ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230358

Effective and Efficient Community Search with Graph Embeddings

Xiaoxuan Gou[a], Xiaoliang Xu[a], Xiangying Wu[a], Runhuai Chen[a], Yuxiang Wang[a;*], Tianxing Wu[b] and Xiangyu Ke[c]

[a]Hangzhou Dianzi University, School of Computer Science, China
[b]Southeast University, School of Computer Science and Engineering, China
[c]Zhejiang University, School of Software Technology, China

Abstract. Given a graph G and a query node q, community search (CS) seeks a cohesive subgraph from G that contains q. CS has gained much research interests recently. In the database research community, researchers aim to find the most cohesive subgraph satisfying a specific community model (e.g., k-core or k-truss) via graph traversal. These works obtain good precision, however suffering from the low efficiency issue. In the AI research community, a new thought of using the deep learning model to support CS without relying on graph traversal emerges. Supervised end-to-end models using GCN are presented, which perform efficiently, but leave a large room for precision improvement. None of them can achieve a good balance between the efficiency and effectiveness. This motivates our solution: First, we present an offline community-injected graph embedding method to preserve the community's cohesiveness features into the learned node representations. Second, we resort to a proximity graph (PG) built from node representations, to quickly return the community online. Moreover, we develop a self-augmented method based on KL divergence to further optimize node representations. Extensive experiments on seven real-world graphs show our solution's superiority on effectiveness (at least 39.3% improvement) and efficiency (one to two orders of magnitude faster).

1 Introduction

Graph is a prevalent model to represent entities (i.e., nodes) and their relationships (i.e., edges) in many of today's real-world information networks [56, 16, 47, 34, 7], e.g., social networks, collaboration networks, and criminal networks. For example, authors in a collaboration network are represented as nodes and the co-authorship between authors is indicated by an edge. Graph-based data analytics has attracted extensive research interests [38], among which *community search* (CS) is a fundamental problem [54, 6]. Given a graph G and a query node q, CS aims to find a cohesive subgraph from G that contains q. CS has been widely applied in many real-life applications [36], such as event organization [44], influence spread [30], graph clustering [21], and biological data analysis [13]. Currently, research on the CS problem can be classified into two main categories.

Cohesiveness-aware CS. CS has been widely studied in the database research community. The key of such work is to define an appropriate community model to measure the community's cohesiveness [18, 10]. A community is often modeled as a k-core (denoted by H_k) of which each node has a degree at least of k [12, 44]. As shown in Figure 1, all nodes belong to a 1-core H_1 as every node has at least one neighbor; H_2 includes nodes except v_{15}; H_3 is the combination of two connected components $\hat{H}_3^1$ and $\hat{H}_3^2$, we use $\hat{H}_k^i$ to denote the i-th connected component of H_k (also known as a connected k-core, defined in Definition 2). The larger the k is, the more cohesiveness the community has. Besides k-core, many models such as k-truss [26], k-ECC [23], and k-clique [11] have also been considered for CS. More details are introduced in our related work (§6). The biggest advantage of such work is that the definition of community is flexible. Users select an existent or define a new community model with a large enough k, according to the cohesiveness requirement of their applications [20]. No matter which model is used, the essence is to find the subgraph satisfying the specific model via a heavy weight graph traversal method. Even some dedicated prune strategies for speed-up purpose are designed, it is still time-consuming especially for large graphs, e.g., from hundreds of milliseconds to tens of seconds to respond [47, 10, 54]. This motivates an emerging research area: Artificial Intelligence (AI)-enabled CS.

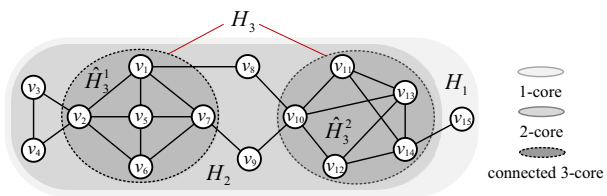


Figure 1. An example of k-core on an undirected graph

AI-enabled CS. A new thought is to leverage the deep leaning model to directly support CS without relying on the graph traversal with prune. Recently, [28, 27] design a supervised end-to-end model using the graph convolutional network (GCN) to map a query node q (represented as a d-dimensional vector) into a set of nodes (represented as a community vector where each dimension has a value of 1 or 0 to indicate a node belonging to the same community of q or not). In this solution, ground-truths (i.e., human-annotated communities) are required as the supervised information, which help update the model parameters by considering the differences between the output community vector and the ground-truth. The CS's efficiency is improved to some extent as the time-consuming graph traversal is skipped. Instead, a well-trained AI model is deployed to return communities immediately. However, **it still has three main**

* Corresponding Author. Email: lsswyx@hdu.edu.cn

drawbacks. **First**, *it leaves much room for effectiveness improvement* (e.g., the best F1-score is only 54% on Cora dataset). **Second**, *it requires prior-knowledge of existing communities as supervised information for training*, which is impractical especially for applications with cold start problems. **Finally**, *the output vector only points out the involvement of a community (i.e., 0-1 value of each dimension) but ignores the inherent relationship between nodes*, e.g., the similarity of a node to the query node. So, it is impossible to leverage the plain output vector for more meaningful downstream tasks, e.g., the recommendation with priority that recommends more high-quality items to users who are more similar to the query user from the same community.

Our solution. With the aforementioned methods' pros and cons in mind, we present a general CS framework with graph embeddings in §3.1 that involves an effective offline learning stage and an efficient online search stage, which *achieves a good balance between effectiveness and efficiency*. For offline learning stage, we first take k-core as the community model to present a community-injected graph embedding method based on a community-oriented triplet sampling to preserve the community's cohesiveness features into learnable node representations (§3.2), so that two nodes are similar in terms of their representations if they belong to the same community. Next, we enable the above method to be scalable with various cohesiveness parameter k via a lightweight triplet sampling with little overhead (§3.3). *To the best of our knowledge, we are the first to embed the community cohesiveness features into the node representations, making it possible to support CS by directly measuring a node's similarity to the query node q.* Thus, for online search stage, we resort to a proximity graph (PG) built from the node representations, to return the most similar nodes to q, as the desired community (§3.4). Moreover, we develop a self-augmented method based on KL divergence to further optimize the node representations in §4.

Contributions. Our solution's superiority is four-folded. **First**, comparing with the existing AI solutions, we can well embed the community cohesiveness features into the node representations, thus significantly improve the CS's effectiveness (at least 39.3% improvement on F1-score). **Second**, we take k-core as the community model and do not rely on the supervised ground-truths. It can easily extend to other models, such as k-truss and k-ECC, by simply replacing k-core community with others in §3.2-3.3. **Third**, thanks to the excellent performance of PG on information retrieval [52], our online search stage is more efficient (e.g., 0.323 ms per query on average) compared with the methods in both DB and AI area. **Finally**, we provide the community nodes with their representations, so we can obtain the similarity between nodes and leverage it for more meaningful tasks. Our contributions can be summarized as follows.

- We present a CS framework with graph embeddings involving an offline learning and an online search stage.
- For offline stage, we propose a k-core injected graph embedding method to preserve the cohesiveness features for arbitrary k. For online stage, we build a PG atop node representations to quickly return communities.
- We optimize the node representations through a self-augmented method based on KL divergence.
- Extensive experiments on 7 real-world graphs show that ours has up to one to two orders of magnitude faster than existing solutions, and achieves an acceptable F1-score of 87.89% on average (at least 39.3% improvement compared to existing AI solutions).

2 Problem Definition

We consider an undirected graph $G = (V, E)$ with a node set V and an edge set E. Given a node $v \in V$, we denote its neighbors by $N(v)$ and use $deg(v)$ to indicate v's degree, i.e., $deg(v) = |N(v)|$. When the context has ambiguity, we will clarify the degree in a certain graph as $deg(v, G)$. Next, we introduce the k-core model that is widely used to represent a community [2, 19].

Definition 1 (k-core [5]). *Given a undirected graph $G = (V, E)$ and a non-negative integer k, a k-core of G is the largest subgraph $H_k \subseteq G$, such that $\forall v \in H_k$ has degree at least k, i.e., $deg(v, H_k) \geq k$.*

Note that, the larger the k is, the more the cohesiveness H_k has. However, a k-core may be a disconnected subgraph which is not in line with the inherent connectivity constraint of a community, so we define the connected k-core as follows.

Definition 2 (Connected k-core [47]). *Given a undirected graph $G = (V, E)$ and a non-negative integer k, a connected k-core of G is a connected component $\hat{H}_k \subseteq H_k \subseteq G$, such that $\forall v \in \hat{H}_k$, $deg(v, \hat{H}_k) \geq k$.*

Generally, a H_k usually contains multiple $\hat{H}_k$, we add a superscript to denote a specific one as $\hat{H}_k^i$ if necessary. We are interested in the following community search (CS) problem.

Problem 1 (CS problem [12]). *Given a undirected graph $G = (V, E)$, a query node $q \in V$, and a non-negative integer k, return the nodes in the connected k-core containing q as the community of q.*

Example 1. *In Figure 1, k-cores with different $k \in [1, 3]$, i.e., $\{H_1, H_2, H_3\}$, are marked in different colors, where H_3 is the combination of two separate connected 3-cores $\hat{H}_3^1$ and $\hat{H}_3^2$. Given $q = v_5$ and $k = 3$, all nodes $\{v_1, v_2, v_5, v_6, v_7\}$ in $\hat{H}_3^1$ are returned as the desired community of q.*

3 CS with Graph Embeddings

3.1 Framework Overview

Figure 2 shows our CS framework with graph embeddings that consists of two stages: offline graph embedding and online community search. For offline stage, we aim to obtain the node representations by considering the community's cohesiveness features. Two key issues should be solved, that are (1) how to embed a k-core community's cohesiveness features w.r.t. a specific k into the learned node representations, i.e., community-injected graph embedding (§3.2) and (2) how to extend it to a general case where k would be an arbitrary value (§3.3). As a result, two nodes are similar if they belong to the same k-core community. It's worth mentioning that, we can extend to other community models, e.g., k-truss and k-clique, by simply replacing k-core with other models in §3.2 and §3.3. For the online phase, we build a proximity graph (PG) which is based on node representations, to return the most similar nodes to the query node as the desired community (§3.4).

3.2 Community-injected Graph Embedding

Given a graph $G = (V, E)$ and an input k, the basic idea of community-injected graph embedding is illustrated in Figure 2 (left top): First, we leverage a GNN model to capture the original topology features of G for getting the node representations $\mathcal{V} =$

$\{\vec{v}_1, \cdots, \vec{v}_{|V|}\}$, where $\vec{v}$ is a node v's representation. Second, we inject the community's cohesiveness features w.r.t. a specific k in terms of triplet samples to optimize GNN model's parameters via a triplet loss. We would like to shorten the distance of nodes from the same community and enlarge that of nodes from different communities.

GNN model. Many different GNN models are available to choose, such as GCN [29], GraphSAGE [22], and GAT [50], etc. Given a graph $G = (V, E)$, we use a GNN model to obtain the initial representation of each node $v \in V$, denoted by $\vec{v}$. We provide the general intra-layer processes of a GNN model as follows, where $u \in N(v)$ is a neighbor of v.

$$\vec{v}^{(l+1)} = \texttt{Dr}(\phi(\texttt{AGG}(\vec{u}^{(l)} W^{(l+1)} + b^{(l+1)}))) \quad (1)$$

Note that, $\vec{v}^{(l+1)} \in \mathbb{R}^{d^{(l+1)}}$ is the representation of v with $d^{(l+1)}$ dimensions in the $(l+1)$-th layer, $\vec{u}^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the representation of each $u \in N(v)$ from the l-th layer, and the input $\vec{v}^{(0)} \in \mathbb{R}^d$ is the one-hot representation of v. Besides, $W^{(l+1)} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ and $b^{(l+1)} \in \mathbb{R}^{d^{(l+1)}}$ are trainable weights and bias, respectively. Moreover, $\texttt{AGG}(\cdot)$ is an aggregation function such as AVG, SUM, MAX, or MIN, $\phi(\cdot)$ is the non-linear activation function, e.g., $\texttt{ReLU}(\cdot)$, and $\texttt{Dr}(\cdot)$ is the dropout method [45] to relieve overfitting in neural networks. The original graph topology features will be embedded into the obtained representation $\vec{v}^{(l+1)}$ after several epochs of training. As the beginning, in our implementation, we directly adopt the classical GCN model to explore the feasibility of our solution. In the future, we will pay more attention on the design of GNN models for CS.

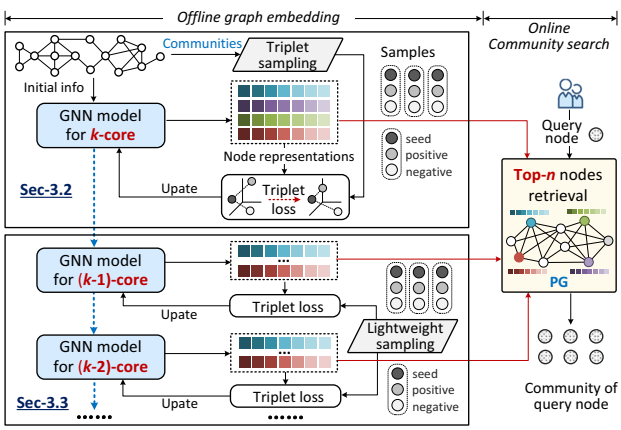


Figure 2. Framework of GNN-based community search

Community-oriented triplet sampling. Intuitively, two nodes are similar if they are from the same k-core community. So, we adopt the following triplet sampling with two steps to collect triplet samples as training data, to update the node representations from GNN model.

(1) *Core-decomposition.* Given a graph G and a cohesiveness parameter k, we apply a classic core-decomposition (CD) method to obtain the k-core H_k of G. We refer readers to [3] for details, but highlight the basic idea of CD here. First, we delete all nodes with $deg(\cdot) < k$ from G as they never be included in H_k. Since deleting a node v will affect v's neighbor's degree, we must check if each $u \in N(v)$ still satisfies the k-constraint after deleting v. If $deg(u) < k$, u is deleted too. We repeat this until all remained nodes have $deg(\cdot) \geq k$, thus returning H_k. Next, we conduct a breadth-first search on H_k to find all connected components $\{\hat{H}_k^1 \ldots \hat{H}_k^n\}$ of H_k. We refer each $\hat{H}_k^i$ as a real community and refer $G \setminus H_k$ as a fake community $\tilde{H}_k$. So, each node in G either belongs to a $\hat{H}_k^i$ or a $\tilde{H}_k$, and we can define a community-oriented triplet sample as follows.

(2) *Seed node and positive/negative sample.* We denote a community-oriented triplet sample by $\langle v_+, v_s, v_- \rangle$, where v_+ (v_-) is the positive (negative) sample w.r.t. a seed node v_s. We randomly select one node from a community H (H could be a $\hat{H}_k^i \in H_k$ or $\tilde{H}_k$) as v_s, then we get v_+ and v_- as follows.

Definition 3 (Positive sample)**.** *Given a community H containing a seed node v_s, a node in $H \setminus v_s$ is a positive sample v_+ w.r.t. v_s, i.e., v_+ is in the same community as v_s.*

Definition 4 (Negative sample)**.** *Given a community H containing a seed node v_s, a node in $G \setminus H$ is a negative sample v_- w.r.t. v_s, i.e., v_- is in a different community from v_s.*

Given a certain v_s from a specific H, we collect one positive sample v^+ from H and n negative samples $\{v_-^1 \ldots v_-^n\}$ from other n communities except H to form n triplet samples. We repeat it until sufficient triplet samples are collected. In §5, we show the effect of sample size on effectiveness.

Example 2. *Figure 3 illustrates 2 real communities $\hat{H}_3^1, \hat{H}_3^2$ and 1 fake community $\tilde{H}_3$ of the graph in Figure 1, for $k = 3$. If we randomly select 3 seed nodes from the 3 communities, then we have $3 \times 2 = 6$ triplet samples in total. For example, given $v_s = v_1$ from $\hat{H}_3^1$, we can collect $v_+ = v_7$ and $\{v_{10}, v_3\}$ as 2 negative samples from communities different from $\hat{H}_3^1$.*

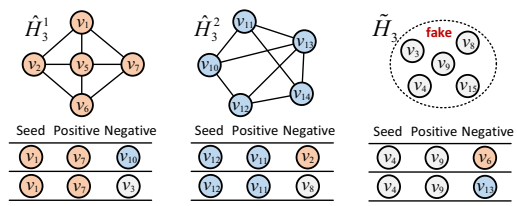


Figure 3. Community-oriented triplet sampling

Training with triplet loss. Given a set of triplet samples as the training data, we use a margin-based triplet loss to update the parameters Θ_G of GNN model. The basic idea is to make nodes from the same community $\langle v_s, v_+ \rangle$ closer in the vector space and nodes from different communities $\langle v_s, v_- \rangle$ as far away as possible. So, we minimize the triplet loss in Eq. 2:

$$\mathcal{L}(\Theta_G) = \max\{\delta(\vec{v}_s, \vec{v}_+) - \delta(\vec{v}_s, \vec{v}_-) + c, 0\}\,, \quad (2)$$

where $\vec{v}_s$, $\vec{v}_+$, $\vec{v}_-$ are the representations of seed node, positive, and negative sample, respectively. The margin hyperparameter c (we empirically choose $c = 1$) is used to ensure that v_+ is closer to v_s than v_- in terms of cosine distance δ.

3.3 Scalability on Various Cohesiveness

We now extend the above method for various cohesiveness parameter k. First, we introduce an important *inclusive hierarchy* property of k-core. Then, we take the GNN model for k-core as a pre-trained model for (k-1)-core, and design a lightweight triplet sampling method based on the inclusive hierarchy property to fine-tune the model with little overhead.

Proposition 1. *Given a graph G and an integer $k > 0$, for $\forall \hat{H}_k \neq \emptyset$, there exists a $\hat{H}_{k-1}$ such that $\hat{H}_k \subseteq \hat{H}_{k-1}$.*

Proposition 2. *Given a graph G and a node $v \in G$ with the degree as $deg(v, G) < k$, we have $v \notin \hat{H}_k$.*

Above two propositions directly follow the observation that each node $v \in \hat{H}_k$ has $k \leq deg(v, \hat{H}_k) \leq deg(v, G)$.

Theorem 1. *Given a node $v \in \hat{H}_k$, we have: (1) $v \in \hat{H}_{k-1}$, (2) for $\forall v \in \hat{H}_{k-1} \setminus \hat{H}_k$, v must belong to the fake community $\tilde{H}_k$, i.e., $v \in \tilde{H}_k$, and (3) v has a degree of $deg(v, G) \geq k - 1$.*

Proof. For (1), it holds according to Proposition 1. For (2), suppose $\exists v \in \hat{H}_{k-1} \setminus \hat{H}_k$ that comes from another connected k-core $\hat{H}'_k$, then $\hat{H}_k$, $\hat{H}'_k$ must be connected, which is contradictory to the fact that they are two separate connected k-cores. Thus, for $\forall v \in \hat{H}_{k-1} \setminus \hat{H}_k$, we have $v \in \tilde{H}_k$. For (3), we have $deg(v, G) \geq k - 1$ as $v \in \hat{H}_{k-1}$ (Proposition 2). □

Theorem 2. *Given a connected (k-1)-core $\hat{H}_{k-1}$ that $\hat{H}_{k-1} \cap H_k = \emptyset$, then we have $\hat{H}_{k-1} \subseteq \tilde{H}_k$.*

Proof. The theorem directly follows the observation. □

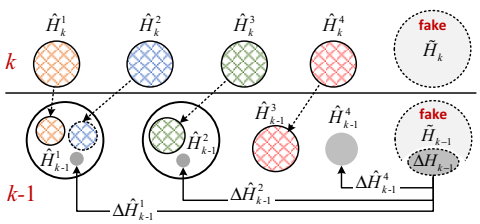


Figure 4. Community evolution from k to $k-1$

We summarize four cases of the community evolution from k to $k-1$ in Figure 4, based on Theorem 1 (case 1-3) and Theorem 2 (case 4). (1) $\hat{H}_{k-1} = \hat{H}_k$ (red circle), (2) $\hat{H}_{k-1}$ contains one $\hat{H}_k$ (green circle), (3) $\hat{H}_{k-1}$ contains more than one $\hat{H}_k$ (orange and blue circles), i.e., $\bigcup_{i=1}^{m} \hat{H}_k^i \subset \hat{H}_{k-1}$, and (4) $\hat{H}_{k-1} \subseteq \tilde{H}_k$ if $\hat{H}_{k-1} \cap H_k = \emptyset$ (dark grey circle). For simplicity we use $\Delta\hat{H}_{k-1}$ to denote the incremental part ($\hat{H}_{k-1} \setminus \hat{H}_k$) of each $\hat{H}_{k-1}$ and $\forall v \in \Delta\hat{H}_{k-1}$ has $deg(v, G) \geq k - 1$ (Theorem 1 (3)). We next target on $\Delta\hat{H}_{k-1}$ to design a lightweight triplet sampling as follows.

(1) *Compute $\Delta\hat{H}_{k-1}$.* We extract all nodes with $deg(v, G) \geq k - 1$ from $\tilde{H}_k$ as candidates ΔV, then we run CD on the graph of $H_k \cup \Delta V$ to find all connected (k-1)-cores $\{\hat{H}_{k-1}^1 \dots \hat{H}_{k-1}^n\}$, thus getting each $\Delta\hat{H}_{k-1}^i$.

(2) *Triplet sampling w.r.t. fake community.* We use $\Delta H_{k-1} = \{\Delta\hat{H}_{k-1}^1 \dots \Delta\hat{H}_{k-1}^n\}$ to denote the incremental part of H_{k-1}. So, the fake community $\tilde{H}_k$ is divided into two disjoint parts $\tilde{H}_{k-1}$ and ΔH_{k-1}. We expect to peel ΔH_{k-1} from $\tilde{H}_k$. To be more precise, we expect ΔH_{k-1} to be as far away from $\tilde{H}_{k-1}$ as possible. Thus, we collect a triplet sample as follows. First, we randomly select two nodes from $\tilde{H}_{k-1}$ as the seed node and positive sample. Then, we randomly select one node from ΔH_{k-1} as the negative sample.

(3) *Triplet sampling w.r.t. real communities.* Since $\Delta\hat{H}_{k-1}^i \subseteq \hat{H}_{k-1}^i$, we need to make nodes from $\Delta\hat{H}_{k-1}^i$ are closer to that from $\hat{H}_{k-1}^i$ and as far away from nodes of another $\hat{H}_{k-1}^j$ ($i \neq j$) as possible. Thus, we collect a triplet sample as follows. First, we randomly select one node from $\Delta\hat{H}_{k-1}^i$ as a seed node. Then, we randomly select one node from $\hat{H}_{k-1}^i$ as a positive sample. Finally, we randomly select one node from $\forall\hat{H}_{k-1}^j$ with $i \neq j$ as a negative sample.

Figure 2 (left bottom) shows that, we fine-tune the inherited GNN model for (k-1)-core by using the above triplet samples and the same triplet loss as Eq. 2, thus obtaining the node representations w.r.t. the cohesiveness parameter of $k-1$.

3.4 Online Community Search with PG

Given a graph $G = (V, E)$ with the node representations $\mathcal{V} = \{\vec{v}_1, \cdots, \vec{v}_{|V|}\}$ for a specific k and a query node q, we expect to return the most similar nodes to q from V as the community of q, by measuring the similarity between $\vec{q}$ and $\vec{v} \in \mathcal{V}$. It is reasonable as we already embed the community's cohesiveness features into $\mathcal{V}$, thus two nodes are likely to be similar if they are from the same community (see case study in §5). We build a proximity graph (PG) [35, 32] for all nodes in V based on $\mathcal{V}$, and then quickly return the top-n similar nodes from PG as the result.

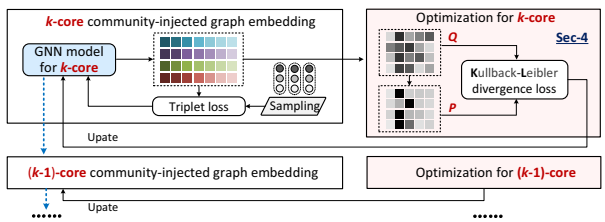


Figure 5. Optimization with KL divergence

(1) *PG construction.* We construct PG in a *insert manner* based on the node similarity (e.g., cosine similarity). For $\forall v \in V$, it has a corresponding node v^* in PG with a node representation $\vec{v}^* = \vec{v}$. Given an empty PG, we start inserting from a randomly selected $v \in V$: First, we create a node v^* for v in PG. Second, we take v^* as a query node to find its top-r nearest nodes from the current PG and add edges between the r nodes and v^*. Here, r is a pre-defined upper bound of the # neighbors in PG. We show r's effect in §5. We repeat this until all nodes in V have been inserted to PG.

(2) *Top-n nodes retrieval from PG.* Given a constructed PG and a query node q, we start a greedy top-n nodes retrieval from a randomly selected entry node s. First, we maintain a min-heap `Cand` to record all candidate nodes for search expansion and a max-heap `Result` to record the top-n nearest nodes to q; both are initialized by $\langle s, \delta(\vec{s}, \vec{q})\rangle$. Next, we obtain a node $v^*_{\max} \in$ `Cand` with the largest similarity to q and a node $v^*_{\min} \in$ `Result` with the smallest similarity to q. If $\delta(\vec{q}, \vec{v}^*_{\max}) < \delta(\vec{q}, \vec{v}^*_{\min})$, we stop the search and return `Result`, as no candidate node is more similar to q than the worst node in `Result`. Otherwise, we expand the search from $v^*_{\max}$ as it is the nearest node to q in `Cand`. We add $v^*_{\max}$'s neighbors to `Cand` and `Result`, and remove the worst nodes from `Result`, if $|$`Result`$| > n$. We repeat above until `Cand` is empty.

4 Optimization with KL Divergence

Ideally, two nodes from the same community would show a higher similarity than others. However, this trend may not be obvious due to the existence of difficult negative samples, e.g., nodes that are adjacent to the seed node v_s but belong to different communities of v_s. To confront this challenge, we develop a self-augmented method based

on KL(Kullback-Leible) divergence to further optimize the quality of node representations.

Intuition. Given a node $v_i \in \hat{H}_k^j$. If we can obtain the probability of v_i belonging to $\hat{H}_k^j$ (i.e., q_{ij}) based on the representation $\vec{v}_i$, then we can optimize $\vec{v}_i$ by emphasizing the value of q_{ij} [39, 51, 4]. Figure 5 illustrates our optimization with three steps.

(1) *Compute the assignment distribution Q.* We leverage the Student's t-distribution [49] to measure the similarity between $\vec{v}_i$ and $\vec{\rho}_j$, where ρ_j is the centroid of $\hat{H}_k^j$, thus getting q_{ij} as follows:

$$q_{ij} = \frac{(1+\delta(\vec{v}_i,\vec{\rho}_j)/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1+\delta(\vec{v}_i,\vec{\rho}_{j'})/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{3}$$

where $\vec{\rho}_j$ is initialized as the averaged vector of all node representations from $\hat{H}_k^j$ and α is a configurable parameter which is set as 1 for all experiments. We consider each q_{ij} as the probability of assigning a node v_i to a community $\hat{H}_k^j$, i.e., a soft assignment, so that the matrix $Q = [q_{ij}]$ is viewed as the distribution of all nodes' assignments.

(2) *Compute the target distribution P.* Soft assignments with high probability (nodes close to the centroid) are considered to be trustworthy in Q, so we emphasize such high confident assignments as much as possible to make their node representations closer to the centroid. Thus, we calculate a target distribution $P = [p_{ij}]$ as:

$$p_{ij} = \frac{q_{ij}^2/\sum_i q_{ij}}{\sum_{j'}(q_{ij'}^2/\sum_i q_{ij'})}, \tag{4}$$

where p_{ij} is the target (or self-augmented) assignment and $\sum_i q_{ij}$ is the soft assignment frequency of $\hat{H}_k^j$. Each $q_{ij} \in Q$ is squared and normalized so that the high confident $q_{ij} \in Q$ will have even higher confidence in the target distribution P.

(3) *Training with KL divergence loss.* Given Q and P, its nature to optimize the node representations by making Q to approach P. So, we have the following KL divergence loss.

$$\mathcal{L}(\Theta_G) = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5}$$

By minimizing Eq. 5, P helps learn a better representations that ensure the nodes surround the community's centroid are closer than others. We call it a self-augmented method as P is computed from Q and used to supervise the updating of Q.

Table 1. Dataset Statistics.

Datasets	# Nodes	# Edges	d_{avg}	$k_{\max}$
Ca-AstroPh	18772	198110	21.1	56
Citeseer	3312	4732	2.7	7
Cora	2708	5429	3.9	4
Deezer	28281	92752	6.6	12
Facebook PPN	22470	171002	15.2	56
PPI	56944	818716	243.3	153
Pubmed	19717	44338	4.5	10

5 Experiments

We compared our solution with several deep learning methods as well as traditional database methods from both the perspectives of **query effectiveness and efficiency** on seven real-world datasets, explored the **parameter sensitivity**, provided **ablation analysis**, gave the **training time consumption**, analyzed **model size**, and showed a **case study**. Our code [1] were implemented in Python3.8, and all experiments were conducted on 20 cores of a 3.7GHZ server (running Ubuntu Linux) with 2 NVIDIA GeForce RTX 3090 (24G memory).

5.1 Setup

Datasets. Table 1 shows seven real-world networks' statistics, d_{avg} and $k_{\max}$ are the average degree and largest k of k-core. Ca-AstroPh [31], Cora [43], Citeseer [40], and Pubmed [37] are citation networks. Deezer [42] is a social network. PPI [46] is a protein-protein graph, and Facebook PPN [41] is a graph of Facebook sites.

Comparing methods. We compared representative CS methods in both AI ((1)-(2)) and DB ((3)-(6)) areas with our solution ((7)-(8)). (1) **Simple-QD-GNN** [28] is the first supervised end-to-end CS solution based on GCN. (2) **QD-GNN** is the extend version of (1). (3) **FastBcore** [20] proposes a CS method for finding $(k, \mathcal{P})$-core communities. (4) **S-Greedy** [2] is a parameter-free CS method for multiple query nodes. (5) **LEKS-path** [47] presents a WC-index-based CS method to find the intimate-core community. We implemented two versions of our solution: (6) **Ours-Triplet** is the implementation of §3 and (7) **Ours-Triplet+KL** is the one with the KL optimization (§4). For ours, we used the same PG-based online search, and the difference only comes from the offline learning stage.

Evaluation metric. We used F1-score (Eq. 6-8) and average response time to measure the effectiveness and efficiency. We let $D_{\text{test}} = \{D_{\text{test}}^k, D_{\text{test}}^{k-1}, \dots\}$ be test data, $D_{\text{test}}^k = \{Q^k, C^k, R^k\}$ is the sub-test data for a specific k. $Q^k = \{q_1, \dots, q_n\}$ is the query node set, $C^k = \{c_{q_1}^k, \dots, c_{q_n}^k\}$ records the communities of q_i returned by our method, and $R^k = \{r_{q_1}^k, \dots, r_{q_n}^k\}$ is the k-core communities of q_i.

$$F1^k(C^k, R^k) = \frac{2 \cdot Pre^k(C^k, R^k) \cdot Rec^k(C^k, R^k)}{Pre^k(C^k, R^k) + Rec^k(C^k, R^k)} \tag{6}$$

$$Pre^k(C^k, R^k) = \frac{\sum_{i=1}^n |c_{q_i}^k \cap r_{q_i}^k|}{\sum_{i=1}^n |c_{q_i}^k|} \tag{7}$$

$$Rec^k(\hat{C}^k, R^k) = \frac{\sum_{i=1}^n |c_{q_i}^k \cap r_{q_i}^k|}{\sum_{i=1}^n |r_{q_i}^k|} \tag{8}$$

$Pre(C^k, R^k)$ is the precision of returned communities and $Rec(C^k, R^k)$ is the recall of returned communities. c_q^k and r_q^k are the predicted and k-core community for query q which is from the query set Q^k respectively.

Parameters. We tried all possible k in a dataset from $k_{\max}$ to 2 for offline graph embedding. We set the epoch for training with triplet loss (§3) and KL divergence loss (§4) as 20 and 10, respectively. For learning rate, we used a combined strategy of warm-up followed by cosine annealing, and we set the batch-size to 2048. Besides, we built PG with $r = 9$.

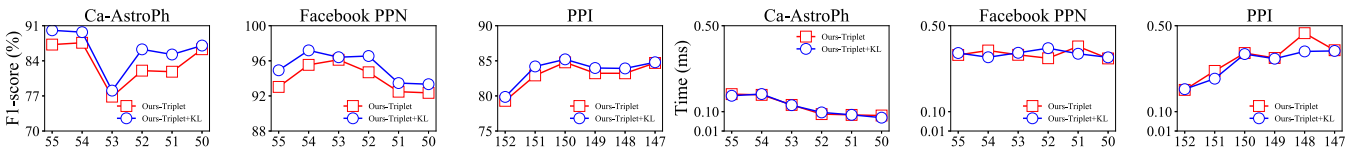
5.2 Experiment Results

Effectiveness and Efficiency. Experimental results over all datasets are summarized in Table 2. We emphasize the best performance of our solutions with bold values. Moreover, "-" indicates that

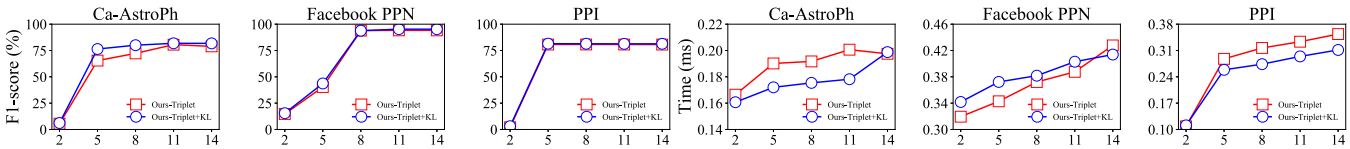
[1] https://anonymous.4open.science/r/EECSGNN-2AC4/

Table 2. Effectiveness (left, F1-score in %) and efficiency results (right, query time in millisecond) on all datasets

Methods	Ca-AstroPh		Citeseer		Cora		Deezer		Facebook PPN		PPI		Pubmed	
Ours-Triplet	83.65	0.125	88.21	0.331	93.04	**0.083**	81.99	0.339	94.04	**0.369**	83.03	0.344	78.12	0.130
Ours-Triplet+KL	**86.08**	**0.123**	**89.96**	**0.330**	**93.33**	0.085	**83.48**	**0.211**	**95.31**	0.370	**83.66**	**0.323**	**83.38**	**0.128**
Simple-QD-GNN	40.47	5.490	28.50	0.819	54.02	0.546	0.82	14.48	32.16	7.820	0.685	46.061	17.13	5.989
QD-GNN	42.80	26.052	28.10	2.305	44.09	1.692	6.04	65.119	38.95	37.378	-	-	15.54	28.650
FastBcore	100.0	80.43	100.0	3.20	100.0	1.96	100.0	62.56	100.0	78.22	100.0	2168.67	100.0	28.17
S-Greedy	100.0	34.900	100.0	0.092	100.0	2.228	100.0	94.783	100.0	5.183	100.0	51.857	100.0	94.629
LEKS-path	100.0	70.030	100.0	32.647	100.0	24.647	100.0	95.293	100.0	138.267	100.0	27079.59	100.0	113.047



(a) Effect of k on CS's effectiveness (left three figures) and efficiency (right three figures)



(b) Effect of r on CS's effectiveness (left three figures) and efficiency (right three figures)

Figure 6. Parameter sensitivity: effect of k and r on effectiveness and efficiency on Ca-AstroPh, Facebook PPN, and PPI

Table 3. Ablation analysis on all datasets.

Datasets	T w/o FT	T w/o FT+KL	T	T+KL
Ca-AstroPh	83.65	86.08	76.18	77.76
Citeseer	88.21	89.96	80.65	83.98
Cora	93.04	93.33	83.56	85.54
Deezer	81.99	83.48	71.51	73.46
Facebook PPN	94.04	95.31	82.64	90.62
PPI	83.03	83.66	81.28	81.95
Pubmed	78.21	83.38	76.69	77.98

a solution hardly implements in a dataset. In a nutshell, ours achieve a good trade-off between effectiveness and efficiency. Specifically, compared with AI solutions, ours significantly improve the F1-score by 61.01% on average for all datasets (at least 39.3% for Cora dataset). Our F1-socre of 87.89% on average still have room for improvement compared with the exact DB-based solutions, however, in terms of efficiency, ours is up to one to two orders of magnitude faster than others. The larger the dataset is, the more the efficiency improvement we have. For example, we only require 0.323 ms to response a query for the largest PPI, which is 142 X faster than AI solution and at least 160 X faster than DB solutions.

Parameter sensitivity. We explored the effect of cohesiveness parameter k, r neighbors of a PG, and sample size on the performance of Ours-Triplet and Ours-Triplet+KL. Due to the page limit, we provide the results on three datasets.

(1) *Effect of k.* As shown in Figure 6(a), the CS's effectiveness and efficiency are stable w.r.t. different k, which means that our solution is scalable to various cohesiveness parameter k. To be more precise, for any query node with an arbitrary k, we can quickly return a community with an acceptable quality.

(2) *Effect of r.* We varied r from 2 to 14 to build different PGs to evaluate the effect of r on CS's effectiveness and efficiency. Figure 6(b) shows that the efficiency decreases as r increases, because in the greedy search over a PG, we have to measure the distance from each of r neighbors to the query node, so the larger the r, the more time is required. Moreover, it's nature to see that the effectiveness increases as r increases. A trade-off can be achieved when we set $r = 9$.

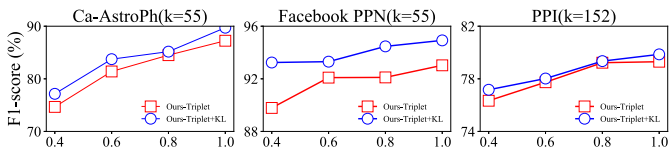


Figure 7. Effect of sample size on CS's effectiveness

(3) *Effect of sample size.* The sample size is critical to the graph embedding's quality, thus affecting the CS's effectiveness. We kept other parameters unchanged and varied the sample size to see its effect(value from 0 to 1 indicates the sample size). From Figure 7, we found that F1-score is positively correlated to the sample size.

Ablation analysis. To verify the effectiveness of the fine-tune strategy with lightweight sampling in §3.3 and the KL optimization in §4, we implemented additional two versions of our solution: (1) Ours-Triplet w/o FT (shorten as T w/o FT) is the one without fine-tune strategy and (2) Ours-Triplet w/o FT+KL (T w/o FT+KL) is the one with KL optimization. It's worth mentioning that Ours-Triplet (T) is the one with the fine-tune strategy and Ours-Triplet+KL (T+KL) is the KL optimized version. We set the same configurations and show the results for $k_{\max} - 1$ of each dataset in Table 3. Ours-Triplet outperforms Ours-Triplet w/o FT with 7.09% improvement on average, showing that the fine-tune strategy is effective. Besides, the KL optimized versions perform better than that w/o KL, proving that the KL optimization is effective. For example, there is a 2.68% improvement on average between Ours-Triplet w/o FT+KL and Ours-Triplet w/o FT and 1.86% improvement between Ours-Triplet+KL and Ours-Triplet.

Table 4. Training time consumption (left, time in second) and model size(right, parameter size in MByte) on all datasets.

Datasets	Triplets training		Finetuning		Simple QD-GNN		QD-GNN	
Ca-AstroPh	988.6	1.2	497.3	1.2	4269.5	7.1	26211.2	15.0
Citeseer	705.2	0.2	495.3	0.2	249.3	1.3	629.4	2.5
Cora	331.0	0.2	166.2	0.2	293.3	1.1	581.4	2.1
Deezer	2887.1	1.8	1433.9	1.8	3390.5	11.0	26253.3	22.0
Facebook PPN	158.4	1.4	81.1	1.4	9071.1	8.5	39987.7	17.0
PPI	2243.7	2.8	88.21	2.8	35149.4	17.0	-	-
Pubmed	2943.5	1.3	945.2	1.3	6683.5	7.4	27568.9	15.0

Training time consumption. We conducted experiments to measure the average training time consumed for all datasets, as presented in Table 4(the left column of each method). Training time predominantly comprises of two key components: (1) the time consumption required for triplets training, (2) the time consumption required for fine-tuning using KL optimization, (3) Simple QD-GNN's training time consumption, and(4) QD-GNN's training time consumption. By calculating the average training time, we find that Simple QD-GNN and QD-GNN time consumption is 2X and 9X slower than Triplets training, and is 12X and 30X slower than fine-tuning.

Model size. From the perspective of the saved model size(the right column of each method), our model has significantly fewer parameters compared to QD-GNN, the average parameters of our model are 1/5 of simple QD-GNN and 1/10 of QD-GNN), which is also a reason why our training is faster than QD-GNN.

Case study. A case study on the largest PPI dataset for $k = 7$ is illustrated in Figure 8, where the node representations are visualized w.r.t. vector similarity. Here, we only provide 30% of nodes from PPI due to the page limit. We select three query nodes with green (q_1), red (q_2), and blue (q_3) stars, and provide the k-core communities of each q with corresponding colors. Note that, most of the k-core members are similar to the query node q and only a small number of nodes (with colored circle) are scattered far away from q. For example, there are 76%, 80%, and 90% of k-core members are distributed around q_1, q_2, and q_3, respectively, showing that our node representations can well reflect the original community involvement, thus benefiting the CS' effectiveness.

6 Related Works

CS in DB area. According to the graph types, it has two subcategories. (1) *CS on homogeneous graphs*. Many works focus on modeling the cohesive community based on k-core [3, 5, 19, 2], k-truss [1, 24, 26, 33], k-clique [11, 48, 55], and k-ECC [23]. However, they ignore the CS on attributed graphs. Thus, many works define different metrics of attribute cohesiveness and integrate it with the structure cohesiveness for CS [8, 9, 16, 15, 17, 25, 34, 56]. (2) *CS on heterogeneous graphs*. The meta-path $\mathcal{P}$ is often used to indicate the relation between two node types, based on which some community models are proposed, e.g., $(k, \mathcal{P})$-core [20], $(k, \mathcal{P})$-Btruss, and $(k, \mathcal{P})$-Ctruss [53]. No matter which model is used, the essence is to find the cohesive subgraph via graph traversal, which is time-consuming (e.g., from hundreds of milliseconds to tens of seconds to respond [47, 10, 54]). This motivates the CS in AI area.

CS in AI area. Recently, [28, 27] design a supervised end-to-end model based on GCN to map an input query node (represented as a d-dimensional vector) into a set of nodes (represented as a community vector). [14] applies meta-learning algorithms to CS, and solves problems with the small data. In these solutions, datasets with ground-truths (i.e., human-annotated communities) are required as the input supervised information. However, the ground-truths are usually unavailable in real-life applications and the effectiveness still have a large room for improvement. This inspires us to design our CS solution with graph embeddings, which achieves a good balance between effectiveness and efficiency.

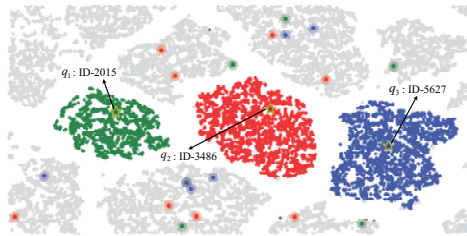


Figure 8. Case study on the PPI dataset

7 Conclusion

In this paper, we first present an offline community-injected graph embedding method to preserve the community's cohesiveness features into node representations, through community-oriented triplet sampling. Second, we resort to a proximity graph (PG) built from the node representations, to quickly return the community online. Finally, we develop a self-augmented method based on KL divergence to optimize the node representations. We conduct experimental study on seven real-world graphs and results show that our solution can achieve a balance between effectiveness and efficiency. In the future, we will investigate how to extend our solution to other community models, such as k-truss and k-clique, over different graph types, e.g., weighted graphs and attribute graphs.

8 Acknowledgments

This work was supported by the National NSF of China (62072149 and 62006040), the Primary R&D Plan of Zhejiang (2021C03156 and 2023C03198), and the Fundamental Research Funds for the Provincial Universities of Zhejiang (GK219909299001-006).

References

[1] Esra Akbas and Peixiang Zhao, 'Truss-based community search: a truss-equivalence based indexing approach', *PVLDB*, **10**(11), 1298–1309, (2017).
[2] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo, 'Efficient and effective community search', *Data Min. Knowl. Discov.*, **29**(5), 1406–1433, (2015).
[3] Vladimir Batagelj and Matjaz Zaversnik, 'An o (m) algorithm for cores decomposition of networks', *arXiv*, **cs.DS/0310049**, (2003).
[4] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui, 'Structural deep clustering network', in *WWW*, pp. 1400–1410, (2020).
[5] Francesco Bonchi, Arijit Khan, and Lorenzo Severini, 'Distance-generalized core decomposition', in *SIGMOD*, pp. 1006–1023, (2019).
[6] Lijun Chang and Lu Qin, 'Cohesive subgraph computation over large sparse graphs', in *ICDE*, pp. 2068–2071, (2019).
[7] Huajun Chen, Ning Hu, Guilin Qi, Haofen Wang, Zhen Bi, Jie Li, and Fan Yang, 'Openkg chain: A blockchain infrastructure for open knowledge graphs', *Data Intell.*, **3**(2), 205–227, (2021).
[8] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang, 'Maximum co-located community search in large scale social networks', *PVLDB*, **11**(10), 1233–1246, (2018).

[9] Yankai Chen, Yixiang Fang, Reynold Cheng, Yun Li, Xiaojun Chen, and Jie Zhang, 'Exploring communities in large profiled graphs', *TKDE*, **31**(8), 1624–1629, (2019).

[10] Yankai Chen, Jie Zhang, Yixiang Fang, Xin Cao, and Irwin King, 'Efficient community search over large directed graph: An augmented index-based approach', in *IJCAI*, pp. 3544–3550, (2020).

[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang, 'Online Search of Overlapping Communities', in *SIGMOD*, pp. 277–288, (2013).

[12] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang, 'Local Search of Communities in Large Graphs', in *SIGMOD*, pp. 991–1002, (2014).

[13] Joel Dudley, Tarangini Deshpande, and Atul J. Butte, 'Exploiting drug-disease relationships for computational drug repositioning', *Briefings Bioinform*, **12**(4), 303–311, (2011).

[14] Shuheng Fang, Kangfei Zhao, Guanghua Li, and Jeffery Xu Yu, 'Community search: A meta-learning approach', (2022).

[15] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu, 'Effective and efficient attributed community search', *VLDBJ*, **26**(6), 803–828, (2017).

[16] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu, 'Effective Community Search over Large Spatial Graphs', *PVLDB*, **10**(6), 709–720, (2017).

[17] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu, 'Effective community search for large attributed graphs', *PVLDB*, **9**(12), 1233–1244, (2016).

[18] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin, 'A survey of community search over big graphs', *VLDBJ*, **29**(1), 353–392, (2020).

[19] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu, 'Effective and efficient community search over large directed graphs', *IEEE Trans. Knowl. Data Eng.*, **31**(11), 2093–2107, (2019).

[20] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao, 'Effective and efficient community search over large heterogeneous information networks', *PVLDB*, **13**(6), 854–867, (2020).

[21] Christos Giatsidis, Fragkiskos D. Malliaros, Dimitrios M. Thilikos, and Michalis Vazirgiannis, 'Corecluster: A degeneracy based graph clustering framework', in *AAAI*, eds., Carla E. Brodley and Peter Stone, pp. 44–50, (2014).

[22] William L. Hamilton, Zhitao Ying, and Jure Leskovec, 'Inductive representation learning on large graphs', in *NeurIPS*, pp. 1024–1034, (2017).

[23] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang, 'Querying Minimal Steiner Maximum-connected Subgraphs in Large Graphs', in *CIKM*, pp. 1241–1250, (2016).

[24] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu, 'Querying k-truss community in large and dynamic graphs', in *SIGMOD*, pp. 1311–1322, (2014).

[25] Xin Huang and Laks V. S. Lakshmanan, 'Attribute-driven community search', *PVLDB*, **10**(9), 949–960, (2017).

[26] Xin Huang, Laks V. S. Lakshmanan, Jeffrey Xu Yu, and Hong Cheng, 'Approximate Closest Community Search in Networks', *PVLDB*, **9**(4), 276–287, (2015).

[27] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang, 'QD-GCN: query-driven graph convolutional networks for attributed community search', *arXiv*, **abs/2104.03583**, (2021).

[28] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang, 'Query driven-graph neural networks for community search: From non-attributed, attributed, to interactive attributed', *PVLDB*, **15**(6), 1243–1255, (2022).

[29] Thomas N. Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', in *ICLR*, (2017).

[30] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse, 'Identification of influential spreaders in complex networks', *Nature physics*, **6**(11), 888–893, (2010).

[31] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos, 'Graph evolution: Densification and shrinking diameters', *ACM Trans. Knowl. Discov. Data*, **1**(1), 2, (2007).

[32] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin, 'Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement', *IEEE Trans. Knowl. Data Eng.*, **32**(8), 1475–1488, (2020).

[33] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao, 'Truss-based community search over large directed graphs', in *SIGMOD*, pp. 2183–2197, (2020).

[34] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao, 'VAC: vertex-centric attributed community search', in *ICDE*, pp. 937–948, (2020).

[35] Yury A. Malkov and Dmitry A. Yashunin, 'Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs', *IEEE Trans. Pattern Anal. Mach. Intell.*, **42**(4), 824–836, (2020).

[36] Xiaoye Miao, Yue Liu, Lu Chen, Yunjun Gao, and Jianwei Yin, 'Reliable community search on uncertain graphs', in *ICDE*, pp. 1166–1179, (2022).

[37] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang, 'Query-driven active surveying for collective classification', in *International Workshop on Mining and Learning with Graphs (MLG)*, (2012).

[38] Jeff Z. Pan, Elspeth Edelstein, Patrik Bansky, and Adam Wyner, 'A knowledge graph based approach to social science surveys', *Data Intell.*, **3**(4), 477–506, (2021).

[39] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang, 'Adversarially regularized graph autoencoder for graph embedding', in *IJCAI*, pp. 2609–2615, (2018).

[40] Ryan A. Rossi and Nesreen K. Ahmed, 'The network data repository with interactive graph analytics and visualization', in *AAAI*, eds., Blai Bonet and Sven Koenig, pp. 4292–4293, (2015).

[41] Benedek Rozemberczki, Carl Allen, and Rik Sarkar, 'Multi-scale attributed node embedding', *J. Complex Networks*, **9**(2), (2021).

[42] Benedek Rozemberczki and Rik Sarkar, 'Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models', in *CIKM*, pp. 1325–1334, (2020).

[43] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad, 'Collective classification in network data', *AI Mag.*, **29**(3), 93–106, (2008).

[44] Mauro Sozio and Aristides Gionis, 'The community-search problem and how to plan a successful cocktail party', in *KDD*, pp. 939–948, (2010).

[45] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *J. Mach. Learn. Res.*, **15**(1), 1929–1958, (2014).

[46] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers, 'Biogrid: A general repository for interaction datasets', *Nucleic Acids Res.*, **34**(Database-Issue), 535–539, (2006).

[47] Longxu Sun, Xin Huang, Ronghua Li, Byron Choi, and Jianliang Xu, 'Index-based intimate-core community search in large weighted graphs', *IEEE Trans. Knowl. Data Eng.*, (2020).

[48] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli, 'Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees', in *KDD*, pp. 104–112, (2013).

[49] Laurens Van der Maaten and Geoffrey Hinton, 'Visualizing data using t-sne', *Journal of machine learning research*, **9**(11), (2008).

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, 'Graph attention networks', *arXiv*, **abs/1710.10903**, (2017).

[51] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang, 'Attributed graph clustering: A deep attentional embedding approach', in *IJCAI*, pp. 3670–3676, (2019).

[52] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang, 'A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search', *PVLDB*, **14**(11), 1964–1978, (2021).

[53] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang, 'Effective and Efficient Truss Computation over Large Heterogeneous Information Networks', in *ICDE*, pp. 901–912, (2020).

[54] Kai Yao and Lijun Chang, 'Efficient size-bounded community search over large networks', *PVLDB*, **14**(8), 1441–1453, (2021).

[55] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang, 'Index-based densest clique percolation community search in networks', *IEEE Trans. Knowl. Data Eng.*, **30**(5), 922–935, (2018).

[56] Zhiwei Zhang, Xin Huang, Jianliang Xu, Byron Choi, and Zechao Shang, 'Keyword-centric community search', in *ICDE*, pp. 422–433, (2019).