a) DBLP (b) Youtube (c) DBLP (d) Youtube

Fig. 14: Effectiveness with feature fuse model.

training dataset size, validation dataset size, and number of epochs. We observed that when training size is increased, the F1-score, NMI, and Jaccard are also increased. This is because if the training size is increased, there will be more information about the ground-truth communities are learned, which leads to an effective model. But when increasing the validation size, these scores have not changed much. Thus, only a few ground-truth communities can help to obtain an effective threshold p. If the number of epochs is increased, these scores are slightly decreased due to over-fitting on Youtube.

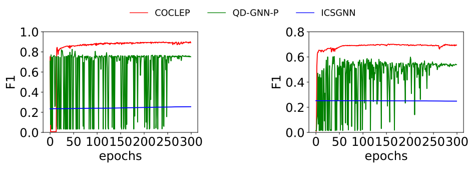
2) Varying α and λ: Figure 9 (a)-(b) report the effectiveness with varying α. We observed that the precision is increased on DBLP and Youtube when α increases. On youtube, the F1-score, NMI, and Jaccard score are increased when α increases. On DBLP, however, the F1-score, NMI, and Jaccard score decrease when we increase α. This is because some nodes in the ground-truth communities are removed when α increases. As shown in Figure 9 (c)-(d), the F1-score, NMI, and Jaccard score are not sensitive to different settings of λ. But on Youtube, the precision has a large fluctuation. This is due to the α is set 100, when we increase the λ, there is a large gap between α and λ. However, when we increase α and λ simultaneously on 9 (e)-(f), the precision is increased, the F1-score, NMI, and Jaccard score have little fluctuation on DBLP, while the F1-score, NMI, and Jaccard score are increased on Youtube. Thus, we can increase α and λ simultaneously to find suitable parameters.

3) Varying n_c: Figure 10 shows the effectiveness with various n_c values on DBLP and Youtube. We observed that when n_c increases, the F1-score, NMI, and Jaccard score have a slightly decreasing trend. This is because the graph structure is changed if n_c is set large and each partition becomes small. To conclude, COCLEP is not sensitive to the setting of n_c for a reasonable range of settings.

4) Varying r: Figure 11 shows the F1-score, NMI, Jaccard score of Amazon, DBLP with varying parameter r in constructing the hypergraph augmentation view. Since on the Youtube, COCLEP runs out of the memory when r=2, we do not show the results on the Youtube. When we increase r, the metric values of COCLEP are decreased, concluding that $r = 1$ can already gives a satisfactory performance. This is because the ground-truth community usually has a small diameter. The 1-hop subgraph of one node already contains enough nodes in the ground-truth community that this node belongs to. Thus, for practical efficiency concerns, we always set $r = 1$.

D. Ablation analysis

1) Augmentation Methods Evaluation: Figure 12 shows the values of variants of COCLEP. COCLEP-NA has no augmentation method, and it only uses the original view to train the model. COCLEP-DE and COCLEP-DN use random edge dropping and node dropping with a ratio of 0.1 to



(a) Amazon (b) DBLP

Fig. 15: Convergence analysis.

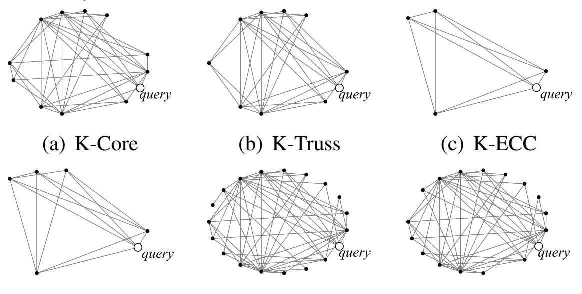
generate the augmentation view, respectively. COCLEP-Diff [48] generates a view by transforming an adjacency matrix to a diffusion matrix. Since the diffusion matrix is too dense, it easily leads to out of the GPU memory. We select the nodes as the neighbors with the top-10 value. We observe that COCLEP achieves competitive performance compared with other methods. Compared with COCLEP-ND, COCLEP achieves a significantly higher F1 score on Youtube. This demonstrates that our proposed hypergraph augmentation method can be more beneficial in finding the community.

We further use synthetic datasets to analyze the effectiveness of different augmentation methods in Figure 13. The synthetic datasets are generated by LFR benchmark [49]. The default parameters of the synthetic network are V=5000, d_{avg}=20, d_{max}=100, C_{min}=20, and C_{max}=100. We change the parameter μ, which is the fraction of edges that are between different communities, to analyze the effectiveness of different augmentation methods. When μ is large, the average clustering coefficient of the network becomes small, and discovering the community is difficult. When μ=0.3, we set α and λ to 0.1. When μ=0.4, we set α and λ to 0.01. For other values of μ, we set α and λ to 1. We use the threshold determined by the optimal F1 score on the validation dataset. In Figure 13, we can see that if μ is small, the F1 score of COCLEP-DE, COCLEP-DN, COCLEP are close. When μ increases, COCLEP achieves better performance. Thus, when the average clustering coefficient of the network is smaller, COCLEP is in general better than other methods. This may also indicate the rationale behind the performance on DBLP and Youtube. On DBLP, the average clustering coefficient is 0.632, while for Youtube it is 0.081. So, compared with other augmentation methods, COCLEP achieves significantly higher effectiveness on the Youtube dataset whereas the results for DBLP are close.

2) Fearture Fuse Model Analysis: Figure 14 (a)-(b) show the values of COCLEP-S and COCLEP. COCLEP-S only uses the structure encoder to train the model. With only the structure encoder used, the model cannot capture the query node information, leading to low effectiveness. Figure 14 (c)-(d) shows the values of COCLEP-Q and COCLEP. COCLEP-Q only uses query node encoder to train the model. COCLEP achieves the better performance compared with the COCLEP-Q. This demonstrates the effectiveness of the proposed fuse model.

3) Convergence Analysis: Figure 15 shows the convergence trend of COCLEP, QD-GNN-P, and ICS-GNN. Since on Youtube, QD-GNN-P can not be completed in a reasonable time, we show the results on Amazon and DBLP. We observed that COCLEP and ICS-GNN converge quickly (around using 50 epochs) while QD-GNN-P tends to have fluctuations. COCLEP has a better F1-score compared with ICS-GNN and QD-GNN across a wide range of epochs.

E. Case study



(a) K-Core (b) K-Truss (c) K-ECC

(d) CTC (e) COCLEP (f) Ground-truth

Fig. 16: Case Study.

We conducted a case study to show the usefulness of COCLEP on Amazon dataset. In Figure 16, the query node is marked in white color, and the nodes in black colors are those in the communities returned by different methods or in the ground-truth community. We note that our algorithm returns a community that is the closest to the ground-truth community. In contrast, other methods only find part of the ground-truth community, for the reason that they use a predefined structure to search the community, making them sometimes inflexible to approximate the ground-truth community.

VIII. Conclusion and Future Work

In this paper, we proposed COCLEP, a contrastive learning-based method that improves the performance of community search. We unleash the potential of using contrastive learning in community search many-fold. These include redesigns of augmentation views, positive/negative pairs, contrastive loss, encoding of the augmentation view, as well as its combination with graph partitioning. To the best of our knowledge, COCLEP is the first contrastive learning based model for community search that significantly reduces the number of required labeled data. Experiments on real-world graphs show the state-of-the-art performance of our models.

There are several promising future directions. The first is to consider handling unlabeled data. There is potential to extend our methods to handle these datasets. For example, one can use some existing unsupervised metrics to shortlist some positive pairs, and then apply our method for a semi-supervised learning. Another direction is to consider sampling techniques to circumvent the efficiency issue. There is always a tradeoff between sampling rate and result quality. Practically, one may select a proper sampling rate based on the practical need on the result quality.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (62032013, 61902004), Singapore MOE AcRF Tier 2 funding (MOE-T2EP20122-0003), Tier-1 seed funding (RS05/21) and NTU SUG-NAP (022029-00001), ARC FT200100787, DP210101347, Singapore Institute of Technology Ignition Grant (R-IE2-A405-0001), Chinese Scholarship Council, the Project of Beijing Municipal Education Commission (KM202010009009).

References

[1] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, no. 1, pp. 353–392, 2020.

[2] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *ICDE*, 2019, pp. 88–99.

[3] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *SIGKDD*, 2010, pp. 939–948.

[4] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endow.*, vol. 9, no. 4, pp. 276–287, 2015.

[5] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang, "Querying minimal steiner maximum-connected subgraphs in large graphs," in *CIKM*, 2016, pp. 1241–1250.

[6] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1298–1309, 2017.

[7] K. Yao and L. Chang, "Efficient size-bounded community search over large networks," *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1441–1453, 2021.

[8] J. Kim, S. Luo, G. Cong, and W. Yu, "DMCS : Density modularity based community search," in *SIGMOD*, 2022, pp. 889–903.

[9] B. Liu, F. Zhang, W. Zhang, X. Lin, and Y. Zhang, "Efficient community search with size constraint," in *ICDE*, 2021, pp. 97–108.

[10] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang, "Index-based optimal algorithms for computing steiner components with maximum connectivity," in *SIGMOD*, 2015, pp. 459–474.

[11] J. Gao, J. Chen, Z. Li, and J. Zhang, "ICS-GNN: lightweight interactive community search via graph neural network," *Proc. VLDB Endow.*, vol. 14, no. 6, pp. 1006–1018, 2021.

[12] Y. Jiang, Y. Rong, H. Cheng, X. Huang, K. Zhao, and J. Huang, "Query driven-graph neural networks for community search: From non-attributed, attributed, to interactive attributed," *Proc. VLDB Endow.*, vol. 15, no. 6, pp. 1243–1255, 2022.

[13] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *SIGKDD*, 2020, pp. 1150–1160.

[14] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5812–5823, 2020.

[15] H. Xiong, J. Yan, and L. Pan, "Contrastive multi-view multiplex network embedding with applications to robust network alignment," in *SIGKDDg*, 2021, pp. 1913–1923.

[16] J. Leskovec, "Stanford large network dataset collection," http://snap.stanford.edu/data.

[17] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endow.*, vol. 8, no. 5, pp. 509–520, 2015.

[18] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 709–720, 2017.

[19] A. Al-Baghdadi and X. Lian, "Topic-based community search over spatial-social networks," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2104–2117, 2020.

[20] J. Kim, T. Guo, K. Feng, G. Cong, A. Khan, and F. M. Choudhury, "Densely connected user community and location cluster search in location-based social networks," in *SIGMOD*, 2020, pp. 2199–2209.

[21] J. Luo, X. Cao, X. Xie, Q. Qu, Z. Xu, and C. S. Jensen, "Efficient attribute-constrained co-located community search," in *ICDE*, 2020, pp. 1201–1212.

[22] F. Guo, Y. Yuan, G. Wang, X. Zhao, and H. Sun, "Multi-attributed community search in road-social networks," in *ICDE*, 2021, pp. 109–120.

[23] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1233–1244, 2016.

[24] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endow.*, vol. 10, no. 9, pp. 949–960, 2017.

[25] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Vac: vertex-centric attributed community search," in *ICDE*, 2020, pp. 937–948.

[26] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 854–867, 2020.

[27] X. Jian, Y. Wang, and L. Chen, "Effective and efficient relational community detection and search in large dynamic heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1723–1736, 2020.

[28] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang, "Efficient and effective community search on large-scale bipartite graphs," in *ICDE*, 2021, pp. 85–96.

[29] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, vol. 119, 2020, pp. 1597–1607.

[30] A. Van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv e-prints*, pp. arXiv–1807, 2018.

[31] Y. Meng, J. Huang, G. Wang, C. Zhang, H. Zhuang, L. Kaplan, and J. Han, "Spherical text embedding," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 297–304.

[33] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *Proceedings of the Web Conference 2021*, 2021, pp. 2069–2080.

[34] B. Li, B. Jing, and H. Tong, "Graph communal contrastive learning," *arXiv preprint arXiv:2110.14863*, 2021.

[35] S. Wan, S. Pan, J. Yang, and C. Gong, "Contrastive and generative graph convolutional networks for graph-based semi-supervised learning," *arXiv preprint arXiv:2009.07111*, 2020.

[36] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 661–18 673, 2020.

[37] T. Wang and P. Isola, "Understanding contrastive representation learning through alignment and uniformity on the hypersphere," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9929–9939.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[39] S. Bai, F. Zhang, and P. H. S. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognit.*, vol. 110, p. 107637, 2021.

[40] F. Wang and H. Liu, "Understanding the behaviour of contrastive loss," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 2495–2504.

[41] https://sites.google.com/view/cocleptechnicalreport/.

[42] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[43] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on machine learning*, 2016, pp. 40–48.

[44] Y. Zhang, Y. Xiong, Y. Ye, T. Liu, W. Wang, Y. Zhu, and P. S. Yu, "Seal: Learning heuristics for community detection with generative adversarial networks," in *SIGKDD*, 2020, pp. 1103–1113.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015.

[46] S. Fang, K. Zhao, G. Li, and J. X. Yu, "Community search: Learn from small data," *arXiv preprint arXiv:2201.00288*, 2022.

[47] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of statistical mechanics: Theory and experiment*, vol. 2005, no. 09, p. P09008, 2005.

[48] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning*, 2020, pp. 4116–4126.

[49] S. Muff, F. Rao, and A. Caflisch, "Local modularity measure for network clusterizations," *Physical Review E*, vol. 72, no. 5, p. 056107, 2005.