QD-GCN: Query-Driven Graph Convolutional Networks for Attributed Community Search

Yuli Jiang[*1], Yu Rong[†2], Hong Cheng[*3], Xin Huang[‡4], Kangfei Zhao[*5], Junzhou Huang[†6]

[*]*The Chinese University of Hong Kong, Hong Kong, China*
[†]*Tencent AI Lab, Shen Zhen, China*
[‡]*Hong Kong Baptist University, Hong Kong, China*
$\{^1$yljiang,^{3}hcheng,^{5}kfzhao$\}$@se.cuhk.edu.hk,^{2}yu.rong@hotmail.com,^{4}xinhuang@comp.hkbu.edu.hk,^{6}jzhuang@uta.edu

Abstract—Recently, attributed community search, a related but different problem to community detection and graph clustering, has been widely studied in the literature. Compared with the community detection that finds all existing static communities from a graph, the attributed community search (ACS) is more challenging since it aims to find dynamic communities with both cohesive structures and homogeneous node attributes given arbitrary queries. To solve the ACS problem, the most popular paradigm is to simplify the problem as two sub-problems, that is, structural matching and attribute filtering and deal with them separately. However, in real-world graphs, the community structure and the node attributes are actually correlated to each other. In this vein, current studies cannot capture these correlations which are vital for the ACS problem.

In this paper, we propose Query-Driven Graph Convolutional Networks (QD-GCN), an end-to-end framework that unifies the community structure as well as node attribute to solve the ACS problem. In particular, QD-GCN leverages the Graph Convolutional Networks, which is a powerful tool to encode the graph topology and node attributes concurrently, as the backbones to extract the query-dependent community information from the original graph. By utilizing this query-dependent community information, QD-GCN is able to predict the target community given any queries. Experiments on real-world graphs with ground-truth communities demonstrate that QD-GCN outperforms existing attributed community search algorithms in terms of both efficiency and effectiveness.

Index Terms—community search, attributed graph, graph neural network

I. INTRODUCTION

Graph is an essential data structure to represent entities and their relationships, e.g., social networks, protein-protein interaction networks, web graphs, and knowledge graphs, to name a few. Community, a subgraph of densely inner-connected and loosely inter-connected structure, naturally exists as a functional module in real-world graphs. Attributed community search, which aims at finding query-dependent communities such that the community members are densely connected and have homogeneous attribute values, has attracted a lot of attention recently [1]–[5]. Figure 1 shows an example of attributed community search in a collaboration network. Different from community detection [6]–[10], which finds all communities in a network, community search finds query-dependent communities. Thus, attributed community search is a more personalized and complicated search problem.

There exist several models [4], [5] for attributed community search in the literature. These methods take a two-stage process. First, it finds a candidate community for query vertices with dense structure; Second, it shrinks the community by optimizing an attribute function. However, they *treat the cohesive structure and attribute homogeneity in a community separately*, which ignores the correlation between structure and attributes. For instance, in protein-protein interaction networks, proteins with similar functions are more likely to interact with each other [11]. This may lead to inferior community search performance.

Current studies also suffer from two serious limitations of *structure inflexibility* and *attribute irrelevance*. First, most community search models are based on a pre-defined subgraph pattern, such as k-core [4], [12], [13], k-truss [5], [14], [15], k-clique [16], [17], and k-edge connected component (ECC) [18], [19]. The pre-defined subgraph pattern imposes a very rigid requirement on the topological structure of communities, which may not perfectly hold in real-world communities. Second, all attributes are treated independently [4], [5]. These models ignore similarities and correlations between attributes. For example in Figure 1, two terms "GNN" and "GCN" are not identical but very similar, thus we should not treat them as two completely different attributes.

Inspired by the great success of deep learning and graph convolutional network (GCN) [20] on combining attribute and structure in many graph learning problems, we utilize GCN to solve this attributed community search problem. However, existing GCN models cannot support the query operation in this problem, that is, given different query inputs, the model will output different query-specific results. To address the limitations of the existing two-stage approaches and GCNs, we propose an end-to-end supervised model, named Query-Diven Graph Convolutional Network (QD-GCN). Due to its data driven characteristics, our method can model different topological structures for different graphs. In addition, we consider the correlation between attributes and the overall graph information, and extend GCNs to support the query operation.

We design four learning components in QD-GCN for different learning tasks, namely, graph encoder, structure encoder, attribute encoder and feature fusion. First, the graph encoder
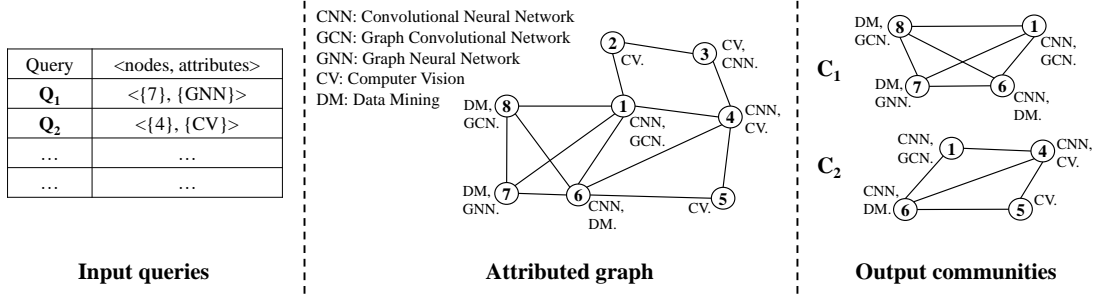
Fig. 1. Attributed community search in a collaboration network, where nodes represent authors and attributes represent authors' research topics. Subgraph C_1 is the answer community of query Q_1 in the input queries. Subgraph C_2 is the answer community of query Q_2.

learns query-independent features from both structure and attribute of the entire graph and make QD-GCN robust to cope with a limited number of training queries. Second, the structure encoder learns query-specific local structural features. Third, the attribute encoder provides related attribute features of queries and considers the similarity between attributes by a structure-attribute bipartite graph. The structure encoder and attribute encoder extend the previous GCN models to support query operations and provide interfaces of query inputs for the QD-GCN model. Last, the feature fusion component fuses the outputs of the above three encoders and obtains the final query-specific output of the QD-GCN model. It combines all features and transmits the fusion result to the structure and attribute encoders, through which QD-GCN can process query-specific structure and attribute information simultaneously. The QD-GCN model is depicted in Figure 2.

To summarize, we make the following contributions.

- We study the problem of attributed community search, that is, finding query-dependent communities with cohesive structure and homogeneous attributes w.r.t. the query vertices and query attributes.
- To the best of our knowledge, this is the first work that proposes a graph convolutional network (GCN) learning model for community search called QD-GCN. Our novel learning framework extends GCN into a query-support model and learns the community information from both the global graph and local features of queries.
- We design four learning components, including a graph encoder, a structure encoder, an attribute encoder, and a feature fusion component. Equipped with these components, a unified framework of QD-GCN is trained to integrate both structure and attribute into communities, which avoids the limitations of the existing two-stage approaches.
- We conduct extensive experiments on real-world data sets with ground-truth communities to evaluate our proposed model. The results demonstrate that our model significantly outperforms the state-of-the-art methods in terms of community quality and query time.

Roadmap. The rest of the paper is organized as follows. Section II discusses related work. Section III gives some preliminaries. Section IV introduces the QD-GCN model for

attributed community search. We present the experimental results in Section V and conclude the paper in Section VI.

II. RELATED WORK

Our study is closely related to the following three research topics:

Community Search. The problem of community search [12] is to find densely connected communities containing the query nodes. A comprehensive survey of community search models and existing approaches can be found in [1], [2]. Various community models have been proposed based on different cohesive metrics, including k-core [12], [13], k-truss [14], [15], [21], k-clique [16], [17], and k-edge connected component (ECC) [18], [19]. These pre-defined cohesive metrics are inflexible and always too loose or tight for the topology structure of communities. k-core based models require the degree of each nodes in community larger than or equal to k, which makes the community not cohesive enough. In order to find cohesive communities, k-clique based models require nodes are connected to all other nodes in a community, which is too restrict. There are also some relaxed versions of clique model [16]. k-truss based models are between above two, which require that the triangle number of each edges in the communities is at least $(k-2)$. However, they still cannot fit into all situations for various networks and nodes.

Apart from community search on simple graphs with only structural information, recently, [4] and [5] have studied attributed community search, which aims to discover communities containing both query nodes and query attributes simultaneously. ACQ [4] is based on the k-core model and finds communities with the maximum number of common query attributes shared by community members. ATC [5] finds k-truss communities with the maximum pre-defined attribute score. Both ACQ and ATC adopt two-stage process. They first impose an inflexible structural constraint to find candidate communities. Then functions of attribute score are optimized to select the most related communities. However, the attribute score functions ignore the similarities between attributes, and these two-stage methods fail to capture the correlation between structure and attribute. In this paper, we propose the QD-GCN model, which considers the cohesive structure and homogeneous attributes in an integrated way.
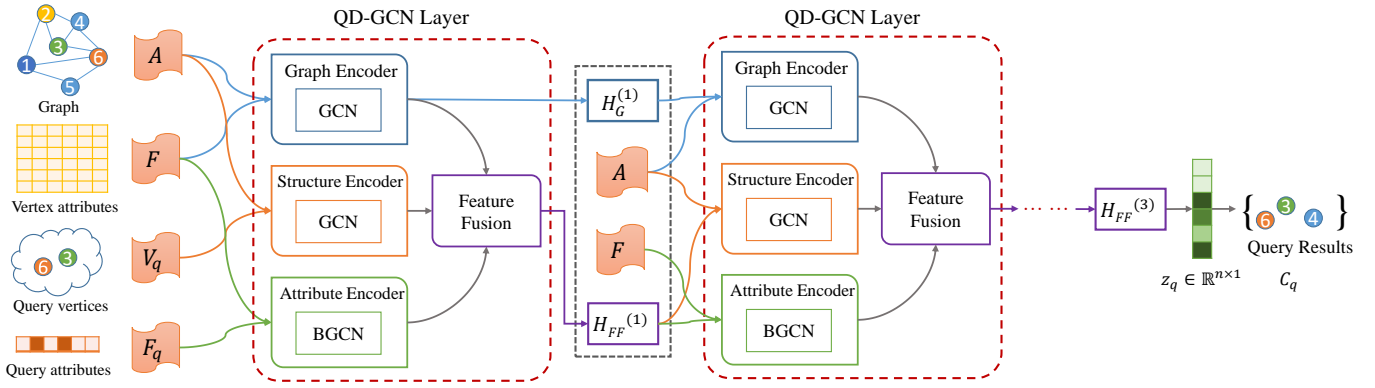
Fig. 2. An illustration of three-layer QD-GCN framework. Given the graph adjacent matrix A, the attribute matrix F, the query $< \mathcal{V}_q, \mathcal{F}_q >$, QD-GCN employs three GCN-based encoders and one fusion components to model different aspects of the attributed community search task.

Community Detection. Community detection aims to find all densely connected subgraphs in the entire network [6]. It is a related but different problem to community search. Community detection concerns more about global structure and has no query input. While community search is a query dependent problem and pay more attention to the local information around the query. A large number of approaches have been proposed for community search in attributed graphs [7], [22]. Recently, with the development of convolutional networks, researchers have also applied deep learning models for community detection [8]. Many of these studies focus on attributed graphs [9], [10], [23]. These models for community detection take an attributed graph as input and all detected communities as output, which don't support query operations in community search problem.

Graph Neural Network. Benefiting from the huge success of convolutional neural networks in natural language processing and computer vision, many graph analytic tasks have been studied via graph convolutional neural networks (GCNs) [20], such as node classification and graph classification [24]–[26], recommendation [27], [28], and adversarial attacks [29], [30]. To build good models, the advanced techniques of pooling [31]–[33] and attention [28], [31], [34] have been developed. However, most learning models are designed for particular tasks based on the techniques of graph embedding [35], [36] or end-to-end solutions [37], [38], which has fixed inputs and outputs for one graph. As for community search problem, there are various queries as inputs for only one graph and it needs corresponding different communities as outputs. There exist no GCN-based solutions that support these query operations in attributed community search problem. To the best of our knowledge, we are the first to propose the GCN-based model for this query-dependent community search problem on attributed graphs.

III. PRELIMINARIES

In the following, we present our problem definition and an existing classical model of graph convolutional network.

A. Problem Definition

Let $G(\mathcal{V}, \mathcal{E}, F)$ be an attributed graph with a set $\mathcal{V}$ of vertices, a set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of edges, and a matrix F of vertex features. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of vertices and edges respectively. The adjacency matrix of G is denoted as $A \in \{0, 1\}^{n \times n}$. For any pair of vertices $v_i, v_j \in V$, if there exists an edge $(v_i, v_j) \in \mathcal{E}$, $A_{ij} = 1$; otherwise, $A_{ij} = 0$. Let $\mathcal{F}$ be the set of vertex features in G, and $d = |\mathcal{F}|$ be the number of distinct vertex features. The vertex feature matrix $F \in \mathbb{R}^{n \times d}$ is defined as follows. For keyword feature f_k, if vertex v_i has this keyword, then $F_{ik} = 1$; otherwise, $F_{ik} = 0$. For numerical feature f_j, F_{ij} is the value of vertex v_i on this feature. The notations we use in this paper are summarized in Table I.

Problem Statement. For an attributed graph G with adjacency matrix A and vertex feature matrix F, given a query $\langle \mathcal{V}_q, \mathcal{F}_q \rangle$ where $\mathcal{V}_q \subseteq \mathcal{V}$ is a set of query vertices, and $\mathcal{F}_q \subseteq \mathcal{F}$ is a set of query attributes, the problem of *attributed community search* (ACS) is to find the query-dependent community $\mathcal{C}_q \subseteq \mathcal{V}$. Nodes in community $\mathcal{C}_q$ need to be both structure cohesive and attribute homogeneous.

According to the problem definition, we can formulate the attributed community search as a multiple binary classification problem: given a query $q = \langle \mathcal{V}_q, \mathcal{F}_q \rangle$, we need to identify all vertices that belong to the output community. We propose a supervised learning model QD-GCN to solve this attributed community search problem.

Given a set of training queries $\mathcal{Q}_{\text{train}} = \{q_1, q_2, ...\}$ and corresponding ground-truth communities $\mathcal{Y}_{\text{train}} = \{y_1, y_2, ...\}$, we train a model to minimize a loss function to fit the training data. Then for other queries, we can apply the trained model easily to predict the community search result. The optimization loss function can be formulated as:

$$\min \mathcal{L} = \sum_{(\mathcal{V}_q, \mathcal{F}_q, y_q) \in \mathcal{Q}_{\text{train}}} \sum_{i=1}^{n} BCE(y_{qi}, z_{qi})$$

$$BCE(y_{qi}, z_{qi}) = \begin{cases} -\log(z_{qi}), & y_{qi} = 1 \\ -\log(1 - z_{qi}), & y_{qi} = 0 \end{cases} \quad (1)$$

In (1), $\mathcal{Q}_{\text{train}} = \{(\mathcal{V}_q, \mathcal{F}_q, y_q)\}_{q=1}^{k}$ is the training set of queries. The triplet $(\mathcal{V}_q, \mathcal{F}_q, y_q)$ represents a query $\langle \mathcal{V}_q, \mathcal{F}_q \rangle$ and its corresponding ground-truth community denoted by the indicator vector $y_q \in \{0,1\}^{n \times 1}$. If v_i is in the ground truth community, $y_{qi} = 1$; otherwise, $y_{qi} = 0$. $z_q \in \mathbb{R}^{n \times 1}$ is the output prediction vector by QD-GCN for query q.

B. A Classical Graph Convolutional Network Model

In the following, we introduce one classical Graph Concolutional Network (GCN) [20] proposed by Kipf and Welling, which is critical useful for our proposed model. And the layer-wise propagation rule is defined as:

$$H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l)}). \tag{2}$$

where

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

is the normalized adjacent matrix of the graph, given $A \in \{0,1\}^{n*n}$ is the adjacent matrix, $I_n \in \mathbb{R}^{n*n}$ is the identity matrix and $D \in \mathbb{R}^{n*n}$ is the degree diagonal matrix $D_{ii} = \sum_j (A + I_n)_{ij}$. The matrix $W^{(l)} \in \mathbb{R}^{d^{(l)}*d^{(l+1)}}$ is the weights of layer $(l+1)$ and $\sigma(\cdot)$ is the activation function, e.g., element-wise rectified linear function. $H^{(l)} \in \mathbb{R}^{n*d^{(l)}}$ is the node features learned by GCN in l-th layer with $H^{(0)} = \hat{F}$, where $\hat{F}$ is the row normalization of node attribute matrix F.

The graph convolutional layer is based on the first-order approximation of spectral convolutions on graphs and can be regarded as a special Laplacian smoothing on the node feature $H^{(l)}$ for generating new features for the $(l+1)$-th layer, i.e., $H^{(l+1)}$. In addition, the learned features of one node are also considered to be the weighted sum of all neighbors' features in the last layer. In other words, the learned features are diffused on the graph through edges (or adjacent matrix). The layer-wise propagation function of node v can be rewritten as:

$$h_v^{(l+1)} = \sigma(\text{SUM}(\{h_u^{(l)} W^{(l)} : u \in \mathcal{N}(v) \cup \{v\}\})), \tag{3}$$

where $h_v^{(l+1)}$ is the learned new features of node v in $(l+1)$-th layer, $h_u^{(l)}$ is the input features of node u, and $\mathcal{N}(v)$ is the neighborhood set of node v. The function $\text{SUM}(\cdot)$ is the summation function. Activation function $\sigma(\cdot)$ and trainable weight matrix $W^{(l)}$ are same to those in Eq (2).

As a popular neural network framework for modeling graph data, GCN serves as a building block in many supervised and semi-supervised learning tasks. Our QD-GCN model for attributed community search problem is also based on this classical GCN.

IV. QD-GCN Model

In this section, we present the Query-Driven Graph Convolutional Network (QD-GCN) model for attributed community search. We first introduce the overall framework of QD-GCN. Then, we describe four components of QD-GCN one by one in detail.

TABLE I
LIST OF KEY SYMBOLS IN QD-GCN

Symbol	Meaning	Symbol	Meaning
$\mathcal{V}$	Set of nodes	$n \in \mathbb{R}$	Number of nodes
$\mathcal{E}$	Set of edges	$m \in \mathbb{R}$	Number of edges
$\mathcal{F}$	Set of attribute	$d \in \mathbb{R}$	Number of attributes
$\mathcal{V}_q$	Set of query nodes	$\mathcal{F}_q$	Set of query attributes

Symbol	Meaning
$A \in \mathbb{R}^{n*n}$	Adjacent matrix
$F \in \mathbb{R}^{n*d}$	Nodes attribute matrix
$y_q \in \{0,1\}^n$	Ground-truth community vector of query $q = \langle \mathcal{V}_q, \mathcal{F}_q \rangle$
$z_q \in \mathbb{R}^n$	Output vector of query q from QD-GCN
$\hat{z}_q \in \{0,1\}^n$	Predicted community vector of z_q through a threshold γ
$d^{(l)} \in \mathbb{R}$	Chanel size (feature map size) of l-th layer
$H_{FF}^{(l+1)} \in \mathbb{R}^{n*d^{(l+1)}}$	Output of Feature Fusion and QD-GCN model on $(l+1)$-th layer
$H_G^{(l)} \in \mathbb{R}^{n*d^{(l)}}$	Hidden features of graph encoder on (l)-th layer
$I_S^{(l)} \in \mathbb{R}^{n*d^{(l)}}$, $H_S^{(l+1)} \in \mathbb{R}^{n*d^{(l+1)}}$	Input/output features of structure encoder on $(l+1)$-th layer
$I_V^{(l)} \in \mathbb{R}^{n*d^{(l)}}$, $H_V^{(l+1)} \in \mathbb{R}^{n*d^{(l+1)}}$	Input/output of the node side GCN in attribute encoder on $(l+1)$-th layer
$I_F^{(l)} \in \mathbb{R}^{d*d^{(l)}}$, $H_F^{(l+1)} \in \mathbb{R}^{d*d^{(l+1)}}$	Input/output of the attribute side GCN in attribute encoder on $(l+1)$-th layer
$E_G^{(l)}, E_S^{(l)}, E_A^{(l)} \in \mathbb{R}^{n*d^{(l)}}$	Embedding of each encoder on l-th layer
$W^{(l)} \in \mathbb{R}^{d^{(l)}*d^{(l+1)}}$	Trainable weighted matrix on $(l+1)$-th layer
$W_{\text{self}}^{(l)} \in \mathbb{R}^{d^{(l)}*d^{(l+1)}}$	Trainable self weighted matrix on $(l+1)$-th layer

A. Framework Overview

Figure 2 illustrates the whole framework of QD-GCN, which is a three-level graph convolutional network learning model. QD-GCN takes an attributed graph $G = (\mathcal{V}, A, F)$ and queries $q = \langle \mathcal{V}_q, \mathcal{F}_q \rangle$ as inputs and produces community $\mathcal{C}_q$ as outputs. In the QD-GCN model, there are four tasks for attributed community search.

Task 1: Supporting query operations. For this online search problem, it is essential to support query operations that are unsupported by other GCN models. Obviously, QD-GCN needs to provide query interfaces for both query nodes $\mathcal{V}_q$ and query attributes $\mathcal{F}_q$. In addition, as a supervised learning-based model, we also need to make the model robust to cope with various query inputs and avoid over-fitting.

Task 2: Capturing local structure features. Different from existing GCN models which treat all graph vertices equally, the community search problem pays more attention to query vertices and their local structure. In order to find densely connected communities for query inputs, QD-GCN needs to have the capacity of learning local structure features surrounding the query vertices.

Task 3: Capturing related attribute features. Similar to the second task, QD-GCN also needs to capture the attribute information related to query attributes. In addition, such attribute information contains not only features for the same attributes but also for similar attributes like "GNN" and "GCN" in Figure 1.

Task 4: Fusing structure and attribute features. For the attributed community search problem, it is important to fuse local structure and related attribute features in the search process, rather than handle these two parts separately.

According to these four tasks, we equip QD-GCN with four learning components: *graph encoder*, *structure encoder*, *attribute encoder*, and *feature fusion* as shown in Figure 2. In the following, we briefly introduce these four components.

Graph Encoder. Graph encoder offers an overview learning function based on both structure and attribute information of G. It takes the input of graph information, i.e., the adjacency matrix A and vertex attribute matrix F, which is independent of any specific queries. This is because there may exist limited training queries for learning the community information, but graph G itself contains rich information of topology structure and vertex attributes for community learning. This component encodes the query-independent graph information with both structure and attribute features. This query-independent information makes our QD-GCN model more robust and produces the graph embedding E_G.

Structure Encoder. Structure encoder provides the interface for query vertices and learns the local topology features for queries. It takes the input of graph structure and query vertices, i.e., adjacency matrix A and query vertices $\mathcal{V}_q$. Structure encoder focuses on modeling the graph local structure information of the query vertices and produces the query-specific structural embedding E_S.

Attribute Encoder. Attribute encoder provides the interface for query attributes and learns attribute information related to queries. It takes the input of graph attributes and query attributes, i.e., vertex feature matrix F and query attributes $\mathcal{F}_q$. Besides attribute similarity, we also consider the node-attribute relationships through a bipartite graph to obtain the query-specific attribute embedding E_A.

Feature Fusion. The feature fusion component combines all the above embeddings and obtains the final query-specific output of the ACS problem. It takes outputs of the three encoders as inputs, i.e., graph embedding E_G, structure embedding E_S and attribute embedding E_A. This component mixes global graph features and local query features, fuses structure and attribute information, and balances them to get an accurate community.

In the following sections, we will introduce each component in more detail.

B. Graph Encoder

The graph encoder fuses graph structure information and graph attribute features including adjacency matrix A and node attribute matrix F, both of which are independent of queries. We apply the layer-wise forward propagation of GCN proposed by Kipf and Welling [20] to graph encoder, which has been introduced in Sec III-B. In order to fully make use of vertex attribute, the graph encoder is designed and equipped with a self feature modeling [34]. The forward layer of graph encoder is defined as:

$$H_G^{(l+1)} = \sigma(\hat{A}H_G^{(l)}W_G^{(l)} + H_G^{(l)}W_{G_{\text{self}}}^{(l)} + b), \qquad (4)$$

where the normalized adjacency matrix $\hat{A}$ is same as it in Eq (2). $H_G^{(l+1)}$ is the hidden feature for the $(l + 1)$-th layer with the initialization by normalized node attribute matrix $H_G^{(0)} = \hat{F}$. $W_G^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ and $W_{G_{\text{self}}} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ are the weight parameter matrices. $\sigma(\cdot)$ is the activation function, e.g., ReLU function $ReLU(x) = \max(0, x)$. b is the bias.

For the encoder output E_G, we remove the activation function since the activation is also applied in the feature fusion component, as well as E_S and E_A. Therefore, the graph embedding E_G is defined as:

$$E_G^{(l+1)} = \hat{A}H_G^{(l)}W_G^{(l)} + H_G^{(l)}W_{G_{\text{self}}}^{(l)} + b. \qquad (5)$$

Noted that, the input features of graph encoder is H_G, just the output of itself in the last layer. The embedding E_G is the input of feature fusion component.

C. Structure Encoder

The structure encoder provides an interface for query vertices and obtains the local structure embedding E_S for QD-GCN. The inputs of the structure encoder are based on the adjacency matrix A and structural input information I_S (w.r.t. the query node set $\mathcal{V}_q$). It encodes them to provide a structural view for a given query $(\mathcal{V}_q, \mathcal{F}_q)$, as shown in the orange part in Figure 2.

Similar to the graph encoder, we adopt GCN [20] as sketch to model the query-specific structural embedding:

$$H_S^{(l+1)} = \sigma(\hat{A}I_S^{(l)}W_S^{(l)} + I_S^{(l)}W_{S_{\text{self}}}^{(l)} + b), \qquad (6)$$

where $I_S^{(l)}$ is the input feature of structure encoder in the $(l + 1)$-th layer, $W_S^{(l)}$ and $W_{S_{\text{self}}}^{(l)}$ are the weight matrices.

Input Feature Construction. In order to input query vertices $\mathcal{V}_q$ to Eq.(6), we initialize the input feature I_S through the shortest path, since the shortest path information can reflect the topological closeness from any vertex to the query vertex set. Concretely, given a vertex $v_i \in \mathcal{V}$, the query distance of v_i, denoted by $\mathsf{dist}(v_i, \mathcal{V}_q)$, is the shortest path between v_i and $\mathcal{V}_q$, i.e., $\mathsf{dist}(v_i, \mathcal{V}_q) = \min_{v_q \in \mathcal{V}_q} \mathsf{dist}(v_i, v_q)$. If two vertices v_i and v_q are disconnected in graph G, the shortest distance $\mathsf{dist}(v_i, v_q) = +\infty$. Let the maximum distance $d_{max} = 1 + \max\{\mathsf{dist}(v_i, \mathcal{V}_q) : v_i \in \mathcal{V} \text{ connects with } V_q \text{ in } G\}$. Based on the query distance, we define $I_S^{(0)}$ as follows.

$$I_S^{(0)}(i) = \begin{cases} 1 & , \ v_i \in \mathcal{V}_q \\ 1 - \dfrac{\mathsf{dist}(v_i, \mathcal{V}_q)}{d_{max}} & , \ v_i \text{ and } \mathcal{V}_q \text{ are connected} \\ 0 & , \ v_i \text{ and } \mathcal{V}_q \text{ are disconnected} \end{cases}$$

$$(7)$$

With the definition of $I_S^{(0)}$ above, we have given an initialization of the structure weight $I_S^{(0)}(i)$ for all vertices v_i in $\mathcal{V}$. The structure weight depends on the closeness to query vertices. That is, the closer to query vertices, the higher the weight is and the higher the probability that belongs to the query community. When we multiply the adjacency matrix with these probabilities (i.e., $\hat{A}I_S$), their probabilities are passing to their neighbors accordingly.

For the intermediate layers, we assign the output of the feature fusion component H_{FF} as I_S to fuse features, which will be introduced in Section IV-E. The input feature of the structure encoder I_S is defined as:

$$I_S^{(l)} = \begin{cases} I_S^{(0)} & l = 0; \\ H_{FF}^{(l)} & \text{otherwise.} \end{cases} \tag{8}$$

The output of the structure encoder E_S is defined by $H_S^{(l+1)}$ without the activation function:

$$E_S^{(l+1)} = \hat{A}I_S^{(l)}W_S^{(l)} + I_S^{(l)}W_{S_{\text{self}}}^{(l)} + b. \tag{9}$$

D. Attribute Encoder

The attribute encoder provides the interface for query attributes $\mathcal{F}_q$ and produces the attribute embedding E_A for the related attribute features of queries. It takes the vertex attribute matrix F and query attribute set $\mathcal{F}_q$ as the input. In other words, the attribute encoder aims to figure out the underlying relationship among different attributes and find the related attribute of queries. In addition, the attribute encoder needs to represent such information in the form of vertices since the query community is formed by vertices but not by attributes.

To achieve the above functions, we construct a bipartite graph called the structure-attribute bipartite graph $BG(\mathcal{V}, \mathcal{F}, \mathcal{B}_E)$. This bipartite graph is formed by two node sets of graph vertices $\mathcal{V}$ and graph attributes $\mathcal{F}$. An edge between vertex v_i and feature f_j is added to the edge set $\mathcal{B}_E$, if and only if vertex v_i has the feature f_j, i.e., $F_{ij} > 0$. The edge weight is F_{ij}. Formally, the adjacency matrix of B is defined as:

$$A_{BG} = \begin{pmatrix} 0_{V,V} & B_V \\ B_F & 0_{F,F} \end{pmatrix}, \tag{10}$$

where $B_V = F \in \mathbb{R}^{n \times d}$ and $B_F = F^{\mathsf{T}} \in \mathbb{R}^{d \times n}$.

Figure 3 illustrates a toy example of the structure-attribute bipartite graph. The graph attributes are in green while the graph vertices are in orange. In Figure 3, we can observe that only one vertex (v_3) has both attributes A and B while two vertices (v_3 and v_4) have attributes B and D. The relation between B and D is closer than that of B and A.

In this vein, we construct a Bipartite Graph Convolutional Network (BGCN) [39] to model the attribute relationship based on the structure-attribute bipartite graph BG. Namely, the forward layer of BGCN is defined as:

$$\begin{aligned} H_V^{(l+1)} &= \sigma(B_V I_V^{(l)} W_V^{(l)}) \\ H_F^{(l+1)} &= \sigma(B_F I_F^{(l)} + H_F^{(l)} W_{F_{\text{self}}}^{(l)}), \end{aligned} \tag{11}$$
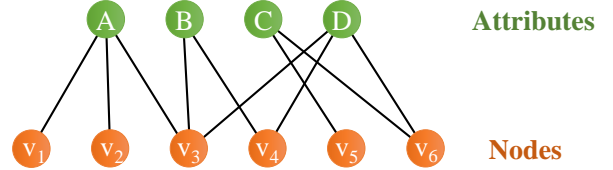

Fig. 3. An example of structure-attribute bipartite graph.

where $I_V^{(l)}$ and $I_F^{(l)}$ are the input feature of vertex side GCN and attribute side GCN respectively in the $(l+1)$-th layer, $H_V^{(l+1)}$ and $H_F^{(l+1)}$ are the learned node features and attribute features, and $W_V^{(l)}$ and $W_{F_{\text{self}}}^{(l)}$ are the weight matrices.

Input Feature Construction. Similar to the structure encoder, in order to input the query attribute set $\mathcal{F}_q$, we use $F_q \in \{0,1\}^{d \times 1}$ as the one-hot representation of $\mathcal{F}_q$ and initialize I_V with F_q. For other layers, the same as BGCN, we use the attribute side output H_F as the input feature of the node side GCN I_V:

$$I_V^{(l)} = \begin{cases} F_q & l = 0; \\ H_F^{(l)} & \text{otherwise.} \end{cases} \tag{12}$$

In the attribute side in BGCN in Eq. (11), we also initialize $H_F^{(0)}$ with F_q. For the input feature I_F, we apply the fusion result in feature fusion H_{FF} to I_F to deliver other types of features in this bipartite graph. This fusion operation will be introduced in Section IV-E. The input feature of the attribute side in the attribute encoder is defined as:

$$I_F^{(l)} = H_{FF}^{(l+1)}. \tag{13}$$

Note that we use the fused features in this layer, not the last layer, as the input of attribute side GCN.

Then Eq. (11) can be rewritten as:

$$\begin{aligned} H_V^{(l+1)} &= \sigma(B_V H_F^{(l)} W_V^{(l)}) \\ H_F^{(l+1)} &= \sigma(B_F H_{FF}^{(l)} + H_F^{(l)} W_{F_{\text{self}}}^{(l)}). \end{aligned} \tag{14}$$

Based on the above BGCN, we can represent attribute information in the form of vertices by the node side output H_V. The final attribute embedding E_A is calculated by removing the activation from H_V:

$$E_A^{(l+1)} = B_V H_F^{(l)} W_V^{(l)}. \tag{15}$$

E. Feature Fusion

The component of feature fusion combines three output features learned by the above three encoders, balances the global and local, structure and attribute information to get the final output of the entire QD-GCN model. The inputs of feature fusion are based on the output of three encoders, i.e., E_G, E_S and E_A. It fuses them and transmits the fusion result to the structure and attribute encoders as shown in the purple part in Figure 2.

Based on the output of three encoders, the forward layer of feature fusion is formulated as:

$$H_{FF}^{(l+1)} = \sigma(\mathsf{AGG}(E_G^{(l+1)}, E_S^{(l+1)}, E_A^{(l+1)})), \tag{16}$$

where $H_{FF}^{(l+1)}$ is the output of feature fusion and the entire QD-GCN model in the $(l+1)$-th layer, AGG$(\cdot)$ is the aggregation function (e.g., concatenation, sum, etc.), and $E_G^{(l+1)}$, $E_S^{(l+1)}$, $E_A^{(l+1)}$ are the outputs of each encoder in the $(l+1)$-th layer respectively.

In Eq.(16), we aggregate three encoders to fuse all types of information and obtain the final output. In order to consider the correlation between structure and attribute and process these two types of information simultaneously, we use H_{FF} as the input of the structure encoder and attribute encoder as Eq. (8) and Eq. (13) show.

For the structure encoder, this fusion operation transmits the query-specific attribute features and stable graph features into the structure encoder and delivers these features between vertices. For the attribute encoder, it enriches the features passed on the bipartite graph with local structure of query and global graph features. In this way, structure features and attribute features learned by QD-GCN will influence each other and these two encoders are not independent, thus QD-GCN is able to learn local structure and related attribute information of queries simultaneously.

For the graph encoder, we do not use the fusion result and just use the output of the graph encoder itself in the l-th layer $H_G^{(l)}$ as the input of the $(l+1)$-th layer. Thus, we can keep the graph encoder independent of any query information and maintain the robustness of the model.

F. Summary

Based on the above four encoder components, our QD-GCN model fuses one independent graph-level of community information and two correlated query-level information of structure and attribute. QD-GCN supports all information propagation in the cascades of structure and attributes from the entire graph to the local subgraph of queries. Moreover, QD-GCN provides an end-to-end multi-query community search model, which takes the queries as input and produces the communities as answers.

V. EXPERIMENTS

In this section, we present our experimental studies to validate the performance of QD-GCN on real graphs. We first introduce the setup of our experiment in detail in Section V-A. Then we present the performance of our QD-GCN in Section V-B. Furthermore, we conduct the ablation study in Section V-C to demonstrate the effectiveness of four components in QD-GCN. Finally, we perform case studies on two data sets in Section V-D to show its representation power in real applications.

A. Experimental Setup

1) Data Sets: To thoroughly evaluate the performance of QD-GCN, we conduct experimental studies on 9 attributed graphs. The graph statistics are presented in Table II. The first six networks: Cornell, Texas, Washington, Wisconsin, Cora and Citeseer, are publication citation networks. Each attribute describes the absence/presence of one word in this publication.

TABLE II
DATASET STATISTICS. $|\mathcal{V}|$ IS THE NUMBER OF NODES, $|\mathcal{E}|$ IS THE NUMBER OF EDGES, $|\mathcal{F}|$ IS THE NUMBER OF DISTINCT ATTRIBUTES, K IS THE NUMBER OF COMMUNITIES AND AS IS THE AVERAGE SIZE OF COMMUNITIES.

Data set		$|\mathcal{V}|$	$|\mathcal{E}|$	$|\mathcal{F}|$	K	AS
Cornell		195	283	1703	5	39
Texas		187	280	1703	5	37.4
Washington		230	366	1703	5	46
Wisconsin		265	459	1703	5	53
Cora		2708	5278	1433	7	386.86
Citeseer		3312	4536	3703	6	552
Facebook Ego-networks	0	348	2852	224	24	13.54
	107	1046	27783	576	9	55.67
	1684	793	14810	319	17	45.71
	1912	756	30772	480	46	23.15
	3437	548	5347	262	32	6
	348	228	3416	161	14	40.5
	414	160	1843	105	7	25.43
	686	171	1824	63	14	34.64

All these graphs can be found at the website[1]. Facebook [40] is a social network where nodes are users and edges are friend relationships. It contains 8 ego-networks shown in Table II. We consider each ego-network as an independent data set. All data sets contain the groundtruch communittes.

2) Approaches Compared: We compare our QD-GCN with four state-of-the-art approaches, including two non-attributed community search algorithms of CTC [21] and k-ECC [18] and two attributed community search algorithms of ACQ [4] and ATC [5] as follows.

- CTC [21] is a k-truss based model of community search for multiple given query vertices. The algorithm first identifies the maximal k-truss containing query vertices with the largest k. Then, it removes the vertices/edges far from query vertices, and returns one immediate subgraph of k-truss with the smallest diameter as an answer, which does not consider the effect of attribute. we take the automatic parameter setting in [21]. The code is implemented by C/C++ and provided by authors.
- k-ECC [18] is a k-ECC based model of community search for multiple given query vertices. The algorithm finds the k-edge connected components containing query vertices as answers. The approach takes the input of only query vertices but no query attributes. We use the default parameter setting in [1] with C/C++ implementation.
- ACQ [4] is a k-core based model of attributed community search for one single query vertex. The algorithm first finds the maximal k-core containing the query vertex, and shrinks it by maximizing the number of query attributes for answers. This approach fails to use the correlation between attributes, and cannot deal with multiple query vertices. We set up the input parameter k as the largest value of k such that there exists at least one feasible community for the given query vertex and query attributes. ACQ is implemented by java with code given by [1].

[1]https://linqs.soe.ucsc.edu/data

- ATC [5] is a k-truss based model of attributed community search for multiple query vertices. The algorithm first finds the maximal k-truss subgraph containing the query vertices with the constrain that distances between subgraph members and query vertices are less than d. Then it shrinks the subgraph by maximizing a pre-defined attribute score function. This approach doesn't consider the correlation between attributes. ATC takes the automatic parameter setting in [5]. The algorithm is implemented by C/C++ and provided by its authors.

3) Query Generation: For each graph, we generate 350 pairs of input query $< \mathcal{V}_q, \mathcal{F}_q >$ and the corresponding ground truth community $\mathcal{C}_q$ in data sets. To generate $\mathcal{V}_q$, given ground-truth community $\mathcal{C}_q$, we randomly select node sets containing 1-3 nodes (or just one node to compare with ACQ) as the query node set. To generate $\mathcal{F}_q$, in order to compare with the state-of-the-arts, we generate three types of $\mathcal{F}_q$.

- Attribute query (attributes from communities). Following the setting of previous works [4], [5], we select query attributes from ground-truth communities. We add up the attribute vectors of all nodes in graph and in different communities separately. We choose 1-5 attributes with high value ranking in community but lower ranking in the entire graph as community attributes. The query attributes $\mathcal{F}_q$ is the ground-truth community attributes we choose.
- Attribute query (attributes from nodes). To make a deep comparison, we simulate the real queries provided by users without the knowledge of ground-truth community attributes. We add up the attribute vectors of all query nodes $q \in \mathcal{V}_q$ and choose 5 attributes with the highest value as query attributes $\mathcal{F}_q$. These attributes may be not the right attributes corresponding to the community attributes.
- Non-attribute query. To compare with those methods not supporting attribute query, we set the attribute query set as empty ($\mathcal{F}_q = \varnothing$).

4) Data Separation: We separate the 350 query-community pairs into training data, validation data and test data with the proportion of 150:100:100. We use the training data to train our model and validation data to choose the best weights during the process of training. Test data is to measure the performance of all methods.

5) Evaluation Metrics: Let $D_{\text{test}} = \{\mathcal{Q}, \hat{\mathcal{Z}}, \mathcal{Y}\}$ be the test data set, where $\mathcal{Q}$ is the query set, $\hat{\mathcal{Z}}$ is the corresponding predicted community set by a method and $\mathcal{Y}$ is the ground-truth community set. To measure the quality of communities found by different methods, we employ two measurements: F_1 score and Jaccard similarity to evaluate the quality of the predicted set $\hat{\mathcal{Z}}$.

- F_1 score is defined as:

$$F_1(\hat{\mathcal{Z}}, \mathcal{Y}) = \frac{2 \cdot pre(\hat{\mathcal{Z}}, \mathcal{Y}) \cdot rec(\hat{\mathcal{Z}}, \mathcal{Y})}{pre(\hat{\mathcal{Z}}, \mathcal{Y}) + rec(\hat{\mathcal{Z}}, \mathcal{Y})}$$

where $pre(\hat{\mathcal{Z}}, \mathcal{Y})$ is the precision of predicted community set $\hat{\mathcal{Z}}$ on the ground-truth community set $\mathcal{Y}$ as:

$$pre(\hat{\mathcal{Z}}, \mathcal{Y}) = \frac{\sum_{\hat{z}_q \in \hat{\mathcal{Z}}} \hat{z}_q \& y_q}{\sum_{\hat{z}_q \in \hat{\mathcal{Z}}} \sum_{i=0}^{n} \hat{z}_{qi}}.$$

$rec(\hat{\mathcal{Z}}, \mathcal{Y})$ is the recall of the output communities:

$$rec(\hat{\mathcal{Z}}, \mathcal{Y}) = \frac{\sum_{\hat{z}_q \in \hat{\mathcal{Z}}} \hat{z}_q \& y_q}{\sum_{y_q \in \mathcal{Y}} \sum_{i=0}^{n} y_{qi}}.$$

- Jaccard similarity is defined as:

$$Jac(\hat{\mathcal{Z}}, \mathcal{Y}) = \frac{\sum_{\hat{z}_q \in \hat{\mathcal{Z}}} z_q \& y_q}{\sum_{\hat{z}_q \in \hat{\mathcal{Z}}} \hat{z}_q | y_q}.$$

Here, $\hat{z}_q \in \{0, 1\}^{n \times 1}$ and $y_q \in \{0, 1\}^{n \times 1}$ are the predicted and ground-truth community vectors for query q respectively.

6) Implementation Details: For our model, we build three layers with 128 neurons in the hidden layer. We train 300 iterations with learning rate 0.001. For the final output z_q, we apply a threshold $\gamma \in (0, 1)$ to filter out the output community set $\mathcal{C}_q$. If $z_{qi} \geq \gamma$, v_i belongs to the output community, then $\hat{z}_{qi} = 1$. We choose γ to a value that achieves the best performance on the validation query set. In the feature fusion component, we choose concatenate as the aggregation fuction in Eq. (16)

In each layer, we employ batch normalization and dropout except the output layer.

- Batch Normalization. After the active function in each layer, we apply the batch normalization [41] with batch size 4 to keep the distribution of the input in each layer. In our QD-GCN model, there are four components. We add the batch normalization ($BN(\cdot)$) at the end of them. For graph encoder, structure encoder and feature fusion, we apply the $BN(\cdot)$ in the layer-wised equation (4), (6) and (16). For attribute encoder, we only implement $BN(\cdot)$ in the attribute side layer-wised equation as:

$$H_F^{(l+1)} = BN(\sigma(B_F H_{FF}^{(l+1)} + H_F^{(l)} W_{F_{\text{self}}}^{(l)})).$$

- Dropout. We add a dropout function for inputs of three encoders with 0.5 dropout rate in each layer. That is, we add dropout at $H_G^{(l)}$, $I_S^{(l)}$, $I_F^{(l)}$ and $I_V^{(l)}$ in the beginning of $(l + 1)$-layer.

B. Community Search Performance

We present comprehensive performance studies to validate the query performance of QD-GCN model under both attributed community search and non-attributed community search setting. In addition, we also analyze the efficiency for this online query problem.

Attributed community search. QD-GCN is compared with two attributed community search algorithms: ACQ and ATC. To make a fair comparison, we generate one-node queries when comparing with ACQ and multi-node queries when comparing with ATC. Regarding the query performance, Table

TABLE III
ATTRIBUTED COMMUNITY SEARCH PERFORMANCE COMPARED WITH OTHER APPROACHES (QUERY ATTRIBUTES FROM COMMUNITIES).

Setting	Method	FB-414		FB-686		FB-348		FB-0		FB-3437		FB-1912		FB-1684	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes & Multi query attributes	QD-GCN	**0.894**	**0.808**	**0.795**	**0.660**	**0.915**	**0.888**	**0.941**	**0.888**	**0.904**	**0.825**	**0.939**	**0.885**	**0.789**	**0.652**
	ATC	0.593	0.422	0.411	0.258	0.406	0.254	0.293	0.171	0.252	0.144	0.427	0.271	0.444	0.282
One query node & Multi query attributes	QD-GCN	**0.913**	**0.839**	**0.837**	**0.720**	**0.869**	**0.768**	**0.907**	**0.831**	**0.878**	**0.783**	**0.952**	**0.909**	**0.903**	**0.923**
	ACQ	0.564	0.393	0.349	0.211	0.502	0.335	0.289	0.169	0.106	0.056	0.325	0.195	0.290	0.170

Setting	Method	FB-107		Cornell		Texas		Washington		Wisconsin		Cora		Citeseer	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes & Multi query attributes	QD-GCN	**0.996**	**0.992**	**0.986**	**0.973**	**0.980**	**0.961**	**0.926**	**0.862**	**0.994**	**0.982**	**1.000**	**1.000**	**1.000**	**0.999**
	ATC	0.3861	0.239	0.218	0.122	0.308	0.182	0.275	0.159	0.147	0.079	0.049	0.025	0.025	0.013
One query node & Multi query attributes	QD-GCN	**0.997**	**0.993**	**0.913**	**0.841**	**0.970**	**0.943**	**0.868**	**0.768**	**0.943**	**0.891**	**0.996**	**0.993**	**0.808**	**0.605**
	ACQ	0.422	0.268	0.428	0.272	0.422	0.268	0.433	0.276	0.425	0.270	0.106	0.056	0.105	0.055

TABLE IV
ATTRIBUTED COMMUNITY SEARCH PERFORMANCE COMPARED WITH OTHER APPROACHES (QUERY ATTRIBUTES FROM NODES).

Setting	Method	FB-414		FB-686		FB-348		FB-0		FB-3437		FB-1912		FB-1684	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes & Multi query attributes	QD-GCN	**0.855**	**0.747**	**0.647**	**0.479**	**0.680**	**0.515**	**0.822**	**0.698**	**0.763**	**0.616**	**0.655**	**0.487**	**0.604**	**0.433**
	ATC	0.585	0.414	0.399	0.250	0.419	0.265	0.276	0.160	0.262	0.151	0.453	0.293	0.444	0.286
One query node & Multi query attributes	QD-GCN	**0.777**	**0.635**	**0.607**	**0.436**	**0.684**	**0.519**	**0.673**	**0.507**	**0.681**	**0.516**	**0.500**	**0.333**	**0.589**	**0.417**
	ACQ	0.517	0.348	0.386	0.239	0.541	0.371	0.283	0.165	0.133	0.071	0.291	0.170	0.280	0.163

Setting	Method	FB-107		Cornell		Texas		Washington		Wisconsin		Cora		Citeseer	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes & Multi query attributes	QD-GCN	**0.935**	**0.862**	**0.974**	**0.949**	**0.976**	**0.952**	**0.985**	**0.971**	**0.973**	**0.947**	**0.794**	**0.658**	**0.709**	**0.549**
	ATC	0.358	0.218	0.175	0.095	0.267	0.154	0.033	0.017	0.180	0.098	0.034	0.017	0.024	0.012
One query node & Multi query attributes	QD-GCN	**0.872**	**0.774**	**0.955**	**0.915**	**0.977**	**0.955**	**0.945**	**0.890**	**0.967**	**0.936**	**0.735**	**0.581**	**0.683**	**0.518**
	ACQ	0.390	0.242	0.286	0.167	0.352	0.213	0.262	0.151	0.322	0.192	0.072	0.037	0.078	0.041

III and Table IV list the F1-score (F1) and Jaccard similarity (Jaccard) performance for all data sets.

Table III shows the comparison results on multiple data sets, in which the query attributes are selected from communities. The size of communities varies a lot among different data sets. ATC and ACQ fail to fit all the data sets well. For Cora and Citeseer with large communities, the performances of ATC and ACQ are quite poor (< 0.05 and ~ 0.1 in F1-score, respectively). It is because their pre-defined community patterns (i.e. k-core and k-truss) are too strict and narrow to fit large communities in the real-world graphs. Notice that the core-based condition of ACQ is too board so that it also performs poorly on the data set with extremely small communities (e.g. FB-3437). Our proposed QD-GCN, on the contrary, consistently performs excellently on all the data set, with at least 0.3 improvements against ATC and ACQ. As a data-driven approach, QD-GCN is capable of adapting to all communities and performs stable on all graphs benefiting from learning adaptive weight matrices for different data sets.

Table IV shows the comparison results in which queries attributes are selected from nodes. Different from Table III where the query attributes are from the ground-truth communities, the query attributes selecting method in Table IV may encounter some bad queries. The method of selecting attributes from nodes makes it possible that the selected query attributes do not correspond to the ground-truth communities at all. For example, in Washington data sets, ATC achieves 0.275 F1-score when the query attributes are selected from communities in Table III. While, in Table IV where the query attributes are selected from query nodes, ATC only achieves 0.033 and always searches empty sets due to the restricted requirements for communities. Even in this challenging scenario, QD-GCN still improves the query performance significantly over all the tested data. Specifically, compared with ACQ and ATC, it achieves 0.461 and 0.537 average improvements respectively in terms of F1-score. It demonstrates that QD-GCN can adapt to various queries and performs more stable on all data sets.

Non-attributed community search. In order to compare with non-attributed community search algorithms, we take the same multi-node queries set from the attributed community search setting and remove the query attributes. For QD-GCN, we feed all-zero vectors as query attributes. Table V shows the performance of QD-GCN, CTC and ECC in terms of F1 score and Jaccard similarity. QD-GCN also outperforms both approaches in all data sets. Interestingly, we can find that the non-attribute results of QD-GCN are comparable to the attribute results in Table IV. This may owe to the fusion operation in QD-GCN. Even though there is no attribute input in the first layer of attribute encoder, the fusion operation

TABLE V
Non-attributed community search performance compared with other approaches.

Setting	Method	FB-414		FB-686		FB-348		FB-0		FB-3437		FB-1912		FB-1684	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes	QD-GCN	**0.867**	**0.765**	**0.703**	**0.542**	**0.705**	**0.544**	**0.680**	**0.51**	**0.670**	**0.504**	**0.698**	**0.536**	**0.632**	**0.463**
&	CTC	0.505	0.337	0.381	0.236	0.347	0.210	0.248	0.142	0.213	0.119	0.492	0.327	0.403	0.252
No query attributes	ECC	0.431	0.275	0.542	0.371	0.685	0.521	0.326	0.195	0.105	0.055	0.316	0.187	0.299	0.176

Setting	Method	FB-107		Cornell		Texas		Washington		Wisconsin		Cora		Citeseer	
		F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard
Multi query nodes	QD-GCN	**0.800**	**0.667**	**0.777**	**0.635**	**0.840**	**0.725**	**0.882**	**0.788**	**0.876**	**0.778**	**0.676**	**0.511**	**0.574**	**0.403**
&	CTC	0.352	0.213	0.081	0.042	0.044	0.022	0.043	0.022	0.019	0.010	0.014	0.007	0.004	0.002
No query attributes	ECC	0.334	0.201	0.218	0.123	0.281	0.164	0.286	0.167	0.286	0.167	0.196	0.109	0.123	0.065

TABLE VI
Average query time (in milliseconds) of different community search methods.

| Method	FB-414	FB-686	FB-348	FB-0	FB-3437	FB-1912	FB-1684	FB-107	Cornell	Texas	Washington	Wisconsin	Cora	Citeseer
CTC	34.78	41.41	120.92	49.25	131.67	4903.38	604.67	2498.82	0.45	0.40	0.42	0.63	1.96	1.28
ECC	3.52	2.29	5.20	2.57	6.60	154.23	26.85	93.62	0.24	0.28	0.23	0.33	2.67	1.76
ACQ	0.00	0.00	1.45	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ATC	4.40	5.10	7.70	5.90	10.30	43.60	22.70	40.10	7.90	14.02	2.11	18.60	11.49	3.68
QD-GCN	4.08	3.40	5.04	3.70	4.48	4.46	5.60	5.44	4.45	4.49	5.50	5.23	5.54	5.25
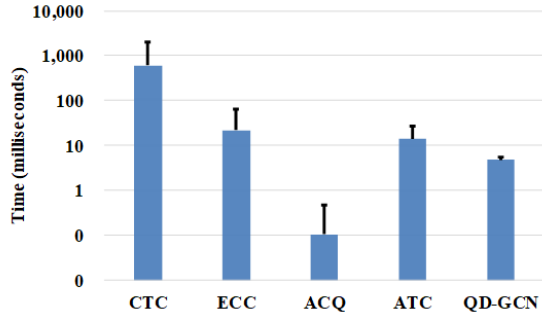
Fig. 4. Mean and variance of query time in different community search methods.

can transmit other information, such as graph information and query vertexes information, to attribute encoder in the second layer. Therefore, QD-GCN can make use of graph attribute information in non-attributed community search problem on attributed graphs. It also validates that correlations between the community structure and the node attributes is meaningful for attributed community search and can not be ignored.

Query Efficiency. To evaluate the efficiency of QD-GCN, we also test the response time of the queries in the test set. Table VI shows the average query time of the test set across each data set with our QD-GCN and other baselines. It is worth noting that QD-GCN achieves a stable performance efficiency of around 5 milliseconds among all data sets. However, the query times of CTC, ECC and ATC increased considerably on relatively large graphs like FB-1912, FB-1684 and FB-107. To be more specific, CTC takes almost 5,000 milliseconds for a query on FB-1912, while QD-GCN only costs 4.46 milliseconds. Figure 4 shows the mean and variance of query time among all data sets. As an online community search

problem, the average query time of QD-GCN is less than 5 milliseconds which is much faster than that of CTC, ECC and ATC. Although ACQ is faster, the method oversimplifies the attributed community search problem that only vertexes' degrees and common attributes are taken into consideration and they are processed separately. It completely ignores other information in the graph such as attribute similarities and hidden structure-attribute relationships. The community qualities shown in Table III and Table IV are much lower than those of QD-GCN.

C. Ablation Study

In this section, we discuss more detailed analysis about our QD-GCN. The results are all under the attributed community search setting.

1) Ablation Study for Model Components: To verify the necessity of every component in QD-GCN, we defer more discussions regarding to the impacts of the key components in QD-GCN, graph encoder, structure encoder, attribute encoder and feature fusion. We report all results in Figure 5.

Graph Encoder. To verify the importance of graph encoder, we construct a QD-GCN-noGE without graph encoder and compare it with the original QD-GCN. The original QD-GCN outperforms the model without graph encoder by 0.2 to 0.6 in F1-score. Thus graph encoder is an important component that provides the base graph information to the whole model and keeps the model stable from the query.

Structure Encoder. Similar with QD-GCN-noGE, we also construct a QD-GCN-noSE without structure encoder. All scores of QD-GCN-noSE are lower than those of QD-GCN model, which proves the necessity of the structure encoder. It is worth noting that for some graphs like Texas and Wisconsin, the loss of structure encoder does not decrease the scores
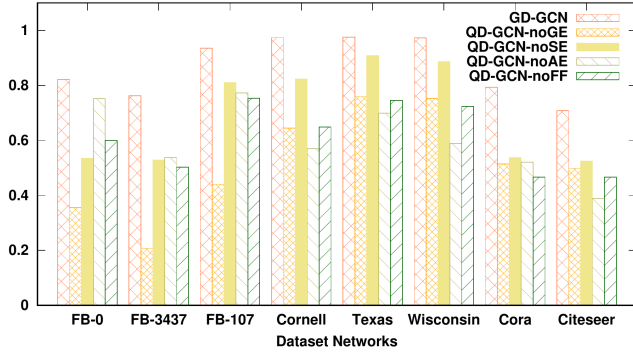
Fig. 5. F1-score of QD-GCN model in ablation study.

Fig. 6. Comparison of different aggregation function.

too much, possibly because communities in those graphs are formed with more influence from attribute than from structure.

Attribute Encoder. QD-GCN-noAE represents a model without attribute encoder. The performances in Figure 5 are worse than those of QD-GCN model, which proves that the attribute encoder is also an importance part of QD-GCN. Similar to the structure encoder, we can observe that the attribute encoder is more important than the structure encoder for Wisconsin graph. However, for FB-0, the structure encoder is more necessary. Therefore, the framework of QD-GCN can balance the attribute and structure for each graph.

Feature Fusion. In our model, we use the aggregation result H_{FF} in Eq. (16) to fuse all information and assign it to the structure and attribute encoder in Eq. (8) and Eq. (13). To verify the effectiveness of this fusion operation, we compare the original QD-GCN with QD-GCN-noFF where the three encoders don't aggregate in the hidden layer. They only aggregate in the last layer to output the final communities. $I_S^{(l)}$ and $I_F^{(l)}$ are just $H_S^{(l)}$ and $H_V^{(l+1)}$ in Eq. (8) and (13). The comparison results are shown in Figure 5. The original fusion operation outperforms the QD-GCN-noFF by 0.2 to 0.4 in all graphs. Therefore, the aggregation of three encoders and fusing information to the structure and attribute encoders in each layer can significantly improve the results.

2) Ablation Study for Aggregation Function: In feature fusion component, there is an aggregation function $\text{AGG}(\cdot)$ to aggregation three different features in Eq. (16). We can choose concatenate, add, max, min or average as the operation of aggregation. The comparison of different aggregation function in graphs is shown in figure 6. We adopt attribute query setting in Sec.V, which generates attribute queries from query nodes. The concatenate performs best in these graphs, which retains all information from three encoders. The max function gains the lowest F1-score, since the output of graph encoder in the first layer is always the largest and max function will lose all query information.

3) The impact of threshold γ: When translating the model output vector z_q from $\mathbb{R}^n$ to $\{0,1\}^n$, we need a threshold $\gamma \in (0,1)$ such that if $z_{qi} \geq \gamma$, then set $\hat{z}_{qi} = 1$, otherwise $\hat{z}_{qi} = 0$. In the experiments, we choose γ which achieves the best performance in the validation set. To analyze the influence
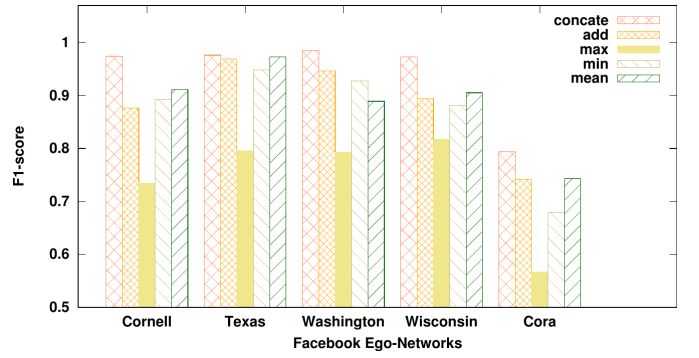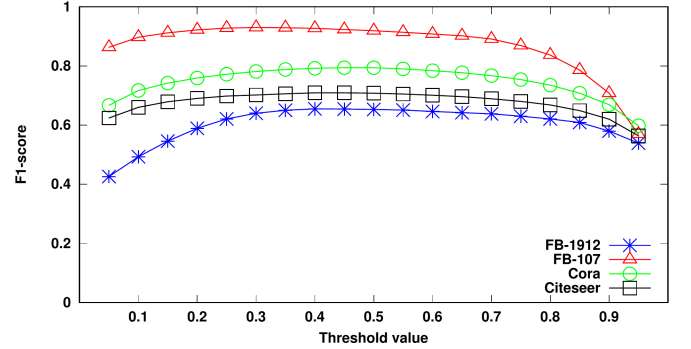

Fig. 7. F1-score of various threshold γ in ablation study.

of the threshold, we vary γ from 0.05 to 0.95 and the F1 score is shown in Figure 7. When γ is between 0.3 and 0.7, there is almost no fluctuation in performance. Therefore, QD-GCN is not very sensitive to the selection of this threshold.

D. Case Study

We perform case studies on two data sets in Table II to investigate the ability of QD-GCN in representing the community search queries. For the queries of one data set, we obtain an embedding from the output of QD-GCN, i.e., the concatenation of the outputs from the graph encoder, structure encoder and attribute encoder in the last layer. We use t-SNE [42] to visualize the high-dimensional embedding for FB-1684 (Figure 8(a)) and FB-1912 (Figure 8(b)) networks. Here, nodes indicate the queries and the color of the nodes indicates the ground-truth community labels where queries are generated

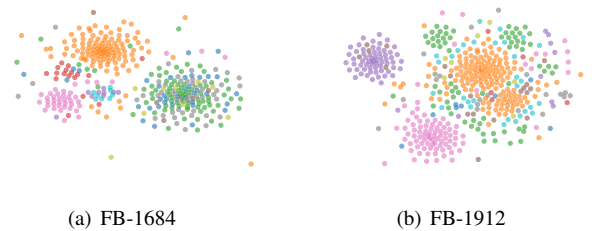(a) FB-1684   (b) FB-1912

Fig. 8. The t-SNE visualization of the output embedding of QD-GCN for two data sets. The color indicates the ground-truth community label.

from. According to Figure 8, we can observe that the queries which are generated from one community tend to cluster in the vector space. This reflects that QD-GCN effectively extracts the common features of one community, supervised by the ground-truth community label. An acute reader may find that in Figure 8(a), the bottom right cluster is mixed by multi-color queries. That is because in social network communities, e.g., Facebook, a larger community is usually composed of many overlapping small groups. The nodes in the small groups share a couple of similar attributes, e.g., user interest. The similar phenomenon also appears in Figure 8(b), where the upper right community is surrounded by small groups.

VI. CONCLUSIONS

In this paper, we concentrate on the attributed community search problem, specifically, finding cohesive subgraph with coherent attribute for a given set of query nodes and query attributes. We propose Query-Diven Graph Convolutional Network (QD-GCN), an end-to-end supervised learning model. To the best of our knowledge, we are the first to propose a novel graph neural network model. Our model effectively extracts the global graph features, query-specific local topology and the query-related attribute features by four components, a graph encoder, structure encoder, attribute encoder and feature fusion jointly. The experiments demonstrate that QD-GCN has the ability to inference on unseen queries with high quality on the real-world graph, outperforming previous approaches significantly. As a learning based model, QD-GCN provides a promising solution for attributed community search.

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," *CoRR*, vol. abs/1904.12539, 2019.

[2] X. Huang, L. V. S. Lakshmanan, and J. Xu, *Community Search over Big Graphs*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019.

[3] X. Huang, L. V. Lakshmanan, and J. Xu, "Community search over big graphs: Models, algorithms, and opportunities," in *ICDE*, 2017, pp. 1451–1454.

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

[5] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *PVLDB*, vol. 10, no. 9, pp. 949–960, 2017.

[6] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[7] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *PVLDB*, vol. 2, no. 1, pp. 718–729, 2009.

[8] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *NeurIPS*, 2017, pp. 373–382.

[9] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, "Attributed graph clustering: A deep attentional embedding approach," in *IJCAI*, 2019, pp. 3670–3676.

[10] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *KDD*, 2017, pp. 665–674.

[11] D. Szklarczyk, A. Franceschini, S. Wyder, K. Forslund, D. Heller, J. Huerta-Cepas, M. Simonovic, A. Roth, A. Santos, K. P. Tsafou *et al.*, "String v10: protein–protein interaction networks, integrated over the tree of life," *Nucleic acids research*, vol. 43, no. D1, pp. D447–D452, 2015.

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

[13] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, 2014, pp. 991–1002.

[14] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, 2014, pp. 1311–1322.

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

[16] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.

[17] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based densest clique percolation community search in networks," *TKDE*, vol. 30, no. 5, pp. 922–935, 2017.

[18] 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.

[19] 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.

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

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

[22] X. Huang, H. Cheng, and J. X. Yu, "Dense community detection in multi-valued attributed networks," *Information Sciences*, vol. 314, pp. 77–99, 2015.

[23] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *CIKM*. ACM, 2017, pp. 889–898.

[24] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective," in *WWW*, 2019, pp. 972–982.

[25] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *NeurIPS*, 2018, pp. 4558–4567.

[26] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[27] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*, 2018, pp. 974–983.

[28] X. Wang, X. He, Y. Cao, M. Liu, and T. Chua, "KGAT: knowledge graph attention network for recommendation," in *KDD*, 2019, pp. 950–958.

[29] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *ICML*, 2019, pp. 695–704.

[30] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *KDD*, 2019, pp. 1399–1407.

[31] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *ICML*, 2019, pp. 3734–3743.

[32] H. Gao and S. Ji, "Graph u-nets," in *ICML*, 2019, pp. 2083–2092.

[33] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *KDD*, 2019, pp. 723–731.

[34] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *NeurIPS*, 2017, pp. 6530–6539.

[35] Y. Li, C. Sha, X. Huang, and Y. Zhang, "Community detection in attributed graphs: An embedding approach," in *AAAI*, 2018, pp. 338–345.

[36] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang, "A vectorized relational graph convolutional network for multi-relational network alignment," in *IJCAI*, 2019, pp. 4135–4141.

[37] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou, "End-to-end structure-aware convolutional networks for knowledge base completion," in *AAAI*, 2019, pp. 3060–3067.

[38] A. Goel, K. T. Ma, and C. Tan, "An end-to-end network for generating social relationship graphs," in *CVPR*, 2019, pp. 11 186–11 195.

[39] C. He, T. Xie, Y. Rong, W. Huang, J. Huang, X. Ren, and C. Shahabi, "Adversarial representation learning on large-scale bipartite graphs," *arXiv preprint arXiv:1906.11994*, 2019.

[40] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *NeurIPS*, 2012, pp. 548–556.

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[42] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.