Community Search: A Meta-Learning Approach

Shuheng Fang*, Kangfei Zhao†, Guanghua Li‡, Jeffrey Xu Yu*

*The Chinese University of Hong Kong, †Beijing Institute of Technology
‡The Hong Kong University of Science and Technology (Guangzhou)
*{shfang,yu}@se.cuhk.edu.hk, †zkf1105@gmail.com, ‡gli945@connect.hkust-gz.edu.cn

2023 IEEE 39th International Conference on Data Engineering (ICDE) | 979-8-3503-2227-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDE55515.2023.00182

***Abstract*—Community Search (CS) is one of the fundamental graph analysis tasks, which is a building block of various real applications. Given any query nodes, CS aims to find cohesive subgraphs that query nodes belong to. Recently, a large number of CS algorithms are designed. These algorithms adopt pre-defined subgraph patterns to model the communities, which cannot find ground-truth communities that do not have such pre-defined patterns in real-world graphs. Thereby, machine learning (ML) and deep learning (DL) based approaches are proposed to capture flexible community structures by learning from ground-truth communities in a data-driven fashion. These approaches rely on sufficient training data to provide enough generalization for ML models, however, the ground-truth cannot be comprehensively collected beforehand.**

In this paper, we study ML/DL-based approaches for CS, under the circumstance of small training data. Instead of directly fitting the small data, we extract prior knowledge which is shared across multiple CS tasks via learning a meta model. Each CS task is a graph with several queries that possess corresponding partial ground-truth. The meta model can be swiftly adapted to a task to be predicted by feeding a few task-specific training data. We find that trivially applying multiple classical meta-learning algorithms to CS suffers from problems regarding prediction effectiveness, generalization capability and efficiency. To address such problems, we propose a novel meta-learning based framework, Conditional Graph Neural Process (CGNP), to fulfill the prior extraction and adaptation procedure. A meta CGNP model is a task-common node embedding function for clustering, learned by metric-based graph learning, which fully exploits the characteristics of CS. We compare CGNP with CS algorithms and ML baselines on real graphs with ground-truth communities. Our experiments verify that CGNP outperforms the other native graph algorithms and ML/DL baselines 0.33 and 0.26 on `F1` score by average.

***Index Terms*—Community search, Meta-learning, Neural process**

I. Introduction

Community is a cohesive subgraph that is densely intra-connected and loosely inter-connected in a graph. Given any query nodes, community search (CS) aims at finding communities covering the query nodes, i.e., local query-dependent communities, which has a wide range of real applications, e.g., friend recommendation, advertisement in e-commence and protein complex identification [1], [2]. In the literature, to model structural cohesiveness, various community models are adopted, including k-core [3]–[5], k-truss [6], [7], k-clique [8], [9] and k-edge connected component [10], [11]. Such models can be computed efficiently by CS algorithms. But such models are designed based on some pre-defined community patterns which are too rigid to be used to find ground-truth communities in real applications. We show a DBLP example in Fig. 1 in which nodes represent researchers and edges represent their collaboration. The ground-truth community of 'Jure Leskovec', i.e., the orange and white nodes in Fig. 1 are with the researchers who have collaborations and share the common interest of 'social networks'. Such a community cannot be accurately found with any k-related subgraph patterns. For example, in the community, some nodes (e.g., Michael W. M.) have one neighbor, which can only be found by 1-core that may result in accommodating the whole graph.

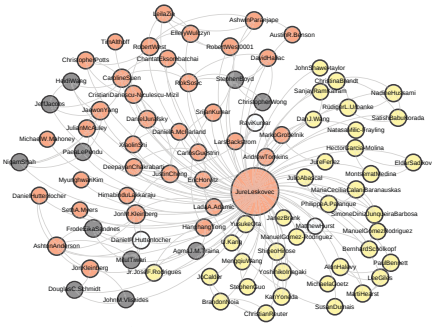


Fig. 1: An Example on DBLP: Query 'Jure Leskovec'

To tackle the structural inflexibility of CS algorithms, ML/DL-based solutions [12], [13] are arising as an attractive research direction. They build ML/DL models from given ground-truth communities and expect the models to generalize to unknown community-member relationships. Such ML/DL-based approaches have achieved success in finding high-quality solutions due to two reasons. For one thing, these data-driven approaches get rid of the inflexible constraints and adapt to implicit structural patterns from data. For another thing, the models can learn via error feedback from its predictions on the query nodes in the ground-truth communities. But, effective error feedback heavily relies on sufficient ground-truth communities to train, which are hard to collect and label. On the one hand, they are labor-intensive, on the other hand, such ground-truth communities for different query nodes can be very different.

To deal with this problem, an effective solution is to inject prior knowledge extracted from multiple CS tasks into the ML model, where one CS task is a subgraph with a small number of query nodes with partial ground-truth community membership. The implicit prior knowledge of the CS tasks is rather intuitive, i.e., for any query node of an arbitrary graph, its communities are the nearby densely connected nodes that

† Corresponding author.

2375-026X/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDE55515.2023.00182

share similar attributes with the query node. Such prior is shared by different CS tasks for different query nodes in any real-world graphs. And the prior knowledge is capable of synthesizing similar or complementary inductive bias across different CS tasks to compensate the insufficient knowledge from small training data, thus can be swiftly adapted to a new task to test. In this paper, we concentrate ourselves on learning a meta model to capture this prior by meta-learning.

There are existing meta-learning algorithms, e.g., simple feature transfer and model-agnostic meta-learning. However, trivial adaptations to CS tasks fail to achieve high performance since they do not exploit the intrinsic characteristic of the CS tasks. For CS, what a model needs to justify for each node in a graph is whether or not it has its community membership with any given query node. To facilitate such binary justification, we propose a novel model, Conditional Graph Neural Process (CGNP), to generate node embeddings conditioned on the small training data, where the distance between a node embedding to that of the query node explicitly indicates their community membership. Furthermore, as a graph specification of Conditional Neural Process (CNP) [14], CGNP inherits the main ideas of CNP that implicitly learns a kernel function between a training query node and a query node to be predicted. In a nutshell, the learned CGNP is not only a *common embedding function* but also a *common kernel function*, shared across different graphs. The embedding function transforms the nodes of each graph into a distance-aware hidden space, while the kernel function memorizes the small training data of each task as a hidden representation. Compared with optimization-based meta-learning approaches whose parameters are easy to overfit, the metric learning and memorization mechanisms are more suitable for classification tasks with small training data, especially for imbalanced labels.

The contributions of this paper are summarized as follows: ① We formulate the problem of learning a meta model to answer CS queries, where the meta model is to absorb the prior knowledge from multiple CS tasks and adapt to a specific task with only a few training data. We generalize three CS task scenarios that represent comprehensive query cases. To the best of our knowledge, our study is the first attempt at meta model/algorithm for CS. ② We explore three Graph Neural Network based solutions, i.e., feature transfer, model-agnostic meta-learning and Graph Prototypical Network, which are trivial adaptations of existing transfer/meta-learning algorithms to CS. We identify their individual limitations regarding prediction effectiveness, generalization capability and efficiency. ③ We propose a novel framework, *Conditional Graph Neural Process* (CGNP) on the basis of conceptual CNP and learn the meta model in an efficient, metric-based learning perspective. We design and explore model variants with different model complexities and different options for the core components. To the best of our knowledge, we made the first effort to explore how to solve CS problem by meta-learning. ④ We conduct extensive experiments on 6 real-world datasets with ground-truth communities for performance evaluation. Compared with 3 CS algorithms, 4 naive approaches, and 3 supervised learning validates our CGNP outperforms the others with small training and prediction cost.

Roadmap: The rest of the paper is organized as follows. section II reviews the relative work. In section III, we give the problem statement followed by three naive solutions introduced in section IV. We introduce the core idea of our approach, CGNP in section V. We elaborate on its architecture design and present the learning algorithms of CGNP in section VI. We present our comprehensive experimental studies in section VII and conclude the paper in section VIII.

II. Related Work

Community Search. A comprehensive survey of CS problems and approaches can be found in [1], [2]. In a nutshell, CS problem can be divided into two categories. One is non-attributed community search which only concerns the structural cohesiveness over simple graphs and the other is attributed community search (ACS) which concerns both the structural cohesiveness and content overlapping or similarities over attributed graphs. Regarding capturing the structural cohesiveness, various community metrics have been proposed, including k-core [3]–[5], k-truss [6], [7], k-clique [8], [9] and k-edge connected component [10], [11]. These metrics are inflexible to adapt to complex real-world graphs and applications.

In addition to only exploiting the structural information, ACS leverages both the structural constraint and attributes such as keywords [15], [16], location [17], temporal [18], etc. As two representative approaches for ACS, ATC [16] finds k-truss community with the maximum pre-defined attribute score. And ACQ [15] finds k-core communities whose nodes share the maximum attributes with the query attributes. Both ATC and ACQ adopt a two-stage process. First, they find the candidate communities based on the structural constraints. Then, the candidates are verified based on the computed attribute score or the appearance of attribute set. However, the quality of the found communities of the two approaches are unpromising since the independent two stages fail to capture the correlations between structures and attributes in a joint fashion.

With the development of ML/DL, recently, GNN has been adopted for CS [12]. By recasting the community membership determination to a classification task, a model can learn via its prediction error feedback given the training samples and can adapt to a specific graph in an end-to-end way. Recently, Gao et al. proposed ICS-GNN [12] for interactive CS, which allows users to provide ground-truth for online incremental learning. The model is a query-specific model that fails to generalize to new query nodes. [13] proposes a graph neural network based model that is trained by a collection of query nodes with their ground-truth, and makes predictions for unseen query nodes in a single graph.

ML/DL for Graph Analytics. Apart from CS, ML/DL techniques are widely used in various graph analytical tasks, including classical combinatorial optimization problems [19],

[20], graph similarity search [21]–[23], subgraph matching [24]–[26], subgraph counting [27]–[29], shortest path query [30], community collapsing [31] and community detection [32]. In brief, the main ideas of these approaches contain learning a model-based algorithm heuristics [19], [20], [23], [25] to replace the traditional predefined heuristics, where Reinforcement Learning algorithms can be used; learning a workload-specific estimator for approximate query processing [21], [27], [28]; constructing a model-based database index for filtering or searching [22], [24], [26], [30], [31], which is node or graph embedding preserving task-related semantics. Our approach is in the first category regarding learning a meta heuristic for CS while the subtle difference is that we leverage metric learning to evaluate the community membership.

Meta-Learning on Graph. Meta-learning is a learning paradigm that learns the prior knowledge from multiple tasks, which can be swiftly transferred to a new task with only a few observed data. In general, the meta-learning approaches fall into three categories, i.e., black-box adaptation [33]–[36], optimization-based [37]–[39], and metric-based [40]–[42] approaches. Meta-learning has been adopted over graph data to deal with various graph learning tasks, including node classification [43], [44], link prediction [44], [45], graph classification [46], [47] and graph alignment [48], [49]. A brief survey that summarizes the applications and methods can be found in [50]. Here, GNN is widely used as the base model or core component of these approaches. The optimization-based, metric-based, or hybrid of optimization and metric-based [44] are used as the meta-learning strategies. However, all the existing approaches are oriented to graph learning tasks and cannot be directly applied to our CS task, where the input is specified by a personalized query node. Our approach CGNP can be regarded as metric-based since the predictive probability of CGNP is derived from inner-product similarity in a hidden embedding space.

III. Problem Statement

We consider an undirected simple graph $\mathcal{G} = (V, E)$, where $V(\mathcal{G})$ is the node set and $E(\mathcal{G})$ is the edge set. Let $n = |V(\mathcal{G})|$ and $m = |E(\mathcal{G})|$ denote the number of nodes and edges, respectively. The neighborhood of node v is denoted as $\mathcal{N}(v) = \{u|(u, v) \in E(\mathcal{G})\}$. The nodes may possess d attributes $\mathcal{A} = \{\mathcal{A}_1, \cdots, \mathcal{A}_d\}$. For each node v, a one-hot d-dimensional vector $\mathcal{A}(v) \in \{0, 1\}^d$ encodes whether v is associated with the d attributes in $\mathcal{A}$. In the following, we use $\mathbf{G}$ to represent a large data graph. A *community* in $\mathbf{G}$ is a cohesive subgraph $G = (V, E)$ induced by its node set $V(G)$, such that the nodes $V(G)$ are intensively connected within G whereas are sparsely connected with other nodes in the graph, i.e., $|E(G)| \gg |\{(u, v)|u \in V(G), v \in V(\mathbf{G}) \backslash V(G)\}|$. Below, we denote a community as an induced subgraph in a graph G by $\mathcal{C}(G)$.

Problem Statement (Community Search): The community search problem is to find the query-dependent community $\mathcal{C}_q$, for a user-given query node q in a graph G, such that $q \in \mathcal{C}_q(G)$. Distinguished from prior algorithmic approaches [15], [16], [51], the community $\mathcal{C}_q(G)$ in this paper is not restricted in any k-related subgraph, instead it is learned from given community membership ground-truth.

We construct a meta model $\mathcal{M}$ to support community search queries in a data graph $\mathbf{G}$ by multiple tasks. The model $\mathcal{M}$ is trained on a set of training tasks $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^N$. Here, a training task, $\mathcal{T}_i$, is a triplet $\mathcal{T} = (G, Q, L)$, where G is a subgraph of $\mathbf{G}$, $Q = \{q_1, \cdots, q_j|q_i \in V(G)\}$ is a set of j query nodes in G, and $L = \{l_{q_1}, \cdots, l_{q_j}\}$ is the ground-truth of the j query nodes, respectively. Specifically, l_q is a nonempty set of nodes in G w.r.t. the query node q, that contains a set of positive samples, $l_q^+ \subset \mathcal{C}_q(G)$, and a set of negative samples, $l_q^- \subset (V(G) \setminus \mathcal{C}_q(G))$. For a new test task $\mathcal{T}^* = (G^*, Q^*, L^*)$, the meta model $\mathcal{M}$ will exploit the query node set Q^* associated with the ground-truth L^* to adapt to task $\mathcal{T}^*$, and can make community search prediction for nodes in $V(G^*) \setminus Q^*$. Note that for test task, the number of query nodes in Q^*, named shots, is rather limited, i.e., $|Q^*| \ll |V(G^*)|$.

It is important to mention that the main idea behind multiple tasks is that it is difficult to obtain all required ground-truth to train. There are many possible scenarios with different ways that the training task set $\mathcal{D}$ and new test tasks are constructed. In this paper, we construct tasks from two dimensions: Single/Multiple graphs and Shared/Disjoint communities.

- *Single Graph Shared Communities.* The graphs in any training task, G, and test task, G^*, are subgraphs of a single large graph $\mathbf{G}$. The query nodes in training/test tasks are different but are from the same communities in $\mathbf{G}$.
- *Single Graph Disjoint Communities.* The graphs in any training task, G, and test task G^*, are subgraphs of a single large graph $\mathbf{G}$. The query nodes in training/test task are from different communities in $\mathbf{G}$, such that $\mathcal{C}_q(G) \cap \mathcal{C}_{q^*}(G^*) = \emptyset$, for all $q \in Q$ and $q^* \in Q^*$.
- *Multiple Graphs Disjoint Communities.* The graphs in any training task, G, and test task, G^*, are from different large data graphs. The query nodes in training/test tasks are from different communities. Here, all the subgraphs G in the training tasks are from the same domain, whereas a subgraph G^* in a test task can be in the same or a different domain.

Example 1: Assume that the DBLP graph in Fig. 1 is a single graph $\mathbf{G}$. A graph G in a training task $\mathcal{T} = (G, Q, L)$ and a graph G^* in a test task $\mathcal{T}^* = (G^*, Q^*, L^*)$ are subgraphs of $\mathbf{G}$. Suppose a subgraph G in a training task contains a part of the community that 'Jure' belongs to (i.e., the orange nodes) in Fig. 1. In the scenario of shared communities, some nodes in Q^* in a test task $\mathcal{T}^*$ may contain some ground-truth (e.g., orange nodes) that do not appear in any training tasks. The model $\mathcal{M}$ trained is to find the community for any query node in Q^* in the same test task $\mathcal{T}^*$ without any ground-truth associated with. In the scenario of disjoint communities, the ground-truth given in a test task may have nothing to do with the community that 'Jure' belongs to. The model $\mathcal{M}$ trained is to find the community for any query node in Q^* in the test task $\mathcal{T}^*$ without any ground-truth associated with. Note that

the community to be found is a different community that 'Jure' belongs to. For the scenario of multiple graphs, a subgraph G in a training task $\mathcal{T}$ is from a data graph, $\mathbf{G}$, whereas a subgraph G^* in a test task is from a different data graph $\mathbf{G}'$.

IV. Naive Approaches

To construct a meta model, a naive approach is to pre-train a Graph Neural Network (GNN) model over $\mathcal{D}$ and finetune the model for a new task $\mathcal{T}^*$. Below, we first introduce multi-label classification by GNN, which serves as the basis of the naive approaches and our meta-learning approach.

Given a graph G, a K-layer GNN follows a neighborhood aggregation paradigm to generate a new representation for each node by aggregating the representations of its neighbors in K iterations. Let $h_v^{(k)}$ denote the representation of a node v generated in the k-th iteration, which is a $d^{(k)}$ dimensional vector. In the GNN k-th iteration (layer), for each node $v \in V(G)$, an aggregate function $f_{\mathcal{A}}^{(k)}$ aggregates the representations of the neighbors of v that are generated in the (k-1)-th iteration as Eq. (1). Then, a combine function $f_{\mathcal{C}}^{(k)}$ updates the representation of v by the aggregated representation $a_v^{(k)}$ and previous representation $h_v^{(k-1)}$ as Eq. (2).

$$a_v^{(k)} = f_{\mathcal{A}}^{(k)}(\{h_u^{(k-1)} | u \in \mathcal{N}(v)\}) \tag{1}$$

$$h_v^{(k)} = f_{\mathcal{C}}^{(k)}(h_v^{(k-1)}, a_v^{(k)}) \tag{2}$$

The functions $f_{\mathcal{A}}^{(k)}$ and $f_{\mathcal{C}}^{(k)}$ are neural networks, e.g., linear transformation with non-linearities and optional Dropout for preventing overfitting. The neural network parameters from $f_{\mathcal{A}}^{(k)}$ and $f_{\mathcal{C}}^{(k)}$ are shared by all the nodes.

For a given task $\mathcal{T} = (G, Q, L)$, a GNN can be built by training over Q and L, then is deployed to make predictions for any query node $q \in V(G) \setminus Q$ as a query. Concretely, a binary query identifier $I_q(v) \in \{0, 1\}$ is concatenated with the attribute feature vector $\mathcal{A}(v)$ to form the initial node representation $h_v^{(0)}$, where $I_q(v) = 1$ if v is the query node q otherwise $I_q(v) = 0$. Through transformation of K layers, the 1-dimensional node representation $h_v^{(K)}$ is activated by a sigmoid function, i.e., $\hat{y}(v) = \mathsf{sigmoid}(h_v^{(K)})$, which is the likelihood that v is in the same community with query node q. The given Q and the ground-truth L provide the training data for the GNN model. For a known node $q \in Q$ with its ground-truth $l_q = (l_q^+, l_q^-)$, where l_q^+ and l_q^- are the positive and negative samples respectively, w.r.t. q, the binary cross entropy (BCE) loss in Eq. (3) evaluates the divergence between the predictive probability of the nodes from the positive and negative samples, under the GNN with parameter θ.

$$\mathcal{L}(q; \theta) = -\sum_{v^+ \in l_q^+} \log \hat{y}(v^+) - \sum_{v^- \in l_q^-} \log\big(1 - \hat{y}(v^-)\big) \tag{3}$$

Based on the simple GNN approach, we review three naive approaches which are simple combinations of GNN and meta/-transfer learning algorithms.

Feature Transfer. The learned parameters of shallow layers in neural network can be transferred to new tasks, instead of learning from scratch. The intuition is that the pre-trained low-level feature transformation can be shared with a new task. Thereby, we can train a GNN by the union of all the Q and L of every training task $\mathcal{T}$ in the training set $\mathcal{D}$. When a new task $\mathcal{T}^*$ arrives, the parameters of $f_{\mathcal{A}}^{(K)}$ and $f_{\mathcal{C}}^{(K)}$ will be updated by minimizing the BCE loss in Eq. (3) over Q^* and L^* by several gradient steps. However, the effectiveness of simple feature transfer is limited. For one thing, this approach is originally proposed for convolutional neural network (CNN) to process image data, which has an explicit feature hierarchy to be transferred. However, whether the same transfer mechanism well suits GNN over graph data still needs exploration. For the other thing, it is hard to control the gradient steps in the fine-tuning procedure for various test tasks.

Model-Agnostic Meta-Learning (MAML). A meta GNN model can be built by a model-agnostic meta-learning algorithm, MAML [37], over a set of training tasks $\mathcal{D}$. MAML is a two-level end-to-end optimization algorithm, where the lower level is to optimize task-specific parameters θ_i for one task $\mathcal{T}_i$ and the upper level is to optimize the task-common parameters θ^* over the training task set $\mathcal{D}$. The learned task-common parameters θ^* will be used as the neural network initialization and updated by a few gradient steps to generalize a new task $\mathcal{T}^*$, given the few-shot task-specific data Q^* and L^*. To be concrete, training data $Q_i = \{q_j\}_{j=1}^{J}$ and $L_i = \{l_{q_j}\}_{j=1}^{J}$ of one training task $\mathcal{T}_i$ are divided into two sets, $\mathcal{S}_i = \{(q_j, l_{q_j})\}_{j=1}^{J'}$ and $\mathcal{Q}_i = \{(q_j, l_{q_j})\}_{j=J'+1}^{J}$. $\mathcal{S}_i$ is called *support set* and $\mathcal{Q}_i$ is called *query set*. The task-specific parameters θ_i is updated by the support set of $\mathcal{T}_i$ as Eq. (4) in an inner loop, and the task-common parameters θ^* is updated by the query set over $\mathcal{D}$ in an outer loop as Eq. (5), by gradient descent with learning rates α and β, respectively.

$$\theta_i \leftarrow \theta - \alpha \nabla_\theta \sum_{(q, l_q) \in \mathcal{S}_i} \mathcal{L}(q; \theta) \tag{4}$$

$$\theta^* \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim \mathcal{D}} \sum_{(q, l_q) \in \mathcal{Q}_i} \mathcal{L}(q; \theta_i) \tag{5}$$

Although MAML is an effective and fairly general framework, it suffers from a variety of problems, including training instability, restrictive model generalization performance and extensive computational overhead [52]. To alleviate the computational overhead, Reptile, is proposed as a first-order meta-learning algorithm [39]. Reptile directly updates the task-common parameters θ^* by the first-order gradients, bypassing the computation of the high-order derivatives. First, the inner loop follows MAML to compute the task-specific parameters θ_i for $\mathcal{T}_i$ as Eq. (4). Then, in the outer loop, the task-common parameters θ^* are directly updated by the difference of θ_i to current parameters θ as shown in Eq. (6).

$$\theta^* \leftarrow \theta + \beta \frac{1}{|\mathcal{D}|} \sum_{\mathcal{T}_i \sim \mathcal{D}} (\theta_i - \theta) \tag{6}$$

Here, since evaluating the query set $\mathcal{Q}_i$ of $\mathcal{T}_i$ is unnecessary, Reptile does not split $\mathcal{Q}_i$ and $\mathcal{S}_i$ for updating θ_i, but update θ_i by all the training data of $\mathcal{T}_i$ in the inner loop.

Graph Prototypical Network (GPN). Prototypical Network [40] is an effective approach for few-shot classification, which learns a metric space in which classification is performed by computing distances to the centroid (prototype) representation of each class. Different from general classification, the prototype representation of CS should be query-specific. For a query node q, two prototypes, c_q^+ and c_q^-, are computed by the mean representations of the positive and negative samples in the ground-truth l_q, respectively (Eq. (7)). Here, $h_v^{(K)}$ is the node representation of v of the K-th layer of GNN, generated by Eq. (2). Then, the likelihood that node v is in the same community with q is predicted by its distances to the prototypes as Eq. (8), given a distance function dist.

$$c_q^+ = \frac{1}{|l_q^+|} \sum_{v^+ \in l_q^+} h_{v^+}^{(K)},\ c_q^- = \frac{1}{|l_q^-|} \sum_{v^- \in l_q^-} h_{v^-}^{(K)} \tag{7}$$

$$\hat{y}(v) = \mathsf{softmax}\Big([-\mathsf{dist}(h_v^{(K)}, c_q^+) \| - \mathsf{dist}(h_v^{(K)}, c_q^-)] \Big) \tag{8}$$

In the training stage, ground-truth sets l_q^+ and l_q^- are split into two sets, respectively. One is used to compute the prototypes in Eq. (7) and the other is to compute the BCE loss in Eq. (3) for parameter update. It is worth noting that each query node must compute its own prototypes as it has its own communities. Therefore, GPN cannot support query node in the test task without any ground-truth, where computing prototypes is infeasible.

V. CGNP for CS

To overcome the disadvantages of the naive approaches, we devise a novel meta-learning framework for CS, named Conditional Graph Neural Process (CGNP), on the basis of Conditional Neural Process (CNP) [14]. In this section, we first introduce CNP as a preliminary, then present the core idea of CGNP for CS as an overview of our framework.

CNP is a neural network approximation of stochastic process, e.g., Gaussian Process (GP). It directly models the predictive distribution conditioned on an arbitrary number of context observations by neural networks. Specifically, given observed data $X = \{x_i\}_{i=1}^N$ with corresponding ground-truth $Y = \{y_i\}_{i=1}^N$, CNP models the predictive distribution of new data x^* with the target y^*, $p(y^*|x^*, X, Y)$, by the neural network architecture in Eq. (9).

$$p(y^*|x^*, X, Y) = \rho_\theta \Big(x^*, \bigoplus_{i=1}^N \phi_\theta(x_i, y_i) \Big) \tag{9}$$

Here, $\phi_\theta : X \times Y \to \mathbb{R}^d$ and $\rho_\theta : X \times \mathbb{R}^d \to \mathbb{R}^e$ are neural networks. The big $\oplus$ is a commutative operation that takes elements in $\mathbb{R}^d$ and aggregates them into a single element of fixed length $\mathbb{R}^d$. ϕ_θ is the encoder that transforms pairs of (x_i, y_i) into d-dimensional hidden representations. The big $\oplus$ aggregates N representations into a context representation in a permutation-invariant fashion which memorizes the whole dataset X and Y. To deal with a query for new observation x^*, a decoder ρ_θ takes the context and x^* as inputs and makes a final prediction for x^*.

Similar to stochastic process, CNP can be used to build meta models via learning the prior of data generation, where each data instance is a collection of (x_i, y_i), i.e., a task. The difference lies in that stochastic process, e.g., GP, explicitly specifies the prior distribution, and optimizes the hyper-parameters of the prior by maximum likelihood. CNP instead explicitly parameterizes the predictive distribution as neural networks thereby learning the prior implicitly.

Conditional Graph Neural Process (CGNP). The CGNP model we propose is a graph specification of CNP for query-dependent node classification. For a CS task $\mathcal{T} = (G, Q, L)$, CGNP directly models the predictive probability $p(\hat{l_{q^*}}|q^*, \mathcal{T})$ for a new query node $q^* \in V(G) \setminus Q$, where $\hat{l_{q^*}} = \{\hat{l_{q^*}}(v)\}_{v \in V(G)} \in \{0, 1\}^n$ is the binary target prediction for all the nodes in G. We instantiate the encoder ϕ_θ that encodes each query node q with its ground l_q in the task $\mathcal{T}$, the commutative operation that generates the context representation of $\mathcal{T}$, and the encoder ρ_θ that predicts $p(\hat{l_{q^*}}|q^*, \mathcal{T})$ as Eq. (10). CGNP inherits the interpretation of CNP [14] that using neural networks to mimic an *implicit kernel function*, $\mathcal{K}_\theta(\cdot, \cdot)$, which evaluating the similarity between an observed query node q and the target query node q^*. The predictive probability is the summation of the observed ground-truth l_q, weighted by the similarities of query nodes as Eq. (11).

$$p(\hat{l_{q^*}}|q^*, \mathcal{T}) = \rho_\theta \Big(q^*, \bigoplus_{(q, l_q) \in (Q, L)} \phi_\theta(q, l_q) \Big) \tag{10}$$

$$\approx \sum_{(q, l_q) \in (Q, L)} \mathcal{K}_\theta(q^*, q) \odot l_q \tag{11}$$

Like k-nearest neighbor (KNN) algorithm, the non-parametric metric learning intuition makes CGNP promising for small samples and classification tasks as CS. Unlike KNN, for CGNP, the kernel function $\mathcal{K}_\theta(\cdot, \cdot)$, as well as the multiplication operation $\odot$ is implicitly learned from the data by our instantiated ρ_θ, ϕ_θ and big $\oplus$. In other words, KNN and CGNP memorize the input data in different ways. KNN persists the input data by simple concatenation whereas CGNP persists it by learning a hidden context representation.

Apart from the implicit kernel $\mathcal{K}_\theta(\cdot, \cdot)$ derived from CNP, in particular, we also impose an explicit metric objective on the learning of CGNP. For CS task, since we only need a binary prediction to indicate whether a node v is a community member of the query node q^*, we let the predictive probability of the membership be determined by the distance of v and q^* in a hidden space H as Eq. (12). The mapping function from the initial node features to that hidden space is specified by the neural network encoder ϕ_θ, decoder ρ_θ and the commutative operation big $\oplus$ of CNP.

$$p(\hat{l_{q^*}}(v)|q^*, \mathcal{T}) \models \mathsf{dist}(H(q^*), H(v)) \tag{12}$$

In this explicit metric-based modeling perspective, we use inner product similarity to distinguish between membership and nonmembership in the hidden space H that is learned in the training process. The meta CGNP model is also a common

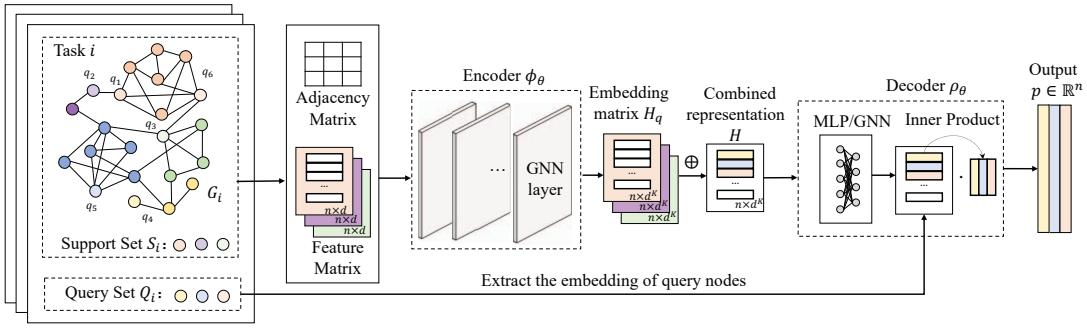


Fig. 2: The Architecture of CGNP

neural network mapping, shared by multiple CS tasks, that maps nodes for partitioning in a task-specific hidden space H. Note the kernel $\mathcal{K}_\theta(\cdot,\cdot)$ in Eq. (11) and the distance $\mathsf{dist}(\cdot,\cdot)$ in Eq. (12) are two different concepts. The kernel measures the similarity between two input query nodes q and q^* regarding their community-member relationship with all the remaining nodes in G, whereas the distance measures the closeness of a query node q and one remaining node v in G regarding their community membership. To learn a task-common kernel and distance mapping, CGNP iterates on the training task set to optimize the neural network parameters θ, where data in one task is processed as a batch. Compared with the two-level optimization-based algorithm MAML, CGNP learns the prior knowledge of CS by metric learning for node clustering/partitioning, which better exploits small data for classification and avoids unstable and inefficient parameter adaptation in the test stage. The metric learning principle of CGNP is also different from that of GPN. GPN computes positive and negative prototypes, c_q^+ and c_q^-, for each query node q by its ground-truth. Inference on other nodes is based on their distances to the prototypes. In contrast, CGNP directly models and evaluates the distances between the query nodes and the remaining nodes, thereby supporting queries without any ground-truth in test tasks.

VI. CGNP Model Design

We elaborate on how to design a CGNP model for the query-dependent node classification over graphs. CGNP adopts an encoder-decoder based architecture and operates on task-level. Fig. 2 delineates the architecture of CGNP, which is composed of a GNN based encoder operating on query-level representation, a commutative operation, big $\oplus$, combining query-level representation to task-level context, and a decoder to perform final predictions.

GNN Encoder ($\phi_\theta(q, l_q, G)$). For each query node $q \in Q$ and its corresponding ground-truth $l_q \in L$, the encoder $\phi_\theta(q, l_q, G)$ is a K-layer GNN that maps the pair (q, l_q) together with the graph G to a node embedding matrix $H_q = \{h_v^{(K)}\}_{v \in V(G)} \in \mathbb{R}^{n \times d^K}$. Here, $h_v^{(K)}$ is a d^K-dimensional output of the K-th layer of GNN for node v. The subscript q of H_q indicates the node embedding H_q is generated particularly for query node q, as all the query nodes in Q share the same GNN encoder. Specifically, as the inputs of GNN, the adjacency matrix of graph G is used for message passing of GNN, and (q, l) determines the initial node $h_v^{(0)}$ as Eq. (13), where $\|$ is the vector/bit concatenation operation, $\mathcal{A}(v)$ is the attribute features of node v. In Eq. (13), $I_l(v) \in \{0, 1\}$ is a binary ground-truth identifier which distinguishes nodes within and without a same community, under the close world assumption.

$$h_v^{(0)} = [I_l(v) \| \mathcal{A}(v)], I_l(v) = \begin{cases} 1 & v \in l_q^+ \cup \{q\} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

We can concatenate auxiliary features, e.g., the core number and local clustering coefficient of node v on $h_v^{(0)}$ to exploit additional structural information. The intuition of the GNN encoder is to generate a view, H_q, for the whole graph given an observation (q, l_q) by message passing. A collection of views will be aggregated by the commutative operation big $\oplus$. This idea is enlightened by a CNP specialization for 3D scene understanding and rendering, Generative Query Network (GQN) [53], where few-shot observed 3D views are summed up for predicting the view of a new query perspective. It is worth mentioning that, to the best of our knowledge, we are the first to introduce the insight of GQN to graph domain.

Commutative Operation ($\oplus$). To combine the views H_q for all query nodes in Q into one context representation, CGNP is equipped with three choices of commutative operations, sum, average and self-attention. All of the three operations are permutation-invariant.

Sum & Average are simple yet widely used pooling operations in many CNP instances [14], [53]. The sum operation conducts element-wise sum up as Eq. (14) and average further imposes a denominator of $|Q|$.

$$H = \sum_{q \in Q} H_q \tag{14}$$

Self-Attention is inspired by Attentive Neural Process (ANP) [54] and GP. Instead of giving the same weight to aggregate multiple data points, ANP and GP aggregate observed data by self-adaptive weights by self-attention [55] and GP's kernel function, respectively. Thereby, CGNP leverages the self-attention to combine the node representations derived from all the query nodes, weighted by a set of learnable

weights $\{w_q\}_{q\in Q} \in \mathbb{R}^{|Q|}$ as $H = \sum_{q\in Q} w_q H_q$. The weights $\{w_q\}_{q\in Q}$ are shared by all the nodes in G. Specifically, to compute the attention weight $\{w_q\}_{q\in Q}$ by the multiple views $\{H_q\}_{q\in Q}$, let $\mathcal{H} = \{H_q[v]\}_{q\in Q} \in \mathbb{R}^{|Q|\times d^K}$ be the matrix stacked by the $|Q|$ node embeddings in $\{H_q\}_{q\in Q}$ for an arbitrary node v. In Eq. (15), $\mathcal{H}_1, \mathcal{H}_2 \in \mathbb{R}^{|Q|\times d'}$ are transformed by linear weight matrices $W_1, W_2 \in \mathbb{R}^{d^K\times d'}$, respectively. $\{w_q\}_{q\in Q}$ is computed by the inner product of $\mathcal{H}_1$ and the transpose of $\mathcal{H}_2$ followed by a softmax function that normalizes the weights to a probability, as Eq. (16) shows.

$$\mathcal{H}_1 = \mathcal{H}W_1,\ \mathcal{H}_2 = \mathcal{H}W_2, \tag{15}$$

$$\{w_q\}_{q\in Q} = \mathsf{softmax}\left(\frac{\langle \mathcal{H}_1, \mathcal{H}_2^T\rangle}{\sqrt{d'}}\right) \tag{16}$$

Decoder ($\rho_\theta(q^*, H)$). Given the combined context H, a decoder $\rho_\theta(q^*, H)$ estimates the membership for a new query node q^*, $p(l^*|q^*, H) \in \mathbb{R}^n$, conditioned on the memorized context H. We design three decoders with different complexities, a simple inner product decoder, multi-layer perception (MLP) decoder and GNN decoder. The latter two decoders MLP and GNN are also based on inner product.

Inner Product Decoder is free of parameters and only operates on the context H. Since H is a node embedding combined by multiple views, we can directly compute the node similarities between the embedding of a query node q and all the other nodes. We use the inner product operation, $\langle\cdot,\cdot\rangle$, to compute the similarity score as Eq. (17), followed by a sigmoid function to predict the probability that one node is in the same community with query node q^*. The inner product operation indicates that the smaller the angle of two node embeddings in the vector space, the more likely the two nodes are from the same community.

$$\hat{p(l_{q^*}}|q^*, \mathcal{T}) = \mathsf{sigmoid}(\langle H[q^*], H\rangle) \tag{17}$$

MLP & GNN Decoder. MLP decoder firstly transforms the context matrix H by an MLP, then feeds the transformed H to an inner product operation of Eq. (17). Similarly, firstly transforms the context matrix H by a K-layer GNN, followed by the inner product. Note the GNN here is independent to the GNN in the encoder. In contrast to the inner product decoder, the MLP and GNN encoder impose additional parametric transformations on the combined context embedding H to improve the modeling capability of the decoder. The difference between MLP and GNN lies in that GNN further allows message passing among the nodes whereas the MLP transforms each node independently.

In the following, we present the learning algorithms to train a meta CGNP model $\mathcal{M}$ and adapt the model to new tasks. Recall that CGNP is to model a generative process of tasks $f \sim \mathcal{D}$, where $\mathcal{D}$ is the set of training tasks $\{\mathcal{T}_i\}_{i=1}^N$. Suppose the tasks are independent and the query nodes are independent in each task. The marginal likelihood of CGNP over $\mathcal{D}$ is

$$p(\{L_1, \cdots, L_N\}|\{Q_1, \cdots, Q_N\}, \theta) = \prod_{\mathcal{T}_i\in\mathcal{D}} p(L_i|Q_i) \tag{18}$$

Algorithm 1: CGNP Meta Train

Input : training task set $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^N$, learning rate α, number of epochs T
Output: parameters θ of meta model $\mathcal{M}$

1 **for** *epoch* $\leftarrow 1$ *to* T **do**
2 Shuffle the task set $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^N$;
3 **for** $\mathcal{T}_i = (G_i, Q_i, L_i) \in \mathcal{D}$ **do**
4 $\mathcal{S}_i, \mathcal{Q}_i \sim (Q_i, L_i)$; ▷ allocate support and query sets
5 **for** $(q, l_q) \in \mathcal{S}_i$ **do**
6 $H_q \leftarrow \phi_\theta(q, l_q, G_i)$; ▷ compute query-specific view
7 $H \leftarrow \bigoplus_{(q,l_q)\in\mathcal{S}_i} H_q$; ▷ compute context embedding (Eq. (14))
8 **for** $(q, l_q) \in \mathcal{Q}_i$ **do**
9 $\hat{p(l_q}|q, \mathcal{S}_i) \leftarrow \rho_\theta(q, H)$; ▷ compute pred. prob. (Eq. (17))
10 Compute the Loss $\mathcal{L}(q)$ by $\hat{p(l_q}|q, \mathcal{S}_i)$ and l_q;
11 $\mathcal{L} \leftarrow \sum_{(q,l_q)\in\mathcal{Q}_i} \mathcal{L}(q)$;
12 $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}$; ▷ update model parameters
13 **return** θ;

Algorithm 2: CGNP Meta Test

Input : test task $\mathcal{T}^* = (G^*, Q^*, L^*)$, parameter θ of meta model $\mathcal{M}$, a query node $q^* \in V(G^*) \setminus Q^*$
Output: predictive probability of q^*

1 $\mathcal{S}^* \leftarrow (Q^*, L^*)$;
2 **for** $(q, l_q) \in \mathcal{S}^*$ **do**
3 $H_q \leftarrow \phi_\theta(q, l_q, G^*)$; ▷ compute query-specific view
4 $H \leftarrow \bigoplus_{q\in\mathcal{S}^*} H_q$; ▷ compute context embedding (Eq. (14))
5 $\hat{p(l_{q^*}}|q^*, \mathcal{S}^*) \leftarrow \rho_\theta(q^*, H)$; ▷ compute pred. prob. (Eq. (17))
6 **return** $\hat{p(l_{q^*}}|q^*, \mathcal{S}^*)$;

Similar to MAML, for one training task $\mathcal{T}_i$, we split the training data $Q_i = \{q_j\}_{j=1}^J$ and $L_i = \{l_{q_j}\}_{j=1}^J$ into the support set $\mathcal{S}_i = \{(q_j, l_{q_j})\}_{j=1}^{J'}$ and query set $\mathcal{Q}_i = \{(q_j, l_{q_j})\}_{j=J'+1}^J$. The learning objective is to minimize the negative log-likelihood of the query set $\mathcal{Q}_i$ conditioned on the support set $\mathcal{S}_i$ across all the tasks in $\mathcal{D}$ as Eq. (19). The negative log-likelihood loss in Eq. (19) is in accordance with the BCE loss (Eq. (3)) of the query nodes in the query set $\mathcal{Q}_i$.

$$\begin{aligned}\mathcal{L} &= -\sum_{\mathcal{T}_i\in\mathcal{D}}\sum_{(q,l_q)\in\mathcal{Q}_i} \log p(l_q|q, \mathcal{S}_i) \\ &= -\sum_{\mathcal{T}_i\in\mathcal{D}}\sum_{(q,l_q)\in\mathcal{Q}_i}\left(\sum_{v^+\in l_q^+}\log\hat{y}(v^+) + \sum_{v^-\in l_q^-}\log\big(1-\hat{y}(v^-)\big)\right)\end{aligned} \tag{19}$$

Meta Training. In the training stage, given the training task set $\mathcal{D}$, learning rate α, and the number of epochs T, a meta CGNP model is trained by optimizing the negative log-likelihood of Eq. (19) by stochastic gradient descent. Algorithm 1 presents the training process. In each epoch (line 1-12), all the training tasks are randomly shuffled in line 2. For each task $\mathcal{T}_i$, we get the allocated support set $\mathcal{S}_i$ and query set $\mathcal{Q}_i$ from the given query node and ground-truth (line 4). First, each query node q associated with the ground-truth l in the support set $\mathcal{S}_i$, together with the graph structure G_i is fed into the GNN encoder, ϕ_θ , to generate a query-specific view H_q (line 5-6). Second, in line 7, all the views are aggregated into the context matrix H by the permutation-invariant operation big

$\oplus$, e.g., by the summation aggregation of Eq. (14). Third, for each query node in the query set $\mathcal{Q}_i$, we compute its predictive probability and loss in line 9-10, via evaluating the inner product similarities between node presentations and the query representation in H as Eq. (17). Fourth, the model is updated by one gradient step of the aggregated task-specific loss (line 11-12).

Meta Testing. For a test task $\mathcal{T}^*$ with graph G^*, few-shot query nodes Q^* and the associated ground-truth L^*, Algorithm 2 presents the steps to predict the community members for a query node q^*. The whole Q^* and L^* serve as the support set $\mathcal{S}^*$ (line 1), followed by computing the context representation H (line 2-4). Finally, the query node q^* and context H are fed into the decoder network ρ_θ to obtain the prediction.

Example 2: We use a real example to illustrate how CGNP works in Algorithm 2 on a test task $\mathcal{T}^* = (G^*, Q^*, L^*)$ of a DBLP subgraph, G^*, in Fig. 1. Suppose the task possesses query nodes $Q^* = \{q_1, q_2, q_3\}$ correspond to 3 users {Julian, Jaewon, Deepayan}, respectively, and the corresponding ground-truth $L^* = \{l_1, l_2, l_3\}$. Each l_i is composed a positive node set l_i^+, and a negative node set l_i^-. First, for the 3 pairs $(q_1, l_1), (q_2, l_2), (q_3, l_3)$, by line 2-3, the GNN encoder ϕ_θ generates 3 node embedding matrices $H_1, H_2, H_3 \in \mathbb{R}^{n \times d^K}$ as the query-specific views, respectively, where $n = |V(G)|$. Second, by line 4, the combine operator big $\oplus$ aggregates the three matrices H_1, H_2, H_3 to one context embedding H. Given a query node q^* in the subgraph G, e.g, Jure, by line 5 the inner product decoder ρ_θ predicts the probability of community membership of Jure for all the nodes, by computing the inner product similarity of vector $H[q^*]$ and H followed by a sigmoid function as Eq. (17).

Computation Complexity. We analyze the time complexity of CGNP in brief. To be concise, we assume fixed dimension vector add, multiplication, and inner product take constant time when the dimension is far smaller than the graph node number n. For the GNN encoder of CGNP, the time complexity is $\mathcal{O}(Km|\mathcal{S}|)$ for a single task, where K is the number of GNN layers, m is the number of edges and $|\mathcal{S}|$ denotes the number of shots. The complexity of the big $\oplus$ operation is $\mathcal{O}(n|\mathcal{S}|)$ for the sum and average pooling and $\mathcal{O}(n|\mathcal{S}|^2)$ for the self-attention, respectively. For the decoders, the inner product operation takes $\mathcal{O}(n|\mathcal{Q}|)$ time, and an MLP decoder and K' layer GNN decoder takes extra $\mathcal{O}(n|\mathcal{Q}|)$ and $\mathcal{O}(K'm|\mathcal{Q}|)$ cost, respectively. In total, the complexity of the meta test algorithm, Algorithm 2, is $\mathcal{O}(c(n+m))$, where c is a constant determined by $K, K', |\mathcal{S}|, |\mathcal{Q}|$. And the training complexity of Algorithm 1 is $\mathcal{O}(TNc(n+m))$, where T and N are the numbers of iterations and training tasks.

VII. Experimental Studies

We introduce the experimental setup (section VII-A) and report our substantial results as follows: ① compare the effectiveness of CGNP under different task configurations (section VII-B), ② evaluate the efficiency of CGNP with the baselines, and conduct scalability test for learning-based approaches (section VII-C), ③ investigate the effect of the volume of the ground-truth on the performance of CGNP (section VII-D), and ④ conduct the ablation studies on the CGNP model regarding the GNN layer and the commutative operation (section VII-E).

TABLE I: Profile of Datasets

Dataset		$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{C}(G)\rvert$
Cora		2,708	5,429	1,433	7
Citeseer		3,327	4,732	3,703	6
Arxiv		199,343	1,166,243	N/A	40
DBLP		317,080	1,049,866	N/A	5,000
Reddit		232,965	114,615,892	N/A	50
Facebook	0	348	2,867	224	24
	107	1,046	27,795	576	9
	348	228	3,420	162	14
	414	160	1,853	105	7
	686	171	1,827	63	14
	698	67	337	48	13
	1684	793	14,817	319	17
	1912	756	30,781	480	46
	3437	548	5,361	262	32
	3980	60	206	42	17

A. Experimental Setup

Datasets: We use 6 real-world graph datasets, including five single graphs (Cora, Citeseer, Arxiv, Reddit, DBLP) and one multiple graph (Facebook). Table I lists the profile of the 6 datasets. Cora, Citeseer and Arxiv are citation networks whose nodes represent research papers and edges represent citation relationships. We use node class labels to simulate the communities derived from the paper citation, which reveal the research topics that papers belong to. DBLP [56] is a co-authorship network where nodes represent authors and two authors are connected if they collaborate on at least one paper. A ground-truth community is by the publication venue. Reddit is collected from an online discussion forum, where nodes refer to posts, and an edge between two posts exists if a user comments on both of the posts. The ground-truth is the communities that posts belong to. Facebook is a dataset containing 10 ego-centric social networks, which have friendship community ground-truth. Cora, Citeseer, and Facebook have discrete node attributes. The attributes of Cora and Citeseer are the keywords in the papers and the attributes of Facebook are the user properties. For Cora, Citeseer, and Facebook, we use one-hot representations of the attributes as the node features, concatenating with the core number and local cluster coefficient of the node. We use core number and local cluster coefficient alone as node features, for Arxiv, DBLP and Reddit, as they do not have node attributes.

Tasks & Queries: We test our CGNP in different subgraphs of same graph, different graphs, and different application scenarios, following the 4 different types of tasks described in section III: ① Single Graph Shared Communities Task (SGSC), ② Single Graph Disjoint Communities Task (SGDC), ③ Multiple Graphs from One Domain Task (MGOD), and ④ Multiple Graphs from Different Domains Task (MGDD). For SGSC, SGDC and MGDD, one task is generated by sampling a subgraph of 200 nodes by BFS. The query nodes are randomly drawn from a sampled subgraph, G, where we assign 1 or 5 query nodes to the support set $\mathcal{S}$, i.e., 1-shot or 5-shot tasks, and assign 30 query nodes to the query set $\mathcal{Q}$ disjointly. It

is worth noting that the query nodes may be from the same ground-truth communities for SGSC whereas the query nodes must be from disjoint communities for SGDC. For each query q, we randomly drawn 5 positive samples from the community of q, $\mathcal{C}_q(G)$, to construct l_q^+ and 10 negative samples from $V(G) \setminus \mathcal{C}_q(G)$ to construct l_q^-. Here, for SGSC and SGDC, we generate 100 training tasks for Cora, Citeseer, Arxiv, Reddit and DBLP, and generate 50 valid tasks and 50 test tasks for the five datasets, respectively. For MGOD, we use one Facebook ego-network as the graph in one task, and sample the same numbers of queries and labels as discussed above. Ten tasks are split into 6 for training, 2 for validation, and 2 for testing. For MGDD, we also generate 100 tasks of Citeseer for training, 50 tasks of Cora for validation, and 50 tasks of Cora for testing, denoted as Cite2Cora.

Baselines: To comprehensively evaluate the performance of CGNP framework for CS, we compare with 10 baseline approaches, including 3 graph algorithms, 4 naive approaches discussed in section IV, 3 traditional ML/DL-based approaches. ❶ Attributed Truss Community Search (ATC) [16]. It is an attributed community search algorithm given the input of query nodes and attributes. Firstly, it finds the maximal (k, d)-truss containing the query nodes. Then, the algorithm iteratively removes unpromising nodes from the truss, which has a small attribute score. ❷ Attributed Community Query (ACQ) [15]. It aims to find subgraph whose nodes are tightly connected and share common attributes with the given query node. ❸ Closest Truss Community (CTC) [51]. It is a k-truss based community search framework for non-attributed graphs. Given a set of query nodes, Q, a greedy algorithm finds a k-truss with the largest k that contains Q and has the minimum diameter among the truss. ❹ Model-Agnostic Meta-Learning (MAML) [37]. We use GNN as the base model. The task-specific parameters of GNN are updated in an inner loop as Eq. (4), and the task-common parameters are updated in an outer loop as Eq. (5) over all training tasks. ❺ First-Order Meta-Learning (Reptile) [39]. As a first-order alternative of MAML, Reptile adopts the same GNN as the base model. Task-common parameters are updated by Eq. (6) in an outer loop, over all the training tasks. ❻ Feature Transfer (FeatTrans). A base GNN model is pre-trained on all the training tasks. For a test task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$, the final layer of the GNN is finetuned on the support set $\mathcal{S}^*$ by one gradient step, while all the other parameters are kept intact. ❼ Graph Prototypical Network (GPN). For each query q, 3 positive samples and 3 negative samples are randomly drawn from l_q to compute the query-specific prototypes. We use Euclidean distance as the distance function in Eq. (8). ❽ Supervised GNN (Supervised). One GNN model is trained for each test task from scratch by the few-shot data in $\mathcal{S}^*$. ❾ ICS-GNN (ICS-GNN) [12]. For each query node q, a GNN model is trained by some positive and negative samples and predicts a score for the remaining nodes. Then, the algorithm finds a subgraph connected to q, with a fixed number of nodes, aiming to maximize the summation of the scores predicted by GNN. ❿ AQD-GNN (AQD-GNN) [13]. The setting is similar to Supervised. For each test task, AQD-GNN trains the model from scratch by the few-shot data in $\mathcal{S}^*$ and test in $\mathcal{Q}^*$. It is worth noting that GPN and ICS-GNN are different from other learning-based approaches, where test query nodes are required to have ground-truth. GPN uses the ground-truth to compute the query-specific prototypes while ICS-GNN uses the ground-truth to train a query-specific model. These two approaches *cannot fully generalize* to query nodes without any prior knowledge of membership.

Implementation and Settings: We give the settings of 8 ML approaches, including our CGNP and 7 baselines, MAML, Reptile, FeatTrans, GPN, Supervised, ICS-GNN and AQD-GNN. For the GNN encoder of CGNP and the base GNN models of the 6 baselines, the number of the GNN layers is 3, where each GNN layer has 128 hidden units and a Dropout probability of 0.2 by default.

We investigate popular GNN layers, including the vanilla Graph Convolutional Network (*GCN*) [57], Graph Attention Network (*GAT*) [58] and *GraphSAGE* [59], and finally choose *GAT* by default due to its high performance. For the MLP decoder of CGNP, we use a two-layer MLP with 512 hidden units. For the GNN decoder of CGNP, we use a two-layer GNN which has the same configuration as the encoder.

The learning framework of CGNP and the 7 ML baselines are built on PyTorch [60] with PyTorch Geometric [61]. We use Adam optimizer with a learning rate of 5×10^{-4} to train CGNP, GPN, ICS-GNN, Supervised, and FeatTrans by 200 epochs. For MAML and Reptile, the inner loop performs 10 gradient steps for training and 20 steps for testing, with a learning rate of 5×10^{-4}, and the learning rate for the outer loop is 10^{-3}. It is worth mentioning that the performance of CGNP is robust in the range of empirical training hyper-parameters. By default, the training and prediction are conducted on a Tesla V100 with 16GB memory. ATC, ACQ and CTC are tested on the same Linux server with 32 Intel(R) Silver 4,215 CPUs and 128GB RAM.

Evaluation Metrics: To evaluate the quality of the found result, we use accuracy, precision, recall and F1-score between the prediction and the ground-truth. F1-score is the harmonic average of precision and recall, which better reflects the overall performance.

B. Effectiveness

We investigate the overall performance of CGNP on the four types of tasks (SGSC, SGDC, MGOD and MGDD), for 1-shot and 5-shot learning. The number of shots is the number of query nodes provided in the support set. The three variants of CGNP, CGNP with simple inner product decoder (CGNP-IP), CGNP with MLP decoder (CGNP-MLP), and CGNP with GNN decoder (CGNP-GNN) are compared with 10 baseline approaches.

Table II presents the performance for tasks of single graph with shared/disjoint communities. Here, we highlight the first (purple) and the second (blue) best F1. We observe that CGNP

TABLE II: Performance on SGSC and SGDC Tasks (First and Second Best F1 Scores are Highlighted)

Dataset	Task config.	Single Graph with Shared Communities								Single Graph with Disjoint Communities							
		1-shot				5-shot				1-shot				5-shot			
	Methods	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
Citeseer	ATC	0.4759	0.8366	0.1044	0.1856	0.4623	0.8344	0.1005	0.1793	0.5393	0.8288	0.1131	0.1990	0.5373	0.8357	0.1144	0.2013
	CTC	0.4386	0.8585	0.0226	0.0440	0.4264	0.8653	0.0225	0.0439	0.5043	0.8262	0.0262	0.0508	0.5010	0.8293	0.0261	0.0507
	MAML	0.5293	0.6450	0.3942	0.4894	0.5494	0.6937	0.4108	0.5160	0.5528	0.5835	0.4071	0.4796	0.5738	0.6277	0.4022	0.4903
	Reptile	0.5474	0.6382	0.4825	0.5495	0.5550	0.6886	0.4363	0.5342	0.5812	0.6038	0.5022	0.5483	0.5970	0.6500	0.4531	0.5340
	FeatTrans	0.4719	0.6625	0.1571	0.2540	0.4548	0.6692	0.1337	0.2229	0.5044	0.5346	0.1602	0.2465	0.4925	0.5127	0.0819	0.1413
	GPN	0.1744	0.1159	0.1564	0.1332	0.1383	0.1208	0.1441	0.1314	0.4498	0.4632	0.6199	0.5302	0.2957	0.3960	0.3263	0.3578
	Supervised	0.5492	0.6574	0.4429	0.5293	0.5688	0.6895	0.4780	0.5646	0.5751	0.6072	0.4544	0.5198	0.6221	0.6692	0.5110	0.5795
	ICS-GNN	0.4856	0.7115	0.1783	0.2852	0.4793	0.7094	0.1760	0.2821	0.5424	0.6738	0.1925	0.2994	0.5411	0.6658	0.1905	0.2963
	AQD-GNN	0.5036	0.5993	0.4406	0.5079	0.5263	0.5806	0.6816	0.6270	0.5558	0.5717	0.5291	0.5496	0.5072	0.5100	0.7761	0.6155
	CGNP-IP	0.6076	0.6429	0.7071	0.6734	0.6150	0.6562	0.7176	0.6855	0.5611	0.5488	0.7469	0.6327	0.5626	0.5515	0.7584	0.6386
	CGNP-MLP	0.6041	0.6556	0.6490	0.6523	0.6160	0.6710	0.6735	0.6723	0.5510	0.5427	0.7174	0.6179	0.5773	0.5633	0.7588	0.6466
	CGNP-GNN	0.6133	0.6393	0.7443	0.6878	0.6158	0.6513	0.7367	0.6914	0.5685	0.5527	0.7730	0.6446	0.5765	0.5631	0.7532	0.6444
Arxiv	ATC	0.5802	0.7253	0.0734	0.1333	0.5850	0.7349	0.0757	0.1373	0.5767	0.7875	0.0542	0.1015	0.5804	0.7930	0.0557	0.1042
	CTC	0.5751	0.7693	0.0484	0.0911	0.5795	0.7783	0.0501	0.0942	0.5733	0.8160	0.0411	0.0782	0.5766	0.8200	0.0415	0.0790
	MAML	0.5674	0.6512	0.0355	0.0673	0.5903	0.8005	0.0806	0.1465	0.5692	0.7345	0.0355	0.0676	0.5770	0.7221	0.0544	0.1011
	Reptile	0.5762	0.6409	0.0829	0.1468	0.5888	0.8260	0.0726	0.1334	0.5697	0.6034	0.0693	0.1242	0.5719	0.6804	0.0409	0.0771
	FeatTrans	0.5735	0.6527	0.0647	0.1177	0.5762	0.6626	0.0577	0.1062	0.5744	0.7288	0.0546	0.1016	0.5775	0.7066	0.0589	0.1087
	GPN	0.2588	0.2458	0.2061	0.2242	0.2754	0.2773	0.2453	0.2603	0.4681	0.6257	0.5354	0.5771	0.4195	0.4867	0.6055	0.5397
	Supervised	0.5531	0.4742	0.1488	0.2265	0.5816	0.6512	0.0877	0.1545	0.5461	0.4508	0.1364	0.2094	0.5791	0.6245	0.0957	0.1659
	ICS-GNN	0.5904	0.6004	0.2016	0.3019	0.5896	0.5995	0.2011	0.3012	0.5968	0.6224	0.2153	0.3199	0.5999	0.6220	0.2168	0.3215
	AQD-GNN	0.5183	0.4622	0.5217	0.4901	0.4821	0.4425	0.7266	0.5501	0.5215	0.4573	0.5105	0.4824	0.5069	0.4619	0.7220	0.5633
	CGNP-IP	0.5172	0.4716	0.8118	0.5966	0.5520	0.4915	0.7889	0.6057	0.5699	0.5076	0.8067	0.6231	0.5856	0.5170	0.8083	0.6306
	CGNP-MLP	0.5079	0.4649	0.7870	0.5845	0.5642	0.5003	0.7371	0.5960	0.5365	0.4806	0.6397	0.5489	0.5847	0.5194	0.6841	0.5905
	CGNP-GNN	0.4699	0.4496	0.9161	0.6032	0.4649	0.4449	0.9205	0.5998	0.4938	0.4548	0.7464	0.5652	0.5520	0.4950	0.8399	0.6229
Reddit	ATC	0.6574	0.3566	0.4286	0.3893	0.6582	0.3553	0.4282	0.3883	0.4784	0.9586	0.4108	0.5752	0.4787	0.9572	0.4136	0.5776
	CTC	0.6614	0.3593	0.4202	0.3874	0.6627	0.3583	0.4190	0.3863	0.4713	0.9593	0.4019	0.5664	0.4722	0.9577	0.4054	0.5697
	MAML	0.7450	0.3254	0.0007	0.0014	0.7465	0.3812	0.0010	0.0020	0.4679	0.9864	0.3861	0.5550	0.5017	0.9863	0.4277	0.5967
	Reptile	0.7414	0.3039	0.0116	0.0224	0.7447	0.2874	0.0050	0.0098	0.4051	0.9904	0.3107	0.4730	0.4046	0.9907	0.3121	0.4746
	FeatTrans	0.7327	0.2972	0.0361	0.0644	0.7393	0.3224	0.0261	0.0484	0.2784	0.9369	0.1719	0.2906	0.2345	0.8634	0.1328	0.2302
	GPN	0.4731	0.2285	0.5556	0.3238	0.5051	0.2253	0.5379	0.3175	0.6708	0.9891	0.6749	0.8024	0.6622	0.9871	0.6675	0.7965
	Supervised	0.7079	0.2677	0.0845	0.1284	0.7288	0.2546	0.0364	0.0637	0.5834	0.9736	0.5296	0.6860	0.5536	0.9827	0.4907	0.6545
	ICS-GNN	0.6949	0.3331	0.1959	0.2467	0.6975	0.3361	0.1990	0.2500	0.2748	0.9460	0.1652	0.2813	0.2725	0.9499	0.1652	0.2815
	AQD-GNN	0.5221	0.2514	0.4427	0.3207	0.4141	0.2616	0.7199	0.3837	0.6476	0.8851	0.6772	0.7673	0.7830	0.9139	0.8250	0.8672
	CGNP-IP	0.2557	0.2549	0.9991	0.4062	0.2543	0.2534	0.9983	0.4042	0.7885	0.9264	0.8184	0.8691	0.8482	0.9110	0.9122	0.9116
	CGNP-MLP	0.2566	0.2544	0.9934	0.4051	0.2774	0.2557	0.9694	0.4047	0.8229	0.9397	0.8479	0.8915	0.8697	0.9508	0.8945	0.9218
	CGNP-GNN	0.2548	0.2548	1.0000	0.4061	0.2534	0.2534	1.0000	0.4043	0.8578	0.8578	1.0000	0.9235	0.8584	0.8584	1.0000	0.9238
DBLP	ATC	0.8376	0.8749	0.1752	0.2919	0.8230	0.8916	0.1676	0.2822	0.7527	0.7539	0.0922	0.1643	0.7360	0.7849	0.1038	0.1834
	CTC	0.8365	0.9107	0.1599	0.2720	0.8216	0.9214	0.1534	0.2629	0.7512	0.7711	0.0803	0.1454	0.7345	0.8012	0.0931	0.1668
	MAML	0.8161	0.6395	0.0864	0.1522	0.8029	0.6545	0.1065	0.1832	0.7383	0.5337	0.0581	0.1047	0.7201	0.5713	0.0776	0.1366
	Reptile	0.8106	0.5135	0.1704	0.2559	0.7993	0.5833	0.1162	0.1938	0.7208	0.3890	0.1033	0.1632	0.7184	0.5508	0.0741	0.1306
	FeatTrans	0.8194	0.6339	0.1296	0.2152	0.8057	0.6600	0.1315	0.2193	0.7417	0.5736	0.0796	0.1397	0.7238	0.6301	0.0789	0.1402
	GPN	0.1819	0.0120	0.4528	0.0235	0.1790	0.0748	0.6846	0.1349	0.4017	0.2292	0.5911	0.3303	0.3581	0.2408	0.3988	0.3003
	Supervised	0.7312	0.2142	0.1523	0.1780	0.7773	0.3987	0.1438	0.2113	0.6805	0.3075	0.1692	0.2183	0.7015	0.4255	0.1307	0.2000
	ICS-GNN	0.7997	0.4662	0.3571	0.4044	0.7911	0.5030	0.3519	0.4141	0.7366	0.4978	0.2304	0.3150	0.7290	0.5414	0.2373	0.3299
	AQD-GNN	0.6129	0.2257	0.4220	0.2941	0.5615	0.2705	0.6556	0.3830	0.5421	0.2990	0.5737	0.3931	0.4567	0.2994	0.6992	0.4192
	CGNP-IP	0.3951	0.2206	0.8548	0.3507	0.5320	0.2829	0.8175	0.4203	0.3988	0.2779	0.8288	0.4162	0.5166	0.3404	0.7720	0.4725
	CGNP-MLP	0.4926	0.2317	0.7147	0.3499	0.5203	0.2487	0.6488	0.3596	0.4437	0.2814	0.7415	0.4080	0.5652	0.3632	0.7304	0.4851
	CGNP-GNN	0.4262	0.2223	0.8018	0.3481	0.4535	0.2463	0.7928	0.3759	0.3999	0.2682	0.7644	0.3971	0.3777	0.2894	0.8377	0.4302

outperforms all the baselines in most cases. The F1 of CGNP succeeds all the baselines 0.28 on average. The superiority of CGNP is reflected in improving the recall significantly, while keeping relatively high accuracy and precision. In the testing, we observe that the optimization-based approaches, e.g., MAML, Reptile, predict almost all the nodes as the negative samples. These approaches are sensitive to the imbalanced label distribution, leading to a higher accuracy but low recall. That indicates accuracy is not a suitable metric to evaluate the overall performance for CS task, because most nodes are in the negative class. A model is easy to achieve high accuracy as long as it predicts more nodes as the negative samples. ICS-GNN performs best in some cases (e.g., DBLP and Facebook), as it is a query-specific model and uses the ground-truth of the test query nodes additionally. The naive approaches like FeatTrans even fail to search the community in most cases due to their low F1 score. As for ML/DL-based methods, they get comparable scores with naive approaches. Since these methods are trained from scratch for each new test task or each new query, ML/DL-based methods can utilize task-specific knowledge. However, our approach gets higher score than these methods. It indicates that our method is an efficient meta-learning approach, which can learn prior knowledge from different tasks. The prior knowledge is beneficial for predicting the community. CGNP is the most robust learner due to its metric-based learning strategy, and this property is similar to KNN and GP, which fully validates the effectiveness of CGNP for small data.

TABLE III: Performance on MGOD and MGDD Tasks

Dataset	Task config.	1-shot				5-shot			
	Methods	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
Facebook	ATC	0.5564	0.2595	0.6305	0.3677	0.5592	0.2611	0.6464	0.3720
	ACQ	0.3625	0.2190	0.8248	0.3461	0.4109	0.2266	0.7944	0.3526
	CTC	0.8518	0.8734	0.3224	0.4710	0.8540	0.8904	0.3159	0.4664
	MAML	0.6050	0.2319	0.1692	0.1956	0.6806	0.4091	0.2687	0.3244
	Reptile	0.6356	0.3049	0.2215	0.2566	0.6680	0.4251	0.4642	0.4438
	FeatTrans	0.6105	0.2462	0.1804	0.2082	0.5867	0.2936	0.3192	0.3059
	GPN	0.1549	0.0648	0.1289	0.0863	0.0938	0.0469	0.0967	0.0631
	Supervised	0.6291	0.2343	0.1350	0.1713	0.6073	0.3421	0.4079	0.3721
	ICS-GNN	0.7606	0.5722	0.5598	0.5659	0.7574	0.5906	0.5516	0.5704
	AQD-GNN	0.6779	0.3548	0.1644	0.2247	0.4239	0.3112	0.8396	0.4540
	CGNP-IP	0.3727	0.3107	0.9925	0.4733	0.5121	0.3666	0.9756	0.5329
	CGNP-MLP	0.4161	0.3203	0.9418	0.4781	0.5659	0.3860	0.8832	0.5372
	CGNP-GNN	0.3077	0.2888	0.9835	0.4465	0.6029	0.4118	0.9145	0.5678
Cite2Cora	ATC	0.5779	0.8191	0.1885	0.3064	0.5783	0.8154	0.1934	0.3127
	CTC	0.5166	0.8714	0.0269	0.0523	0.5156	0.8692	0.0271	0.0526
	MAML	0.5042	0.4986	0.3630	0.4202	0.5334	0.5544	0.3008	0.3900
	Reptile	0.5202	0.5219	0.3620	0.4275	0.5658	0.6066	0.3538	0.4469
	FeatTrans	0.5289	0.5529	0.2498	0.3442	0.5122	0.5464	0.0960	0.1632
	GPN	0.2521	0.1716	0.3800	0.2364	0.1766	0.1656	0.3524	0.2254
	Supervised	0.5446	0.5537	0.4099	0.4711	0.6066	0.6206	0.5320	0.5729
	ICS-GNN	0.5532	0.6642	0.1923	0.2982	0.5538	0.6677	0.1932	0.2996
	AQD-GNN	0.5145	0.5040	0.5685	0.5343	0.4652	0.4626	0.6365	0.5358
	CGNP-IP	0.5351	0.5177	0.8822	0.6525	0.5280	0.5134	0.9241	0.6601
	CGNP-MLP	0.5397	0.5207	0.8781	0.6537	0.5476	0.5267	0.8654	0.6548
	CGNP-GNN	0.5367	0.5179	0.9181	0.6623	0.5456	0.5191	0.8532	0.6455

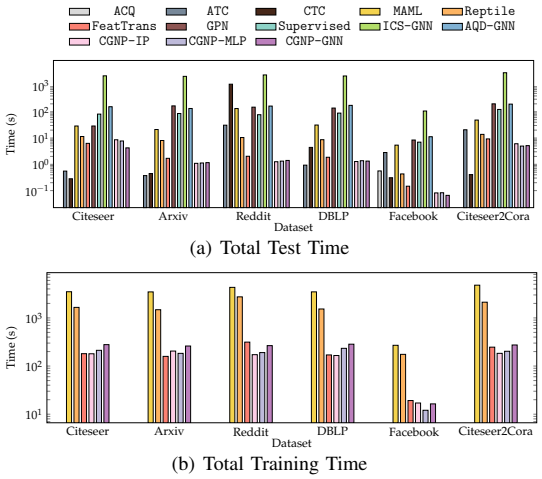
Table III shows the performance for tasks of multiple graphs. The tasks of multiple graphs are harder than that of the single graph, and the tasks across domains are even harder. The F1 of CGNP surpasses the F1 of all the baselines 0.25 by average. The CGNP variants dominate the top two best models on Cite2Cora while it is overwhelmed by ICS-GNN on Facebook. This demonstrates that CGNP can effectively learn prior knowledge from only a few data of one graph and adapt to other graphs even from different domains, and the learned prior is indeed helpful. In fact, transferring the prior of a shared node embedding function for clustering, as what CGNP does, is much easier than transferring model parameters, as MAML, Reptile and FeatTrans do.

CGNP with different decoders may bring different performance to the result. The difference between them is subtle i.e. less than 5%. Both MAML and Reptile perform worse than CGNP in general. The graph algorithms ATC, ACQ and CTC also fail to outperform learning-based approaches due to their low recall. It is worth to mentioning that ACQ fails to return the results for Cite2Cora and Citeseer in 12 hours, since ACQ needs to enumerate all the sets of attributes that are shared by the query node and candidates. In addition, ACQ relies on the node attributes and it cannot support graphs without node attributes, such as Arxiv, DBLP and Reddit.

C. Efficiency

We compare the efficiency of CGNP and the baselines regarding the test/training time. Fig. 3(a) presents the total test time. Regarding the prediction efficiency, our CGNP is the best learning-based approach and the second-best among all the approaches, which is over one order of magnitude faster than ATC, ACQ, MAML, Reptile, GPN, Supervised, ICS-GNN and AQD-GNN, and slightly faster than FeatTrans. For one test task, MAML, Reptile and FeatTrans apply the backward propagation algorithm to update the parameters online, and Supervised and AQD-GNN train the parameters from scratch. ICS-GNN needs to train a model for each query node on-the-fly. GPN not only needs to apply the backward propagation algorithm to update parameters but also has to compute the distance between each node and prototypes. ACQ needs to enumerate all the sets of attributes shared by the query node and candidates, so that it fails to return the results for Cite2Cora and Citeseer in 12 hours. For CTC, the intermediate candidate communities of Reddit and Facebook are large, it takes much longer time to compute the diameter and maintain the k-truss structure.

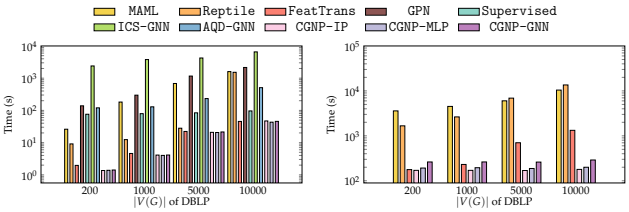
Fig. 3(b) shows the meta training time of the learning-based approaches on the training task set, where all the models are trained by the same epoch of 200. Note that ATC, ACQ, CTC, GPN, Supervised, ICS-GNN and AQD-GNN do not involve this meta training stage. Our CGNP is one order of magnitude faster than MAML and Reptile and its training efficiency is close to the simplest transfer strategy, FeatTrans, MAML and Reptile are quite time-consuming due to their two-level optimization paradigm. For the three CGNP variants, due to different model complexities, training CGNP-GNN is



(a) Total Test Time

(b) Total Training Time

Fig. 3: Training & Test Time (s)



(a) Total Test Time (b) Total Training Time

Fig. 4: Scalability of Training & Test (s)

slightly slower than that of CGNP-MLP, which is further slightly slower than that of CGNP-IP. We observe that these differences are negligible in the testing stage in Fig. 3(a).

Scalability Test. We explore the scalability of the learning-based approaches. Fig. 4 shows the GPU training time and test time of our CGNP and 6 ML baselines, as the number of nodes of graph in each task increases from $1,000$ to $10,000$. All the methods can scale to graphs of $10,000$ nodes in the limited 16GB GPU memory. The test time of CGNP costs the least time than other baselines in all the sizes. Only FeatTrans spends close time to CGNP. Other methods like MAML, Reptile, GPN, AQD-GNN show similar results as shown in Fig. 3(a) due to their training strategy. While the training time of CGNP does not increase significantly with the size increasing. And it is one or two orders of magnitude faster than other methods in large graph.

D. Effect of the ground-truth number

We further evaluate how the number of ground-truth samples influences the performance of the learning-based approaches. For each query node q in the support set, we vary the number of positive/negative samples, i.e., $|l_q^+|/|l_q^-|$, from 2%/10% to 20%/100% of the total number of the nodes. Fig. 5 shows the F1-score of the 3 CGNP variants and 7 ML baselines on the 6 different tasks in 1-shot scenario. In Fig. 5, CGNP variants surpass the ML baselines by 30% on average, particularly under the circumstances of the scarce ground-truth. For small training samples, Supervised suffers from

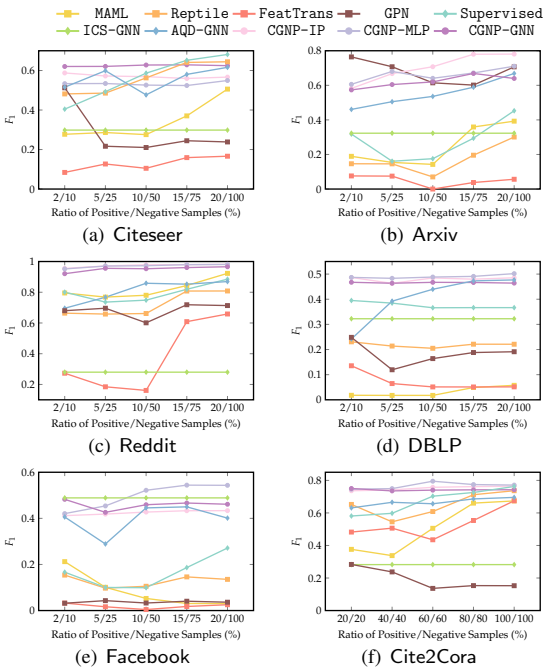


Fig. 5: F1 under Different Ratios of Ground-truth

severe over-fitting, and FeatTrans and GPN also face a high risk of over-fitting in their adaptation step. As the number of ground-truth increases, the performance of GPN would degrade, since more samples may blur the representation of prototypes. The performance of MAML, Reptile and AQD-GNN increase in general with the increasing number of ground truth. In addition, Supervised would overtake CGNP as shown in Fig. 5(a), when the number of ground-truth is at a high level. Given sufficient training data, a task-specific Supervised model can better adapt to the task, compared with a meta model. The F1 of ICS-GNN tends to be stable on varying ratios of samples. We conjecture the reason could be its hyper-parameter, the community size to search, determines the final results in the post-processing stage. Furthermore, we find that the performance of CGNP is robust to the number of ground-truth. That is in accordance with the nature of metric-based learning, where only a few training samples can achieve high performance for KNN and kernel learning.

E. Ablation Study

In this section, we conduct ablation studies to investigate the effect of different options for the GNN layer and the commutative operation on the performance of CGNP. CGNP-MLP, the CGNP with an GNN decoder, is tested as the base CGNP model and all the model variants are trained by the same hyper-parameters. These model variants are tested on the 5-shot Citeseer SGSC, Arxiv SGSC, Reddit SGDC, DBLP SGDC, Facebook MGOD and Cite2Cora MGDD tasks, respectively.

GNN Layer. We adopt three popular GNN, *GCN* [57], *GAT* [58] and *GraphSAGE* [59] as the encoder, where the commutative operation is fixed to the average pooling. Table IV lists the performance of the CGNP-GNN variants on the 2 tasks.

TABLE IV: Performance with Different Layers and Com. Op.

Dataset	Layer	Acc	Pre	Rec	F1	⊕	Acc	Pre	Rec	F1
Citeseer	GCN	0.5001	0.4601	0.7779	0.5782	Att.	0.5956	0.6437	0.6893	0.6657
	GAT	0.5520	0.4950	0.8399	0.6229	Sum	0.6154	0.6526	0.7306	0.6894
	SAGE	0.6348	0.5555	0.8553	0.6736	Ave.	0.6158	0.6513	0.7367	0.6914
Arxiv	GCN	0.4540	0.4388	0.9076	0.5916	Att.	0.4816	0.4526	0.9080	0.6041
	GAT	0.4649	0.4449	0.9205	0.5998	Sum	0.4696	0.4405	0.8044	0.5692
	SAGE	0.6035	0.5305	0.7800	0.6315	Ave.	0.4649	0.4449	0.9205	0.5998
Reddit	GCN	0.8006	0.8596	0.9175	0.8876	Att.	0.8006	0.8006	1.0000	0.8893
	GAT	0.8584	0.8584	1.0000	0.9238	Sum	0.7122	0.7122	1.0000	0.8319
	SAGE	0.9335	0.9553	0.9679	0.9615	Ave.	0.8584	0.8584	1.0000	0.9238
DBLP	GCN	0.3189	0.2829	0.9308	0.4339	Att.	0.3761	0.2866	0.8223	0.4251
	GAT	0.3777	0.2894	0.8377	0.4302	Sum	0.3742	0.2896	0.8472	0.4316
	SAGE	0.5480	0.3446	0.6783	0.4570	Ave.	0.3777	0.2894	0.8377	0.4302
Facebook	GCN	0.3082	0.2880	0.9676	0.4438	Att.	0.5547	0.3795	0.8826	0.5307
	GAT	0.6029	0.4118	0.9145	0.5678	Sum	0.5427	0.3762	0.9156	0.5333
	SAGE	0.4361	0.3143	0.8263	0.4554	Ave.	0.6029	0.4118	0.9145	0.5678
Cite2Cora	GCN	0.5427	0.5179	0.8224	0.6356	Att.	0.5626	0.5310	0.8390	0.6504
	GAT	0.5456	0.5191	0.8532	0.6455	Sum	0.5341	0.5113	0.8898	0.6494
	SAGE	0.5591	0.5306	0.7867	0.6337	Ave.	0.5456	0.5191	0.8532	0.6455

In general, the *GAT* encoder consistently outperforms *GCN* encoders. This is because *GAT* aggregates the node representation weighted by learnable weights via self-attention, where the importance of each neighbor are considered regarding its local structure, possible features and positive/negative labels. The higher F1 of *GAT* demonstrates the attention mechanism can also contribute to improving the performance of CGNP in the encoder part. *GraphSAGE* encoder gets highest F1 scores in SGDC and SGSC tasks. This is because *GraphSAGE* uses a generalized aggregation function and this mechanism is beneficial for encoder of CGNP.

Commutative Operation. We adopt the sum, average pooling and self-attention, introduced in section VI as the commutative operation big $\oplus$ of CGNP-GNN, by fixing *GAT* as the encoder GNN. Table IV shows the corresponding performance of the 3 model variants. In different tasks, the performance of three commutative operations is different. However, the differences between the three variants are relatively slight. We speculate that different tasks, graphs or ground-truth distributions may benefit from different commutative operations, and the effect of the type of commutative operation is not as remarkable as that of the GNN encoder.

VIII. Conclusion

In this paper, we study leveraging ML/DL approaches for community search (CS), under the circumstance that the training data is scarce. We propose a metric-based meta-learning framework, Conditional Graph Neural Process (CGNP) to learn a meta model to capture the prior knowledge of CS. The meta model is adapted to a new task swiftly to make predictions of the community membership, where a task is a graph with only a few given ground-truth. To the best of our knowledge, CGNP is the first meta-learning model for CS that utilizes the generalization ability of neural networks to the greatest extent. Compared with algorithmic approaches, CGNP supports flexible community structures learned from the data. Compared with general meta-learning algorithms, CGNP further exploits the characteristic of CS. Our extensive experiments demonstrate that CGNP outperforms the two lines of approaches significantly regarding accuracy and efficiency.

Acknowledgment

This work was supported by the Research Grants Council of Hong Kong, China, No. 14203618, No. 14202919 and No. 14205520.

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," *VLDB J.*, vol. 29, no. 1, pp. 353–392, 2020.

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

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

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

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

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

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

[9] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based densest clique percolation community search in networks (extended abstract)," in *Proc. ICDE*. IEEE, 2019, pp. 2161–2162.

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

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

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

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

[14] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami, "Conditional neural processes," in *Proc. ICML*, vol. 80. PMLR, 2018, pp. 1690–1699.

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

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

[17] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin, "Efficient computing of radius-bounded k-cores," in *Proc. ICDE*. IEEE Computer Society, 2018, pp. 233–244.

[18] R. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, "Persistent community search in temporal networks," in *Proc. ICDE*. IEEE Computer Society, 2018, pp. 797–808.

[19] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. NeurIPS*, 2018, pp. 537–546.

[20] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. NIPS*, 2017, pp. 6348–6358.

[21] J. Bai and P. Zhao, "Tagsim: Type-aware graph similarity learning and computation," *Proc. VLDB Endow.*, vol. 15, no. 2, pp. 335–347, 2021.

[22] Z. Qin, Y. Bai, and Y. Sun, "Ghashing: Semantic graph hashing for approximate similarity search in graph databases," in *Proc. KDD*. ACM, 2020, pp. 2062–2072.

[23] Y. Bai, D. Xu, Y. Sun, and W. Wang, "Glsearch: Maximum common subgraph detection via learning to search," in *Proc. ICML*, ser. Proc. of Machine Learning Research, vol. 139. PMLR, 2021, pp. 588–598.

[24] C. T. Duong, D. Hoang, H. Yin, M. Weidlich, Q. V. H. Nguyen, and K. Aberer, "Efficient streaming subgraph isomorphism with graph neural networks," *Proc. VLDB Endow.*, vol. 14, no. 5, pp. 730–742, 2021.

[25] H. Wang, Y. Zhang, L. Qin, W. Wang, W. Zhang, and X. Lin, "Reinforcement learning based query vertex ordering model for subgraph matching," *CoRR*, vol. abs/2201.11251, 2022.

[26] R. Ying, Z. Lou, J. You, C. Wen, A. Canedo, and J. Leskovec, "Neural subgraph matching," *CoRR*, vol. abs/2007.03092, 2020.

[27] K. Zhao, J. X. Yu, H. Zhang, Q. Li, and Y. Rong, "A learned sketch for subgraph counting," in *Proc. SIGMOD*. ACM, 2021, pp. 2142–2155.

[28] X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang, "Neural subgraph isomorphism counting," in *Proc. KDD*. ACM, 2020, pp. 1959–1969.

[29] K. Zhao, J. X. Yu, Q. Li, H. Zhang, and Y. Rong, "Learned sketch for subgraph counting: a holistic approach," *The VLDB Journal*, pp. 1–26, 2023.

[30] J. Qi, W. Wang, R. Zhang, and Z. Zhao, "A learning based approach to predict shortest-path distances," in *Proc. EDBT*. OpenProceedings.org, 2020, pp. 367–370.

[31] K. Zhao, Z. Zhang, Y. Rong, J. X. Yu, and J. Huang, "Finding critical users in social communities via graph convolutions," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[32] X. Su, S. Xue, F. Liu, J. Wu, J. Yang, C. Zhou, W. Hu, C. Paris, S. Nepal, D. Jin, Q. Z. Sheng, and P. S. Yu, "A comprehensive survey on community detection with deep learning," *CoRR*, vol. abs/2105.12584, 2021.

[33] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014.

[34] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proc. ICLR*, 2015.

[35] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 1842–1850.

[36] T. Munkhdalai and H. Yu, "Meta networks," in *Proc.ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 2554–2563.

[37] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1126–1135.

[38] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. ICLR*, 2017.

[39] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[40] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," in *Proc. NIPS*, 2017, pp. 4077–4087.

[41] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. CVPR*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 1199–1208.

[42] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. NIPS*, 2016, pp. 3630–3638.

[43] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng, "Meta-gnn: On few-shot node classification in graph meta-learning," in *Proc. CIKM*. ACM, 2019, pp. 2357–2360.

[44] K. Huang and M. Zitnik, "Graph meta learning via local subgraphs," in *Proc. NeurIPS*, 2020.

[45] A. J. Bose, A. Jain, P. Molino, and W. L. Hamilton, "Meta-graph: Few shot link prediction via meta learning," *CoRR*, vol. abs/1912.09867, 2019. [Online]. Available: http://arxiv.org/abs/1912.09867

[46] J. Chauhan, D. Nathani, and M. Kaul, "Few-shot learning on graphs via super-classes based on graph spectral measures," in *Proc. ICLR*, 2020.

[47] N. Ma, J. Bu, J. Yang, Z. Zhang, C. Yao, Z. Yu, S. Zhou, and X. Yan, "Adaptive-step graph meta-learner for few-shot graph classification," in *Proc. CIKM*. ACM, 2020, pp. 1055–1064.

[48] F. Zhou, C. Cao, G. Trajcevski, K. Zhang, T. Zhong, and J. Geng, "Fast network alignment via graph meta-learning," in *Proc. INFOCOM*. IEEE, 2020, pp. 686–695.

[49] W. Xiong, M. Yu, S. Chang, X. Guo, and W. Y. Wang, "One-shot relational learning for knowledge graphs," in *Proc. EMNLP*. Association for Computational Linguistics, 2018, pp. 1980–1990.

[50] D. Mandal, S. Medya, B. Uzzi, and C. Aggarwal, "Meta-learning with graph neural networks: Methods and applications," *CoRR*, vol. abs/2103.00137, 2021.

[51] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endow.*, vol. 9, no. 4, pp. 276–287, 2015.
[52] A. Antoniou, H. Edwards, and A. J. Storkey, "How to train your MAML," in *Proc. ICLR*, 2019.
[53] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.
[54] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, S. M. A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, "Attentive neural processes," in *Proc. ICLR*, 2019.
[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS*, 2017, pp. 5998–6008.
[56] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. ICDM*. IEEE Computer Society, 2012, pp. 745–754.
[57] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017.
[58] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. ICLR*, 2018.
[59] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, pp. 1024–1034.
[60] "Pytorch," https://github.com/pytorch/pytorch.
[61] "Pytorch Geometric," https://github.com/rusty1s/pytorch_geometric.