Data Science and Engineering (2023) 8:292–305
https://doi.org/10.1007/s41019-023-00223-w

RESEARCH PAPER

Learning with Small Data: Subgraph Counting Queries

Kangfei Zhao[1] · Zongyan He[2] · Jeffrey Xu Yu[2] · Yu Rong[3]

Received: 19 May 2023 / Revised: 5 July 2023 / Accepted: 7 August 2023 / Published online: 2 September 2023

Abstract
Deep Learning (DL) has been widely used in many applications, and its success is achieved with large training data. A key issue is how to provide a DL solution when there is no large training data to learn initially. In this paper, we explore a meta-learning approach for a specific problem, subgraph isomorphism counting, which is a fundamental problem in graph analysis to count the number of a given pattern graph, p, in a data graph, g, that matches p. There are various data graphs and pattern graphs. A subgraph isomorphism counting query is specified by a pair, (g, p). This problem is NP-hard and needs large training data to learn by DL in nature. We design a Gaussian Process (GP) model which combines Graph Neural Network with Bayesian nonparametric, and we train the GP by a meta-learning algorithm on a small set of training data. By meta-learning, we can obtain a generalized meta-model to better encode the information of data and pattern graphs and capture the prior of small tasks. With the meta-model learned, we handle a collection of pairs (g, p), as a task, where some pairs may be associated with the ground-truth, and some pairs are the queries to answer. There are two cases. One is there are some with ground-truth (few-shot), and one is there is none with ground-truth (zero-shot). We provide our solutions for both. In particular, for zero-shot, we propose a new data-driven approach to predict the count values. Note that zero-shot learning for our regression tasks is difficult, and there is no hands-on solution in the literature. We conducted extensive experimental studies to confirm that our approach is robust to model degeneration on small training data, and our meta-model can fast adapt to new queries by few-shot and zero-shot learning.

Keywords Subgraph isomorphism · Subgraph counting · Meta-learning

1 Introduction

Deep learning (DL) has achieved great success in database systems to support query optimization [31], index recommendation [12], view materialization [27], cardinality estimation [13, 25], and subgraph counting [28, 65]. The two main keys that lead to the success of deploying DL effectively to deal with real problems are (1) to learn an advanced model that meets the problem, and (2) to learn with large training data. Almost all the works focus on modeling techniques assuming that it is possible to collect enough training data to learn. Take query optimization as an example, training a feed-forward neural network consumes 20,000 unique queries over a fixed database schema [13]. A natural question that arises is what a system can do if there are only a few training data to learn a model that can be effectively used. The solution rules out learning a model until the training dataset is large. This problem is important because a system needs to support various needs from time to time, and it is the last thing that the system responses negative if there is no sufficient training data to learn. To address this issue, it requires an approach by meta-learning. That is to learn a model and refine the model with limited or even no training data if any from time to time. We will discuss it with a specific problem in database, as it is difficult to come up with a general solution to deal with the requirement of sufficient training at this stage.

✉ Kangfei Zhao
zkf1105@gmail.com

Zongyan He
zyhe@se.cuhk.edu.hk

Jeffrey Xu Yu
yu@se.cuhk.edu.hk

Yu Rong
yu.rong@hotmail.com

[1] Beijing Institute of Technology, Beijing, China

[2] The Chinese University of Hong Kong, Hong Kong, China

[3] Tencent AI Lab, Shenzhen, China

As a specific problem, in this paper, we study a subgraph isomorphism counting query that has been widely used in many applications in bioinformatics, chemoinformatics, and social network analysis [28]. Such a query is specified by a pair of (g, p), where g is a data graph and p is a pattern graph, and is to find the number of matches of p in g. This problem itself is NP-hard [8] and is difficult to learn. In general, over a set of data graphs, $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, there is a set of pattern graphs $\mathcal{P} = \cup_{1 \le i \le n} \mathcal{P}(g_i)$ where $\mathcal{P}(g_i)$ is a set of pattern graphs associated with g_i. In [28], a DL model is trained by feeding a large number of training pairs (g, p), aiming to be well generalized on unseen test pairs (g^*, p^*). The DL model consumes hundreds of thousands of pairs, and for each graph g_i, its $\mathcal{P}(g_i)$ is large enough. The issue that arises is how to collect a sufficiently large training dataset to synthesize comprehensive features to be learned. In real applications, the data graph may come from different domains, and the pattern graphs may be diverse regarding the sizes, node/edge labels, and structures. For example, in computational biology, the data graphs could be the molecules in many drug banks, which have different scaffolds. The pattern graphs are functional groups which are a large combination of different numbers and types of atoms/bonds.

Hence, it is impractical to learn a model for given $\mathcal{G}$ and $\mathcal{Q}$ with limited training pairs, and it is impractical to learn a model for each data graph g_i with its pattern graphs $\mathcal{P}(g_i)$. To deal with such a dilemma, we construct a meta-model for given $\mathcal{G}$ and $\mathcal{P}$, which learns the prior knowledge of subgraph counting across multiple tasks. Consider an encountered task, i.e., a collection of pairs, where some pairs may be with ground-truth (e.g., the exact number of matchings), and some pairs need to answer. With some possible pairs with ground-truth (known as shots), the meta-model can be swiftly adapted to deal with the other pairs in the task that need to answer. Inspired by deep kernel learning and deep kernel transfer [39, 57], we devise a new meta-model that warps a graph neural network (GNN) as a special Gaussian Process (GP). For one thing, the graph neural network preserves the powerful modeling capability of deep learning for subgraph counting. For the other thing, Bayesian nonparametric, inherited by GP, enables learning from scratch over small samples with a distribution-free assumption, and the prior of structural subgraph counting tasks are captured via optimizing the prior of the GP. Furthermore, as there may be no pair in a task with ground-truth (zero-shot), we adapt the kernel-based meta-learning algorithm to support zero-shot cases in a data-driven fashion. It is worth mentioning that zero-shot learning for regression tasks is difficult, and there is no hands-on solution in the literature. The contributions of this paper are summarized as follows:

- We study a specific problem of subgraph isomorphism counting in a paradigm of meta-learning to address the initial small training data issue in DL. We propose a Gaussian Process model, called RGIN-GP, that combines graph neural network and kernel method, aiming to learn over small training data.
- We employ a meta-learning algorithm to train a meta-RGIN-GP model over a small set of training data. The small training dataset is divided into tasks, where a task is a collection of (g, p) pairs, for the purpose of predicting a new counting task that may share the similar task structure.
- We provide solutions for both few-shot and zero-shot cases to deal with a new subgraph counting task. In particular, for zero-shot, we propose a new data-driven approach to predict the count values for a new task without any ground-truth.
- We conduct extensive experiments for few-shot and zero-shot learning, using real and synthetic graph datasets in different scenarios. The experimental results verify that the meta-learned RGIN-GP outperforms the supervised learned neural network counterparts by small training data and is effective to adapt to new tasks by few-shot/zero-shot learning.

Roadmap: Section 2 gives the problem statement and a neural network framework for subgraph counting. In Sect. 3, we introduce the RGIN-GP model, the meta training and testing algorithms, and the feature encoding. Section 4 reports the experimental results. Finally, we review the related works in Sect. 5 and conclude the paper in Sect. 6.

2 Preliminaries

We model both data graph g and pattern graph p as a labeled undirected graph as a tuple $G = (V, E, L_V, L_E, \Sigma_V, \Sigma_E)$. Here, V is a set of nodes, E is a set of undirected edges, and L_V (L_E) is a mapping function that maps a node $u \in V$ (edge $e \in E$) to a node label (edge label) in Σ_V (Σ_E). We denote neighbors of node u in G as $N(u) = \{v | (u, v) \in E\}$.

Subgraph Isomorphism: Given a data graph $g = (V_g, E_g, L_V, L_E, \Sigma_V, \Sigma_E)$ and a pattern graph $p = (V_p, E_p, L_V, L_E, \Sigma_V, \Sigma_E)$, subgraph isomorphism p to g is an *injective* function f: $V_p \mapsto V_g$ such that (1) for every $u \in V_p$, $L_V(u) = L_V(f(u))$, (2) for every $(u, v) \in E_p$, $(f(u), f(v)) \in E_g$, and (3) for every $e = (u, v) \in E_p$ and $e' = (f(u), f(v)) \in E_g$, $L_E(e) = L_E(e')$. The injective function f imposes the constrain that $f(u) \neq f(v)$ for any pair of u and v in V_p if $u \neq v$. A subgraph isomorphism function f of p induces a subgraph $g_f = (V_f, E_f, L_V, L_E, \Sigma_V, \Sigma_E)$ in g, where V_f is the set of nodes by $f(u)$ for every u in V_p, and E_f is the set of edges by $(f(u), f(v))$ for every edge (u, v) in E_p. We say g_f is a subgraph isomorphism matching of p to g. Finding the subgraph isomorphism matching for given p and g is

computationally hard as the decision problem of subgraph isomorphism is NP-complete [8].

Subgraph Isomorphism Counting Query: Given a pair of data graph g and a pattern graph p, a subgraph isomorphism counting query is to find the total number of subgraph isomorphism matchings of p to g, denoted as $c(g, p)$. Here, a node in p is allowed to be unlabeled. When a node in p is unlabeled, a special label, interpreted as **any**, is assigned.

A graph database is a set of small/medium sized graphs, $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, with a set of pattern graphs $\mathcal{P} = \cup_{1 \le i \le n} \mathcal{P}(g_i)$, where $\mathcal{P}(g_i)$ is a set of pattern graphs associated with g_i. For simplicity, we also use $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ to denote the whole possible set of patterns. A subgraph isomorphism counting query is a pair of (g, p) such that $g \in \mathcal{G}$, and $p \in \mathcal{P}(g)$.

A machine learning model can be built from a set of pairs of data graph and pattern graph $X = \{x_1, x_2, \ldots x_{|X|}\}$, where x_i is a pair of data and pattern graphs, $x_i = (g_i, p_i)$, with the corresponding true count denoted as $c(x_i)$ (or $c(g_i, p_i)$) for all $1 \le i \le |X|$. A true count is the exact count of subgraph isomorphism matching of p_i in g_i. The model will take an unseen pair $x^* = (g^*, p^*)$, and estimate its count $\hat{c}(g^*, p^*)$. In estimation, the pair $x^* = (g^*, p^*)$ is not seen in the training set, where either g^* or q^* may appear in the training set. We use the absolute error to evaluate the accuracy of the estimated value.

$$\text{abs- error}(g^*, p^*) = |c(g^*, p^*) - \hat{c}(g^*, p^*)| \tag{1}$$

Note that the model can answer the subgraph isomorphism query, i.e., whether p^* is subgraph isomorphism to g^* by $\hat{c}(g^*, p^*) > 0.5$.

Problem Statement: The problem is first to build a model $\mathcal{M}$ to support subgraph isomorphism counting queries, when the size of the training set is not large enough. Then, in testing $\mathcal{M}$, there are new tasks coming, where a task $\mathcal{T}^*$ is the union of two subsets, $\mathcal{S}^*$ and $\mathcal{Q}^*$, denoted as $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$, where $\mathcal{S}^*$ is a support set with ground-truth such that $\mathcal{S}^* = \{(g_i^*, p_i^*)\}_{i=1}^{l}$, where $c(g_i^*, p_i^*)$ for each i is given, and $\mathcal{Q}^*$ is a query set to be answered each such that $\mathcal{Q}^* = \{(g_j^*, p_j^*)\}_{j=1}^{k}$. Note that $\mathcal{S}^*$ may be empty, i.e., $l = 0$ (called zero-shot), and is small in size when it is non-empty, i.e., $l < k$ (called few-shot). The problem is how to enhance $\mathcal{M}$ to answer queries in $\mathcal{Q}^*$ on-demand with the assistance of $\mathcal{S}^*$ which may be empty.

2.1 A Model Learned with Large Data

Recently, a learning framework has been proposed for the subgraph isomorphism counting of a pair of data and pattern graphs in [28]. This neural network framework is composed of graph representation layers, interaction layers, and Multilayer perceptron (MLP), to learn $\mathcal{M}$ with a large training data X. The graph representation layers take the labeled data and pattern graph as input and generate vector representation for the data graph and pattern graph, respectively. The interaction layer combines the two representations into one representation and the MLP finally outputs the estimated count. [28] explores different options for the graph representation layers (i.e., CNN [24], LSTM [20], Transformer-XL [10], Graph Neural Networks (GNN) [47, 59]) and for the interaction layers (i.e., sum/mean/max pooling, multi-head attention [52] and dynamic intermedium attention). Regarding the trade-off between prediction accuracy and efficiency, a GNN variant, Relational Graph Isomorphism Network (RGIN), coupled with sum pooling interaction performs best in [28]. Below, we introduce RGIN which we adopt to build a neural network.

RGIN Graph Representation. The K-layer GNN [18, 53, 59] follows a neighborhood aggregation paradigm to update the representation of a node by aggregating the representations of its neighbors in K iterations. Let $\mathbf{e}_v^{(k)}$ denote the representation of a node v generated in the k-th iteration. In the GNN k-th iteration (layer), for a node v, an aggregate function $f_{\mathcal{A}}^{(k)}(\cdot)$ aggregates the representations of the neighbors of v that are generated in the (k-1)-th iteration as Eq. (2). Then, a combine function $f_{\mathcal{C}}^{(k)}(\cdot)$ updates the representation of v by the aggregated representation $a_v^{(k)}$ and the previous representation $\mathbf{e}_v^{(k-1)}$ itself as Eq. (3). The functions $f_{\mathcal{A}}^{(k)}(\cdot)$ and $f_{\mathcal{C}}^{(k)}(\cdot)$ are neural networks, e.g., linear transformation with nonlinearity and optional Dropout [49] for preventing overfitting.

$$a_v^{(k)} = f_{\mathcal{A}}^{(k)}(\{\mathbf{e}_u^{(k-1)} | u \in N(v)\}) \tag{2}$$

$$\mathbf{e}_v^{(k)} = f_{\mathcal{C}}^{(k)}(\mathbf{e}_v^{(k-1)}, a_v^{(k)}) \tag{3}$$

Take the RGIN layer as an example, for each node v, the aggregate function in Eq. (4) distinguishes its neighbors by the edge label, and aggregates the $|\Sigma_E|$ types of neighbors respectively. $W_l^{(k)}$ is the weight matrix for the neighbors with edge label l in the k-th layer and the $|L_E|$ types aggregation are further summed up. In the combine function of Eq. (5), the aggregated $a_v^{(k)}$ is summed up with the $(k-1)$-layer representation $e_v^{(k-1)}$, which transformed by weigh $W_0^{(k)}$, and finally be transformed by an MLP layer.

$$a_v^{(k)} = \sum_{l \in \Sigma_E} \sum_{u \in N(v), L_E((u,v))=l} W_l^{(k)} \mathbf{e}_u^{(k-1)} \tag{4}$$

$$\mathbf{e}_v^{(k)} = \text{MLP}(W_0^{(k)} \mathbf{e}_v^{(k-1)} + a_v^{(k)}) \tag{5}$$

For the data graph g and pattern graph p, there are two independent K-layer RGIN models that generate the $|V_g| \times d$-dim data graph node embedding and $|V_p| \times d$-dim pattern graph node embedding of the K-th layer, respectively.

Sum Pooling Interaction. After obtaining the data graph and pattern graph representations, the interaction layer is to combine the two representations to one pair-wise representation. The sum pooling interaction sums up the node embedding of the data and pattern graphs, respectively, and simply concatenates the two vectors to a long vector, as shown in Eq. (6).

$$h = \text{Concat}\left(\sum_{v \in V_g} \mathsf{e}_v^{(K)}, \sum_{v' \in V_p} \mathsf{e}_{v'}^{(K)}\right) \tag{6}$$

The concatenated vector h will pass an MLP to generate the prediction $\hat{c}(g, p)$. The experimental study in [28] shows that although the sum pooling is simple, it is easy to train and achieve the approaching accuracy of the dynamic intermedium attention memory interaction (DIAMNet) and is two times faster than DIAMNet. In contrast, complex interaction layer such as DIAMNet and multi-head attention are hard to train and face the risk of overfitting. To train the DL models, [28] adopts the mean-squared-error loss.

3 A Meta-Learning Approach

We explore a meta-learning approach to build a meta-counting model $\mathcal{M}$. Here, the main problem is to answer each pair in $\mathcal{Q}^*$ from a new coming task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$ by $\mathcal{M}$ built with limited training data $X = \{x_1, x_2, \ldots, x_{|X|}\}$, where each $x_i = (g_i, p_i)$ in X is with the ground-truth $c(x_i)$. To capture the prior knowledge by the meta-model that is persisted with task-common parameters, the initial training data, X, is disjointly distributed into multiple training tasks, $\mathcal{D} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$. Here, a task $\mathcal{T}_i$ is also a pair, $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{Q}_i)$. Different from $\mathcal{T}^*$, we have ground-truth for both $\mathcal{S}_i$ and $\mathcal{Q}_i$. A meta-learning algorithm is to learn the knowledge prior over multiple tasks which may exhibit some specific task structure.

Figure 1 presents the overview of the meta training and testing procedures. In training, for each task $\mathcal{T}_i$ with a small number of pairs to train, we use RGIN-GP (RGIN Gaussian Process) to construct a kernel K, which is specified by a deep neural network parameter w and a stationary kernel with hyperparameter θ. As illustrated in (Fig. 1a), a task $\mathcal{T}_i$ is presented with multiple data graphs ($\mathcal{G}_i$) and multiple pattern graphs ($\mathcal{P}_i$), and the meta-model is a Gaussian Process whose parameters are trained by gradient descent task by task. With the kernel built with w and θ, in testing for a new incoming task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$,

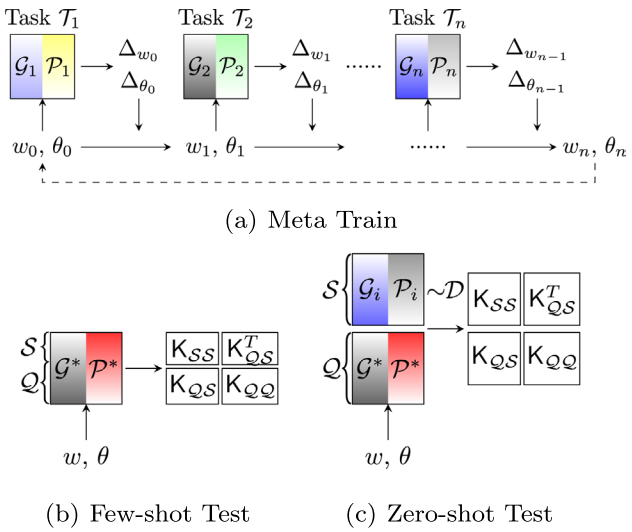

Fig. 1 Meta-model train and test

there are two cases. One is few-shot where $|\mathcal{S}^*|$ is small but is nonzero, and one is zero-shot where $|\mathcal{S}^*| = 0$. For few-shot testing, the meta-model leverages the kernel matrix of the whole task to make a prediction for the query set (Fig. 1b). For zero-shot test, we take a data-driven approach to build the kernel by making use of one task, $\mathcal{T}_i$, drawn from the training data (Fig. 1c). In this work, meta-testing is not to refine the meta-model built with w and θ. The meta-model built remains unaffected. With a new task, $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$, it is to answer queries in $\mathcal{Q}^*$ by Bayesian inference with additional $\mathcal{S}^*$, which we will discuss later.

The approach we take is based on an optimization-based Bayesian meta-learning algorithm, Deep Kernel Transfer [39], for four reasons, where two reasons are from the perspective of training algorithms, and two are from the perspective of model characteristics. From the perspective of training, first, as an optimization-based approach, DKT directly optimizes the task-common parameters by processing one task at a time via stochastic gradient descent, which is effective to learn the task-level prior knowledge. Second, the model can be learned by one-level gradient descent, which is more efficient and stable than classical bi-level meta-learning algorithms, e.g., MAML [15]. From the perspective of the model, third, by leveraging DKL, the proposed GP model, RGIN-GP, is suitable for modeling small data and naturally endorses regression tasks, exploiting the main advantages of GP. Fourth, as a GP model which adapts Bayesian inference for prediction, RGIN-GP can provide robust prediction and principled uncertainty quantification derived from Bayesian method.

In the following, we present how to deploy a kernel for a task, which exploits deep neural network transformation (Sect. 3.1), how to train the kernel and test in the few-shot and zero-shot scenarios (Sect. 3.2), and how to encode the input pairs (Sect. 3.3).

3.1 RGIN Gaussian Process (RGIN-GP)

To learn over a small set of samples, nonparametric modeling is an effective method in Bayesian learning. Deep Kernel Learning (DKL) [45, 57] provides a way to integrate Bayesian nonparametric into deep learning models. It constructs a special GP model which has a conventional stationary kernel function, e.g., the RBF kernel, but the input features space is the embedding space of deep learning models. The hyperparameters of the kernel and the parameters of the deep learning model can be jointly optimized by stochastic gradient descent. Inspired by this, we adopt RGIN with sum pooling layer as the deep learning model that transforms the data and pattern graph into the input vector embedding for a stationary kernel, and the RGIN-GP derives from the deep kernel function constructed as follows. Given an input $x = (g, p)$ as a pair of data and pattern graphs, the deep kernel function K measures the similarity of a pair of inputs x_i, x_j as

$$\mathsf{K}(x_i, x_j; w, \theta) = \mathcal{K}(\mathcal{F}(x_i; w), \mathcal{F}(x_j; w); \theta) \tag{7}$$

Here, $\mathcal{F}(x;w)$ is a nonlinear transformation specified by a deep neural network with parameters w, i.e., the RGIN together with the sum pooling interaction layer. And the function $\mathcal{K}(h_i, h_j;\theta)$ is a stationary kernel function that is invariant to input transformation with the hyperparameter θ, e.g., the RBF kernel. In a nutshell, the neural network $\mathcal{F}(x;w)$ is responsible for learning an effective intermediate representation h to capture the non-stationary and hierarchical features of the input. Then, the kernel $\mathcal{K}(h_i, h_j;\theta)$ discovery stationary structure by an interpretable basis function.

Given n training inputs, $X = \{x_1, \ldots, x_n\}$, the deep kernel K defined in Eq. (7), the model $f(X)$ is a Gaussian Process as Eq. (8) [43] that we call RGIN-GP, where $\mu_X = [\mu]^n$ is an assumed constant mean and $\mathsf{K}_{X,X} = [\mathsf{K}(x_i, x_j; w, \theta)]^{n \times n}$ is the covariance function.

$$f(X) = [f(x_1), \ldots, f(x_n)]^T \sim \mathcal{N}(\mu_X, \mathsf{K}_{X,X}) \tag{8}$$

To make prediction for the testing inputs $X^* = \{x_1^*, \ldots x_m^*\}$, we need to compute the conditional distribution $p(f(X^*)|f(X))$ as the prediction, assuming the output is disturbed by a Gaussian noise $\mathcal{N}(0, \sigma^2)$. It is also proved to be a Gaussian distribution as Eq. (9), where the expectation and covariance of the predictive distribution can be solved in closed form in Eqs. (10), (11).

$$f(X^*)|f(X) \sim \mathcal{N}(\mathbb{E}(\boldsymbol{c}^*), C) \tag{9}$$

$$\mathbb{E}(\boldsymbol{c}^*) = \mu_X + \mathsf{K}_{X,X^*}^T [\mathsf{K}_{X,X} + \sigma^2 \mathcal{I}]^{-1} (\boldsymbol{c} - \mu_X) \tag{10}$$

$$C = \mathsf{K}_{X^*,X^*} - \mathsf{K}_{X,X^*}^T . [\mathsf{K}_{X,X} + \sigma^2 \mathcal{I}]^{-1} \mathsf{K}_{X,X^*} \tag{11}$$

Here, $\boldsymbol{c} = [c(x_i)]^n$ is the ground-truth vector of the input X. The matrices $\mathsf{K}_{X,X} = [\mathsf{K}(x_i, x_j; w, \theta)]^{n \times n}$, $\mathsf{K}_{X,X^*} = [\mathsf{K}(x_i, x_j^*; w, \theta)]^{n \times m}$ and $\mathsf{K}_{X^*,X^*} = [\mathsf{K}(x_i^*, x_j^*; w, \theta)]^{m \times m}$ are the train-train, train-test, test-test kernels, respectively. The expectation $\mathbb{E}(\boldsymbol{c}^*)$ will be treated as the explicit prediction counts $\hat{\boldsymbol{c}}$, and the diagonal element of matrix C in Eq. (11) measures the variance of the prediction. Meanwhile, we can easily compute the δ-confidential interval of $\hat{\boldsymbol{c}}$ as $[\hat{\boldsymbol{c}} - q_\delta diag(C), \hat{\boldsymbol{c}} + q_\delta diag(C)]$, where q_δ is the δ-quantile of $\mathcal{N}(0, 1)$.

The predictive distribution highly depends on the kernel matrix K, which is determined by the neural network weight w and the stationary kernel hyperparameter θ. Training the kernel is to infer these parameters to adapt to the training data. The optimization is conducted by minimizing the negative marginal (log) likelihood of the training data X.

$$\begin{aligned} \mathcal{L}_{\text{mll}} &= -\log p(\boldsymbol{c}|X) = \int p(\boldsymbol{c}|f(X), X) p(f(X)|X) d(f(X)) \\ &\propto \boldsymbol{c}^T [\mathsf{K}_{X,X} + \sigma^2 \mathcal{I}]^{-1} \boldsymbol{c} + \log |\mathsf{K}_{X,X} + \sigma^2 \mathcal{I}| \end{aligned} \tag{12}$$

There is no analytical solution for optimizing the loss of Eq. (12), but the objective is differentiable. To train an RGIN-GP, the neural network weights w and the kernel hyperparameter θ are jointly optimized by stochastic gradient descent. We use the spectral mixtures base kernels [56] as the stationary kernel function $\mathcal{K}$ since the kernel is able to approximate continuous stationary kernels to an arbitrary precision given sufficient number of mixtures.

3.2 Meta-Learning for RGIN-GP

In this section, we discuss how to train an RGIN-GP as a meta-model and test it in the few-shot and zero-shot scenarios. In real applications, new tasks may arrive in different ways, since the data graph may come from different domains, and the pattern graphs may be diverse regarding the sizes, node/edge labels, and structures. In this paper, we firstly explore the following 5 task configurations where a single variable (e.g., a data/pattern graph) is controlled. Here, given a training data X, we have a set of data graphs, $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$, and a set of pattern graphs,

$\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$. The graphs in $\mathcal{G}$ may come from different domains. Instead of specifying a task as $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{Q}_i)$, below, for the task structure configuration, we specify a task $\mathcal{T}_i = (\mathcal{G}_i, \mathcal{P}_i)$, where $\mathcal{G}_i \subset \mathcal{G}$ and $\mathcal{P}_i \subset \mathcal{P}$, and we have the ground-truth for any pair (g, p) for $g \in \mathcal{G}_i$ and $p \in \mathcal{P}_i$.

- Same Graph Tasks (`SameG`): Data-pattern pairs are from a single domain. The data graphs that appear in training tasks will not appear in any testing task. The pattern graphs $\mathcal{P}(g_i)$ that are associated with a data graph g_i will appear together with g_i in a task where g_i appears. Note that a pattern graph p_j may appear in both training and testing pairs.
- Same Pattern Tasks (`SameP`). Data-pattern pairs are from a single domain. The pattern graphs that appear in training tasks will not appear in any testing task. The data graphs g_i will appear in a task together with p_j if $p_j \in \mathcal{P}(g_i)$ appears in the task. Note that a data graph g_i may appear in both training and testing pairs.
- Hybrid Domains with Same Graph Tasks (`HySameG`). Data-pattern pairs are from multiple domains, whereas pairs in one task are from the same domain. For one domain, training and testing tasks follow `SameG`.
- Hybrid Domains with Same Pattern Tasks (`HySameP`). Data-pattern pairs are from multiple domains, whereas pairs in one task are from the same domain. For one domain, training and testing tasks follow `SameP`.
- Random Tasks (`Random`). Data-pattern graph pairs are from a single domain. Pairs are randomly and disjointly distributed in all the tasks in the training and testing task sets. A pair in a training task does not appear in any testing tasks.

Here, `SameG` corresponds to the situations where the pattern graphs are static and there may have new arrival data graphs. `SameP` corresponds to the situations where the data graphs are static and there may have new arrival pattern graphs. `HySameG` and `HySameP` do similar like `SameG` and `SameP` in the same single domain, but are on multiple domains. `Random` is to test different possibilities in a single domain.

Meta-Training: Algorithm 1 presents the meta-training process for RGIN-GP. The algorithm is to learn (w, θ) of the kernel K that minimizes the negative marginal likelihood across all the training tasks. As shown in Fig. 1a, for each gradient step, a task $\mathcal{T}$ is sampled from the training tasks (line 1), then the marginal likelihood $\mathcal{L}_{\text{mll}}$ (Eq. (12)) is computed over all the pairs in the task, i.e., $\mathcal{S} \cup \mathcal{Q}$ (line 4), and the parameters (w, θ) are updated for that task (line 5). The meta-training algorithm is different from training the kernel from scratch, where marginalization of the likelihood is over all data instead of a distinct task. The parameters (w, θ) learned by Algorithm 1 better leverage the structure of the tasks, which are shared across all tasks as the task-common parameters.

Few-shot Testing: Given a testing task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$ where $|\mathcal{S}^*| \neq 0$, the meta-model will adapt to the task based on its support set and the task-common parameters learned. Algorithm 2 shows the steps to compute the predictive distribution of Eq. (9) for the query set $\mathcal{Q}^*$. It is computed by conditioning on the support set $\mathcal{S}^*$ (line 1–2), which analytical solution is given in Eqs. (10)–(11). The Bayesian inference is essential to compute the integral

$$p(c_{\mathcal{Q}^*}|X_{\mathcal{Q}^*}, c_{\mathcal{S}^*}, X_{\mathcal{S}^*}, w, \theta) = \int p(c_{\mathcal{Q}^*}|X_{\mathcal{Q}^*}, \rho_{\mathcal{T}^*}, w) \; p(\rho_{\mathcal{T}^*}|c_{\mathcal{S}^*}, X_{\mathcal{S}^*}) d(\rho_{\mathcal{T}^*}) \tag{13}$$

where w is the learned neural network weights, and $\rho_{\mathcal{T}^*}$ is the task-specific parameters derived from the kernel hyperparameters θ. Equation (13) ensembles all the models with all possible configurations of the task-specific parameters $\rho_{\mathcal{T}^*}$, weighted by the posterior of the parameters $p(\rho_{\mathcal{T}^*}|c_{\mathcal{S}^*}, X_{\mathcal{S}^*})$ given the support set $\mathcal{S}^*$. Equation (13) is known as *Bayesian model average* [58].

Zero-shot Testing: Given a testing task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$ where $|\mathcal{S}^*| = 0$, the meta-model cannot adapt to the task based on its support set. In the literature [55, 60], zero-shot learning is mainly done for classification where the classes are limited. Different from classification, our problem here is for subgraph isomorphism counting by regression. Performing zero-shot testing for a regression task is difficult as it is to predict a value. To make predictions, we utilize training tasks. The basic idea is to borrow one training task $\mathcal{T}_i$ as the support set for the new coming task $\mathcal{T}^*$. The assumptions are that the training tasks and test tasks may be similar regarding data/pattern graph, and they share some specific task structures. The kernel K leverages the similarity. The algorithm is given in Algorithm 3. First, a training task is sampled randomly from the training data (line 1). Then, a set of auxiliary data (X, c) is taken from the task to serve as the support set (line 2), and is used to compute the posterior of the parameters $p(\rho_{\mathcal{T}^*}|c, X)$ (line 3-4).

The discrepancy between the two posterior densities $p(\rho_{\mathcal{T}^*}|c, X)$ and $p(\rho_{\mathcal{T}^*}|c_{\mathcal{S}^*}, X_{\mathcal{S}^*})$ determines the difference between the few- and zero-shot testing, which is further determined by the discrepancy of $p(c, X)$ and $p(c_{\mathcal{S}^*}, X_{\mathcal{S}^*})$. As the shape of $p(c, X)$ is delineated by $p(X)$ and $p(c|X)$, the higher the similarity between $p(X)$ and $p(X_{\mathcal{S}^*})$, and $p(c|X)$ and $p(c_{\mathcal{S}^*}|X_{\mathcal{S}^*})$, the well the auxiliary task works in the zero-shot testing [60].

For the subgraph counting tasks, it is difficult to make practical assumptions for both $p(X)$ and $p(c|X)$, for the 5 task configurations. The zero-shot testing will also have

different effects on different task configurations. It is worth noting that, for HySameG and HySameP tasks, the auxiliary task sampled should be the same domain/database with the testing task to pursue a high similarity between $p(c, X)$ and $p(c_{\mathcal{S}^*}, X_{\mathcal{S}^*})$. We also explore adding the auxiliary data (c, X) in the few-shot testing, together with $(c_{\mathcal{S}^*}, X_{\mathcal{S}^*})$ to approximate the posterior of $\rho_{\mathcal{T}} *$. Bayesian model average marginalizes all possible $\rho_{\mathcal{T}} *$ and gives a smooth and robust prediction.

Algorithm 1: RGIN-GP Meta Train

Input : Training Task Set $\mathcal{D} = \{\mathcal{T}_1, \cdots, \mathcal{T}_n\}$, learning rates α_w and α_θ

Output: neural network parameter w, kernel hyperparameter θ

repeat
  Sample a task $\mathcal{T} = (\mathcal{S}, \mathcal{Q})$ from $\mathcal{D}$
  $X \leftarrow X_{\mathcal{S}} \cup X_{\mathcal{Q}}$; $\boldsymbol{c} \leftarrow \boldsymbol{c}_{\mathcal{S}} \cup \boldsymbol{c}_{\mathcal{Q}}$
  Compute marginal loglikihood loss $\mathcal{L}_{\mathsf{mll}}$ by Eq. (12)
  $w \leftarrow w - \alpha_w \Delta_w$; $\theta \leftarrow \theta - \alpha_\theta \Delta_\theta$
until *stop criterion*

Algorithm 2: RGIN-GP Few-shot Test

Input : One Test Task $\mathcal{T}^* = (\mathcal{S}^*, \mathcal{Q}^*)$, neural network parameter w, kernel hyperparameter θ

Output: prediction $\boldsymbol{c}_{\mathcal{Q}^*}$ for $\mathcal{Q}^*$

$X \leftarrow X_{\mathcal{S}^*}$; $\boldsymbol{c} \leftarrow \boldsymbol{c}_{\mathcal{S}^*}$; $X^* \leftarrow X_{\mathcal{Q}^*}$;
Compute the conditional distribution of $\boldsymbol{c}_{\mathcal{Q}^*}$ by Eq. (10)-(11)

Algorithm 3: RGIN-GP Zero-shot Test

Input : One Test Task $\mathcal{T}^* = (\mathcal{Q}^*)$, Training Task Set $\mathcal{D} = \{\mathcal{T}_1, \cdots, \mathcal{T}_n\}$, neural network parameter w, kernel hyperparameter θ

Output: prediction $\boldsymbol{c}_{\mathcal{Q}^*}$ for $\mathcal{Q}^*$

Sample a task $\mathcal{T} = (\mathcal{S}, \mathcal{Q})$ from $\mathcal{D}$
Sample $(X, \boldsymbol{c}) \sim (X_{\mathcal{S}} \cup X_{\mathcal{Q}}, c_{\mathcal{S}} \cup c_{\mathcal{Q}})$
$X^* \leftarrow X_{\mathcal{Q}^*}$;
Compute the conditional distribution of $\boldsymbol{c}_{\mathcal{Q}^*}$ by Eq. (10)-(11)

3.3 Feature Encoding

Encoding initial node representation $\mathsf{e}_v^{(0)}$ for RGIN-GP in the neural network mapping $\mathcal{F}$ is important in learning. The one-hot encoding is widely used to represent the attribute features in GNN models for node classification, link prediction [26, 59] and subgraph counting [28]. However, such sparse encoding is lack of insight for the analytical subgraph counting. It is worth noting that the labels of a pattern node serve as the predicates of the pattern, and are used to filter nodes in the data graph. We explore frequency-based encoding and pre-trained embedding-based encoding to encode label information and topological structure.

Frequency − based Encoding. The frequency-based features encode the filter capability of a pattern node regarding the data graph. For a data graph g_i, we denote the number of occurrence of a node label l as $F(l) = |\{v \mid l \in L_V(v) \text{ for } v \in V_{g_i}\}|$. The node representation for v in a pattern graph q, $\mathsf{e}_v^{(0)}$, is encoded as a $|\Sigma_V|$-dimensional vector, $\mathsf{e}_v^{(0)} \in \mathbb{R}^{|\Sigma_V|}$, where Σ_V is the universal set of the node labels on data graphs, and the i-th dimension corresponds to the i-th node label $l_i \in \Sigma_V$. The value of $\mathsf{e}_v^{(0)}[i]$ is the fraction of the nodes in g can be matched to v. In detail, if node v is associated with a node label l_i, $\mathsf{e}_v^{(0)}[i]$ will be set to $F(l_i)/|V|$, otherwise $\mathsf{e}_v^{(0)}[i]$ will be set to 1.0. It is worth noting that for one query, its frequency-based encoding for different data graphs is different.

Embedding − based Encoding. The frequency-based encoding takes the attribute frequency of the data graph into account, but fails to leverage the topology of the data graphs. An encoding to encode the topological structure of the data graph together with its labels is needed. As feeding GNN a pre-trained and unsupervised embedding as node features can boost the performance, we pre-train a node label embedding for the data graphs to enhance the pattern graph encoding. To preserve the topological property of the training data graphs $\{g_1, \ldots, g_n\}$ together with the universal node label set (Σ_V), we construct a label-augmented graph $G_L = (V \cup V_L, E \cup E_L)$. Here, $V = V_{g_1} \cup \cdots V_{g_n}$ and $E = E_{g_1} \cup \cdots E_{g_n}$, which means the data graphs are treated as connected components of a large graph with node V and E. V_L is a set of nodes where a node represents a label in Σ_V, and there are $|\Sigma_V|$ nodes in V_L. E_L is a set of edges where an edge is between a node, v, in V with a node, l, in V_L, if v has the node label l that the node l corresponds to. We use a scalable, task-independent graph embedding algorithm (e.g., *DeepWalk* [40], *node2vec* [17], *ProNE* [62]) to pre-train a node embedding for the label-augmented graph G_L. With the pre-trained label embedding, we encode every node in a pattern graph q. For a node v in q, we set $\mathsf{e}_v^{(0)}$ to be $\sum_{l \in L_V(v)} \mathsf{e}'(l)$,

where $\mathbf{e}'(l)$ is the pre-trained embedding of the label l in G_L if v has the label l. A node v will have an all-zero vector if it does not have any labels.

4 Experimental Studies

In this section, we give the experimental setting (Sect. 4.1) and report our experiments in the facets: ① compare RGIN-GP with the neural network counterpart (Sect. 4.2), ② investigate the prediction performance under different task configurations (Sect. 4.3), ③ compare the meta-RGIN-GP with other optimization-based meta-learning approaches (Sect. 4.4), ④ compare RGIN-GP with subgraph counting algorithms (Sect. 4.5).

4.1 Experimental Setup

Implementation and Setting. We give the settings of RGIN-GP. For the neural network transformation $\mathcal{F}$, the number of RGIN layers is 3, where each hidden layer has 64 units and a Dropout probability of 0.2. For the stationary kernel function $\mathcal{K}$, we use the spectral mixtures based kernels [56], whose loss is consistently easy to converge than the widely used RBF kernel for our learning task. For the embedding-based encoding, we try 4 scalable task-independent node embedding approaches, i.e., *DeepWalk* [40], *node2vec* [17], *ProNE* [62], *NRP* [61] and finally choose *ProNE* as the embedding algorithm for the label-augmented graph due to its efficiency and stable performance. Following the setting in [62], the dimension of the embedding is 128.

The learning framework is built on PyTorch with PyTorch Geometric and GPyTorch [16]. We use the Adam optimizer with a decaying learning rate to train our models via 200 epochs. The initial learning rates for the neural network parameters and kernel hyperparameters (α_w and α_θ in Algorithm 1) are set to 5e-4 and 1e-3 empirically, respectively. Both training and prediction are performed on a Linux server with a Tesla V100 with 16GB memory.

Datasets. We use one real graph database MUTAG, and two synthetic graph databases SYN-Small and SYN-Large. MUTAG collection [11] has 188 unique nitroaromatic compounds where nodes represent atoms and edges represent bonds. The node (edge) label represents the atom (bond) type. The 24 patterns are from [28]. The two synthetic datasets SYN-Small and SYN-Large are generated by the generator of [28]. SYN-Small follows the same scale of MUTAG for the data graphs and pattern graphs. SYN-Large enlarges the scale of the MUTAG data and pattern graphs two times. Here, the data graphs are generated from patterns by adding nodes and edges to the patterns. Table 1 lists the profile of the three datasets.

Baseline Approaches. From the perspective of the model, we compare meta-learned RGIN-GP (`RGIN − GP`) with its neural network counterpart in [28], `RGIN + SumPool` and the model in [28] with the best prediction accuracy, `RGIN + DIAMNet`. From the perspective of the learning algorithm, we compare our meta-learned `RGIN − GP` with `RGIN + SumPool` trained by a classical meta-learning algorithm Model-Agnostic Meta-Learning (`MAML`) [15] and transfer learning under the linear protocol (`FeatTrans`). `MAML` optimizes the task parameters end-to-end by a bi-level back propagation. Specifically, the algorithm optimizes the task-specific parameters in one inner loop and the task-common parameters in the outer loop. `FeatTrans` treats each training task as one batch to optimize the task-common parameters. For a testing task, the parameters of the final layer in the MLP are finetuned by one gradient step. For the two algorithms, the model is the neural network model `RGIN + SumPool`.

Evaluation Metrics: We use the mean of abs-error, i.e., MAE (Eq. (1)), of the counts and the accuracy of the subgraph isomorphism query to evaluate the model performance. The two metrics do not have a direct correlation. The well performed model should achieve small MAE and high accuracy simultaneously on the test set. Small MAE but low accuracy indicates all the estimations are over-smoothed to the mean of the true counts. Large MAE but high accuracy indicates the model can only distinguish zero and nonzero counts but cannot predict well for nonzero counts.

4.2 RGIN-GP versus Neural Network Models

We first compare our `RGIN − GP` with its neural network counterpart `RGIN + SumPool`, and a more powerful model

Table 1 Profile of datasets

Dataset	Data graphs					Pattern graphs					# (g, p)	$c(g, p)$
	$\lvert V_g\rvert$	$\lvert E_g\rvert$	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_E\rvert$	# g	$\lvert V_p\rvert$	$\lvert E_p\rvert$	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_E\rvert$	# p		
MUTAG	[10, 28]	[20, 66]	[3, 7]	[3, 4]	188	[3, 4]	[2, 3]	[1, 2]	[1, 2]	24	4,512	[0, 156]
SYN-Small	[10, 28]	[20, 66]	[3, 7]	[3, 4]	30,681	[3, 4]	[2, 3]	[1, 2]	[1, 2]	240	30,681	[0, 126]
SYN-Large	[10, 56]	[22, 132]	[3, 7]	[3, 4]	102,057	[3, 8]	[2, 12]	[1, 2]	[1, 2]	1,680	102,057	[0, 128]

Table 2 MAE and accuracy on MUTAG

Train ratio	Model	MAE	Accuracy
0.6	RGIN + SumPool	10.41 ± 6.25	0.89 ± 0.02
	RGIN + DIAMNet	3.29 ± 0.82	0.83 ± 0.06
	RGIN − GP(onehot)	0.96 ± 0.15	0.92 ± 0.07
	RGIN − GP(freq)	0.92 ± 0.18	0.94 ± 0.04
	RGIN − GP(prone)	0.87 ± 0.13	0.93 ± 0.02
0.4	RGIN + SumPool	8.78 ± 9.14	0.88 ± 0.03
	RGIN + DIAMNet	7.72 ± 1.21	0.85 ± 0.36
	RGIN − GP(onehot)	0.94 ± 0.14	0.92 ± 0.06
	RGIN − GP(freq)	0.87 ± 0.12	0.91 ± 0.05
	RGIN − GP(prone)	0.82 ± 0.13	0.89 ± 0.04
0.2	RGIN + SumPool	8.65 ± 5.12	0.87 ± 0.02
	RGIN + DIAMNet	8.66 ± 5.11	0.87 ± 0.02
	RGIN − GP(onehot)	1.19 ± 0.23	0.84 ± 0.02
	RGIN − GP(freq)	1.21 ± 0.07	0.87 ± 0.05
	RGIN − GP(prone)	1.21 ± 0.08	0.84 ± 0.01

RGIN + DIAMNet on MUTAG dataset. For RGIN − GP, we organize the training and testing pairs as Random tasks with 128 pairs in each task. Testing is conducted in the zero-shot mode (Algorithm 3) with 128 auxiliary pairs drawn from the training data. For the two neural network models, they are trained by standard supervised learning. Table 2 shows the testing performance on 20% testing pairs when the training pairs are set to 60%, 40%, and 20% of the overall 4,512 pairs, respectively. In general, the 3 RGIN − GP variants remarkably outperform the two neural network models w.r.t. MAE. When the training ratio is 60% and 40%, the test MAE of RGIN − GP models is below 1.0. When the training ratio declines to 20%, the MAE only arises to about 1.2. The neural network RGIN + SumPool and RGIN + DIAMNet suffer from model degradation. We observed the loss converges slowly during training, and the test performance has a large variance w.r.t. multiple training, particularly for the complicated model RGIN + DIAMNet. The implication of this experiment is our RGIN − GP is robust, data-efficient and much easier to train than its neural network counterpart. This is because of its nature as a GP whose prediction is conducted by Bayesian model average.

4.3 Influence of Task Configuration

We investigate how task configurations influence the performance of RGIN − GP. For the 5 types of tasks, i.e., Random, SameG, SameP, HySameG, HySameP, by default, we use RGIN − GP(prone) as the model and use 7 training tasks from MUTAG database with 128 pairs and 16 shots for one task. For HySameG and HySameP, 239 tasks from the domain of SYN-Small are added into the training task set. First, we vary the number of shots in {1, 4, 16, 64}, and the prediction

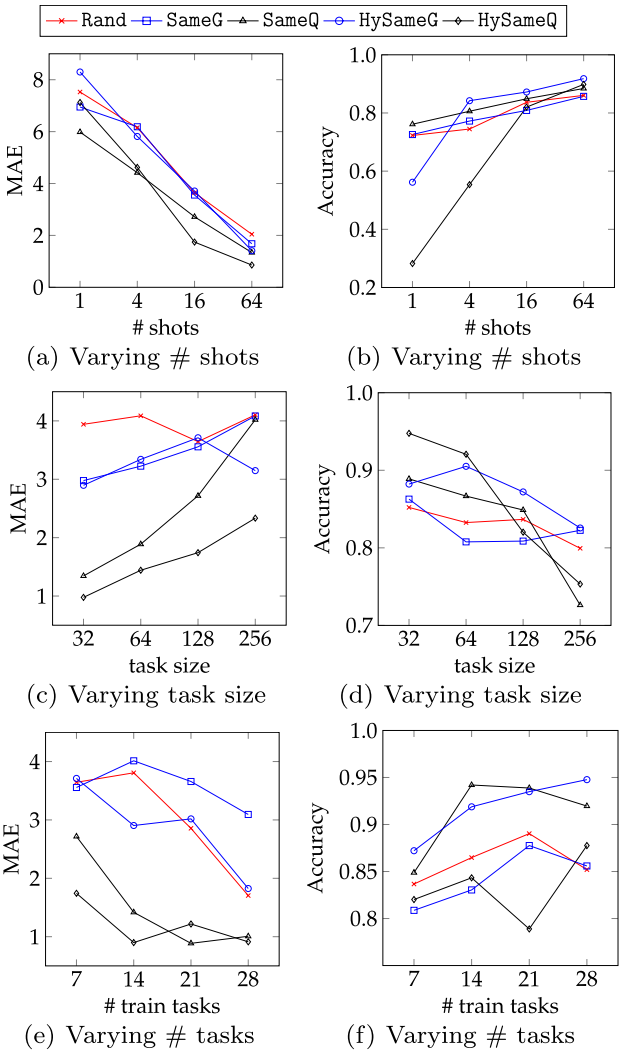

Fig. 2 Test results on MUTAG over various task configurations

MAE and accuracy are shown in Fig. 2a and b. The number of shots has a large influence on the testing MAE and accuracy. The larger the support set in each task, the better the performance, which is consistent with our intuition. As the number of shots grows exponentially, the performance gain improves marginally. Second, we vary the number of pairs in each task in {32, 64, 128, 256} by fixing the number of shots to 16 and MUTAG training tasks to 7. The testing results are shown in Fig. 2c and d. In general, varying the task size will not incur a large influence as varying the number of shots. However, we observe HySameG, HySameP are more sensitive to the task size compared with the other 3 task types. The reason would be when varying the MUTAG task size, the task structure between synthetic tasks and MUTAG task becomes different, e.g., the percentage of pairs with ground-truth in one task. And this difference brings greater performance variation. Third, we vary the number of MUTAG training tasks in {7, 14, 21, 28} by fixing the task size to 128

and the number of shots to 16. The corresponding MAE and accuracy on MUTAG task is presented in Fig. 2e and f. We conjecture that here the performance change of 21 training tasks is that some new task introduces inconsistent noise when training together with the synthetic dataset.

Furthermore, we investigate the effect of adding sampled data from the training tasks to the support set of the test task in the zero-shot and few-shot scenarios for the 5 task types. For SameG, SameP and Random, 7 tasks from MUTAG are used for training. For HySameG and SameP, 239 tasks from SYN-Small are added to the training tasks. We test 28 MUTAG tasks by varying the number of shots and the auxiliary pairs in $\{0, 1, 4, 16, 64\}, \{0, 16, 32, 64, 128\}$, respectively. The size of all the train and test tasks is 128. The testing performance over the 5 task configurations is shown in Table 3. For SameG, HySameG, and Random, the upper-left cell of the table is the worst performance for zero-shot without auxiliary data and the lower-right cell is the best performance for 64 shots with 128 auxiliary. As the number of shots or auxiliary pairs increases, the test performance improves from the upper-left to the upper-right, lower-left and upper-right, which is consistent with intuition. In addition, we find when the number of shots or the auxiliary pairs is sufficiently large, e.g., 64 shots or 128 auxiliary pairs, increasing the amount of auxiliary data or shot only contributes to marginal improvement. For a fixed size support set, the more data from the shot, the better the performance. For example, 16 shots without auxiliary pairs is better than zero-shot with 16 pairs. This is because the support set in a test task exhibits more task-specific features than the sampled training data. However, we find for tasks with the type SameP and HySameP, adding auxiliary data from training tasks will degrade the MAE and accuracy. Recall that SameP and HySameP task is one new pattern p^* for different data graphs $\mathcal{G}$ in a database. We observe that, for different patterns p_1 and p_2, their true count distributions of $\mathcal{G}$ are rather different, because of the different topology between p_1 and p_2. For example, p_1 in $\mathcal{G}$ has most zero counts and p_2 in $\mathcal{G}$ has most nonzero counts. A large discrepancy between the ground-truth distribution makes the knowledge transfer difficult. Under this circumstance, few-shot for the new pattern is important.

4.4 Comparison with Meta-Algorithms

We compare the meta-learned RGIN − GP with the meta-learned RGIN + SumPool by MAML and FeatTrans for

Table 3 Test MAE/accuracy for zero-shot

MAE/Acc.		# Auxiliary data				
		0	16	32	64	128
SameG # shots	0	8.06/0.72	4.01/0.82	3.13/0.84	2.37/0.84	2.24/0.86
	1	6.97/0.74	3.5/0.83	2.84/0.82	2.29/0.84	1.73/0.87
	4	6.22/0.77	3.14/0.83	2.71/0.85	2.28/0.85	1.81/0.87
	16	3.65/0.82	2.68/0.83	2.24/0.85	2.04/0.86	1.79/0.87
	64	1.80/0.85	1.67/0.86	1.67/0.86	1.49/0.87	1.48/0.87
SameP # shots	0	7.08/0.75	15.02/0.57	15.78/0.50	15.00/0.54	14.92/0.48
	1	6.10/0.77	11.11/0.52	12.45/0.54	16.45/0.40	14.54/0.42
	4	4.54/0.80	11.84/0.51	11.65/0.52	11.27/0.55	11.01/0.53
	16	2.51/0.84	6.48/0.54	6.68/0.63	6.51/0.64	10.09/0.57
	64	1.34/0.88	2.61/0.76	2.57/0.76	3.89/0.70	3.68/0.69
HySameG # shots	0	10.92/0.28	3.68/0.86	3.02/0.88	2.23/0.89	1.67/0.91
	1	8.51/0.49	3.76/0.86	2.69/0.87	2.01/0.90	1.81/0.90
	4	4.91/0.81	3.61/0.86	2.44/0.89	2.13/0.90	1.71/0.91
	16	3.71/0.87	2.89/0.88	1.88/0.89	1.77/0.90	1.59/0.91
	64	1.49/0.89	1.35/0.90	1.29/0.90	1.21/0.91	1.11/0.92
HySameP # shots	0	10.22/0.23	11.35/0.27	10.60/0.30	10.93/0.32	9.62/0.37
	1	7.67/0.32	7.77/0.36	9.54/0.36	8.32/0.36	9.34/0.42
	4	4.67/0.49	5.94/0.48	5.96/0.46	6.79/0.48	5.41/0.50
	16	1.95/0.78	2.50/0.72	2.45/0.71	2.54/0.69	2.62/0.70
	64	0.86/0.90	1.00/0.84	1.08/0.85	1.05/0.85	1.02/0.88
Random # shots	0	8.05/0.73	4.38/0.81	2.72/0.84	1.79/0.85	1.41/0.85
	1	7.17/0.76	3.83/0.82	3.45/0.83	1.74/0.84	1.43/0.87
	4	6.00/0.78	3.49/0.82	2.39/0.84	1.84/0.85	1.46/0.86
	16	3.98/0.79	2.21/0.82	2.06/0.81	1.45/0.84	1.26/0.85
	64	1.81/0.86	1.61/0.86	1.51/0.86	1.35/0.86	1.23/0.88

Table 4 Test MAE/accuracy for different algorithms

Task type	RGIN – GP	MAML	FeatTrans
Random	3.64/0.84	6.90/0.41	3.47/0.56
SameG	3.56/0.81	6.70/0.40	3.54/0.47
SameP	2.72/0.85	4.34/0.71	6.63/0.62
HySameG (SYN-Small)	3.71/0.87	6.85/0.53	2.26/0.67
HySameP (SYN-Small)	1.74/0.82	4.60/0.74	9.65/0.47
HySameG (SYN-Large)	3.09/0.86	OOM	2.01/0.81
HySameP (SYN-Large)	3.19/0.86	OOM	5.70/0.74

few-shot learning. For Random, SameG, SameP, 7 tasks from MUTAG are used as the training task set. For HySameG and HySameP, except for the 7 MUTAG tasks, either 239 tasks from SYN-Small or 797 tasks from SYN-Large are added. By default, the task size is 128 and the number of shots is 16. The MAE and accuracy on 28 test tasks from MUTAG are shown in Table 4. Only RGIN – GP achieves both low MAE and high accuracy consistently over all the 5 task types. Both MAML and FeatTrans fail to distinguish zero/nonzero counts as the test accuracy is low. MAML conducts the inner and outer loop gradient update end-to-end, differentiating through the inner loop to obtain the gradients for the outer loop, which will cause instability problems [1]. For the subgraph counting task, training loss of MAML converges slow. MAML is also resource consuming as it runs out of GPU memory on Hybrid tasks from MUTAG and SYN-Large. FeatTrans finetunes the final layer of RGIN + SumPool one gradient step for each task by freezing the parameters of the former feature transformation layers. The one gradient step may underfit the task data, whereas more gradient steps will overfit the task data.

4.5 Comparison with Algorithmic Approaches

We compare our meta-learned RGIN – GP with traditional subgraph counting algorithms, including 7 approximate algorithms in the GCARE benchmark [38], and an exact counting algorithm VF2 [9] implemented by NetworkX. The MAE, quantiles of the error and the total counting time are presented in Table 5. The prediction results of RGIN – GP are collected by fivefold cross-validation where one RGIN – GP(prone) model is trained over 20% MUTAG pairs that are organized in 7 tasks with type of Random and size of 128. The prediction is conducted by zero-shot testing with 128 auxiliary pairs. In Table 5, RGIN – GP achieves the lowest MAE among the 8 approximate approaches and its prediction is 6× faster than the exact algorithm VF2. Most of the approximate baselines have the problem of underestimation due to sampling failure.

5 Related Work

Subgraph Counting. There are exact and approximate subgraph counting algorithms [44]. For exact counting, existing approaches in the literature are categorized into enumeration-based [19, 22, 37] and analytical approaches [32, 36, 41]. For approximate subgraph counting, various estimation strategies have been explored, such as path sampling [23, 54], color coding [3, 4], random walk [6], and graph summarization [34, 50]. Most of the above approaches are designed to count graphlets [42], a.k.a. graph motifs, small, unlabeled graph queries, over a simple undirected graph. Recently, ML/DL-based approaches have been proposed to estimate the subgraph counting. [64, 65] devise a GNN based model as the counting sketch to estimate the subgraph isomorphism/homomorphism count over a large data graph, but the model cannot generalize to unseen data graphs. [28] proposes DL models to count queries specified by a pair of data graph and pattern graph. Both of the two models need training by a large number of training data. [7] analyzes the ability of GNNs in detecting subgraph isomorphism, and proposes a Local Relational Pooling model for counting both subgraphs and induced subgraphs. However, the model is specific to a given query pattern.

Meta-Learning on Graphs. There are three types of meta-learning algorithms, i.e., black-box adaption [33, 46], optimization-based [15, 35] and metric-based [48, 51] meta-learning algorithms. Black-box adaption relies on specific neural network architecture, e.g., RNN to learn each task one by one. These approaches have the powerful expressive

Table 5 Comparison with subgraph counting algorithms

Method	RGIN – GP	WJ	CS	CSET	IMPR	JSUB	BSK	SumRDF	VF2
MAE	1.32	8.05	25.35	7.87	40.28	7.69	156.15	6.48	0
5%	−4.66	−48	−56	−56	−24	−48	0	−45	0
25%	0.01	−4	−4	−4	0	−4	0	0	0
50%	0.01	0	0	0	0	0	0	0	0
75%	0.35	0	0	0	0	0	66	0	0
95%	3.68	0	0	0	252.96	0	932	2.21	0
Time (s)	0.14	0.59	0.98	0.55	0.82	0.61	693.02	40.55	0.89

power to model the task prior but are data inefficient and challenging for optimization. The optimization-based algorithms learn a hierarchical model by gradient-based back propagation. These algorithms are effective regarding learning the meta-model but are time and memory consuming in computation due to the hierarchical optimization paradigm. The metric-based algorithms borrow the idea from the clustering algorithms that learn an embedding for the input. The algorithms are applicable for small data but only classification tasks.

These meta-learning approaches have been adopted over graph data to deal with various graph learning tasks, including node classification [21, 66], link prediction [2, 21], graph classification [5, 29]. [14] proposes a meta-learning framework, conditional graph neural process, for community search. A survey summarizes the applications and methods can be found in [30]. Here, GNN is widely used as the base model or core component of these approaches. However, all the existing approaches are oriented to few-shot graph learning tasks and cannot be directly applied to zero-shot learning or the subgraph counting task, where the input is specified by a data graph and a pattern graph.

6 Conclusion

In this paper, we study an NP-complete problem, subgraph isomorphism counting, by DL techniques. To alleviate the reliance on a large volume of training data, we devise a Gaussian Process, called RGIN-GP, which warps a neural network layer. The model is trained end-to-end by a meta-learning algorithm, which aims to exploit the knowledge prior of training tasks. Compared with the baseline approach, the meta-trained RGIN-GP reduces the MAE from 8 to 1, with only one thousand training samples. In addition, in our extensive experiments, the meta-learned RGIN-GP can fast adapt to new tasks by few-shot and zero-shot learning.

Acknowledgements The conference version of this paper appears in [63]. Jeffrey Xu Yu was supported by the Research Grants Council of Hong Kong, China, under Nos. 14202919 and 14205520. This work was supported by the Research Grants Council of Hong Kong, China, under Nos. 14203618, 14202919 and 14205520. The conference version of this work appears in DASFAA 2023 "Learning with Small Data: Subgraph Counting Queries".

Author Contributions KZ methodology, implementation, presentation ZH data preparation, presentation JXY methodology, presentation YR testing.

Fundings Research Grants Council of Hong Kong, No. 14203618 Research Grants Council of Hong Kong, No. 14202919 Research Grants Council of Hong Kong, No. 14205520.

References

1. Antoniou A, Edwards H, Storkey AJ (2019) How to train your MAML. In: Proc. of ICLR. OpenReview.net
2. Bose AJ, Jain A, Molino P, Hamilton, WL (2019) Meta-graph: few shot link prediction via meta learning. CoRR. arXiv:abs/1912.09867
3. Bressan M, Chierichetti F, Kumar R, Leucci S, Panconesi A (2017) Counting graphlets: space vs time. In: Proceedings of the WSDM'17, pp 557–566
4. Bressan M, Leucci S, Panconesi A (2019) Motivo: fast motif counting via succinct color coding and adaptive sampling. Proc VLDB 12(11):1651–1663
5. Chauhan J, Nathani D, Kaul, M (2020) Few-shot learning on graphs via super-classes based on graph spectral measures. In: Proceedings of the CLR 2020. OpenReview.net
6. Chen X, Lui JCS (2016) Mining graphlet counts in online social networks. In: Proceedings of the ICDM'16, pp 71–80
7. Chen Z, Chen L, Villar S, Bruna J (2020) Can graph neural networks count substructures? In: Proceedings of the NeurIPS
8. Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the STOC, pp 151–158
9. Cordella LP, Foggia P, Sansone C, Vento M (2004) A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans Pattern Anal Mach Intell 26(10):1367–1372
10. Dai Z, Yang Z, Yang Y, Carbonell JG, Le QV, Salakhutdinov R (2019) Transformer-xl: attentive language models beyond a fixed-length context. In: Proceedings of the ACL, pp 2978–2988
11. Debnath AK, Lopez de Compadre RL, Debnath G, Shusterman AJ, Hansch C (1991) Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. J Med Chem 34(2):786–797
12. Ding B, Das S, Marcus R, Wu W, Chaudhuri S, Narasayya VR (2019) AI meets AI: leveraging query executions to improve index recommendations. In Proceedings of SIGMOD, pp 1241–1258
13. Dutt A, Wang C, Nazi A, Kandula S, Narasayya VR, Chaudhuri S (2019) Selectivity estimation for range predicates using lightweight models. Proc VLDB 12(9):1044–1057
14. Fang S, Zhao K, Li G, Yu JX (2023) Community search: a meta-learning approach. In: Proceedings of the ICDE'23. IEEE, pp 2358–2371
15. Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of ICML, vol 70, pp 1126–1135

16. Gardner JR, Pleiss G, Weinberger KQ, Bindel D, Wilson AG (2018) Gpytorch: blackbox matrix-matrix gaussian process inference with GPU acceleration. In: Proceedings of the NeurIPS, pp 7587–7597
17. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the KDD'16, pp 855–864
18. Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the NeurIPS'17, pp 1024–1034
19. Hocevar T, Demsar J (2016) Combinatorial algorithm for counting small induced graphs and orbits. CoRR arXiv:abs/1601.06834
20. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
21. Huang K, Zitnik M (2020) Graph meta learning via local subgraphs. In: Proceedings of the NeurIPS
22. Jain S, et al (2017) Impact of memory space optimization technique on fast network motif search algorithm. In: Advances in computer and computational sciences. Springer, pp 559–567
23. Jha M, Seshadhri C, Pinar A (2015) Path sampling: a fast and provable method for estimating 4-vertex subgraph counts. In: Proceedings of the WWW'15, pp 495–505
24. Kim Y (2014) Convolutional neural networks for sentence classification. In: ACL. ACL, pp 1746–1751
25. Kipf A, Kipf T, Radke B, Leis V, Boncz PA, Kemper A (2019) Learned cardinalities: estimating correlated joins with deep learning. In Proceedings of the CIDR'19
26. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the ICLR'17
27. Liang X, Elmore AJ, Krishnan S (2019) Opportunistic view materialization with deep reinforcement learning. CoRR arXiv:abs/1903.01363
28. Liu X, Pan H, He M, Song Y, Jiang X, Shang L (2020) Neural subgraph isomorphism counting. In: Proceedings of the KDD '20, pp 1959–1969
29. Ma N, Bu J, Yang J, Zhang Z, Yao C, Yu Z, Zhou S, Yan X (2020) Adaptive-step graph meta-learner for few-shot graph classification. In Proceedings of the CIKM 2020. ACM, pp 1055–1064
30. Mandal D, Medya S, Uzzi B, Aggarwal C (2021) Meta-learning with graph neural networks: methods and applications. CoRR arXiv:abs/2103.00137
31. Marcus RC, Negi P, Mao H, Zhang C, Alizadeh M, Kraska T, Papaemmanouil O, Tatbul N (2019) Neo: a learned query optimizer. Proc VLDB Endow 12(11):1705–1718
32. Melckenbeeck I, Audenaert P, Colle D, Pickavet M (2018) Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. Bioinformatics 34(8):1372–1380
33. Munkhdalai T, Yu H (2017) Meta networks. In: Proceedings of the ICML. PMLR, pp 2554–2563
34. Neumann T, Moerkotte G (2011) Characteristic sets: accurate cardinality estimation for RDF queries with multiple joins. In: Proceedings of the ICDE'11, pp 984–994
35. Nichol A, Achiam J, Schulman J (2018) On first-order meta-learning algorithms. CoRR arXiv:abs/1803.02999
36. Ortmann M, Brandes U (2017) Efficient orbit-aware triad and quad census in directed and undirected graphs. Appl Netw Sci 2:13
37. Paredes P, Ribeiro PMP (2013) Towards a faster network-centric subgraph census. In: Proceedings of the ASONAM '13, pp 264–271
38. Park Y, Ko S, Bhowmick SS, Kim K, Hong K, Han W (2020) G-CARE: a framework for performance benchmarking of cardinality estimation techniques for subgraph matching. In: Proceedings of the SIGMOD'20, pp 1099–1114
39. Patacchiola M, Turner J, Crowley EJ, Storkey A (2020) Bayesian meta-learning for the few-shot setting via deep kernels. In: Proceedings of the NeurIPS
40. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the KDD'14, pp 701–710
41. Pinar A, Seshadhri C, Vishal V (2017) ESCAPE: efficiently counting all 5-vertex subgraphs. In: Proceedings of the WWW'17, pp 1431–1440
42. Przulj N (2007) Biological network comparison using graphlet degree distribution. Bioinformatics 23(2):177–183
43. Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. MIT Press, New York
44. Ribeiro P, Paredes P, Silva MEP, Aparício D, Silva F (2019) A survey on subgraph counting: Concepts, algorithms and applications to network motifs and graphlets. CoRR arXiv:abs/1910.13011
45. Salakhutdinov R, Hinton GE (2007) Using deep belief nets to learn covariance kernels for gaussian processes. In: Proceedings of NIPS, pp 1249–1256
46. Santoro A, Bartunov S, Botvinick M, Wierstra D, Lillicrap TP (2016) Meta-learning with memory-augmented neural networks. In: Proceedings of ICML, vol 48. JMLR.org, pp 1842–1850
47. Schlichtkrull MS, Kipf TN, Bloem P, van den Berg R, Titov I, Welling M (2018) Modeling relational data with graph convolutional networks. In: Proceedings of the ESWC vol 10843, pp 593–607
48. Snell J, Swersky K, Zemel RS (2017) Prototypical networks for few-shot learning. In: Proceedings of the NIPS, pp 4077–4087
49. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
50. Stefanoni G, Motik B, Kostylev EV (2018) Estimating the cardinality of conjunctive queries over RDF data using graph summarisation. In: Proceedings of the WWW'18, pp 1043–1052
51. Sung F, Yang Y, Zhang L, Xiang T, Torr PHS, Hospedales TM (2018) Learning to compare: relation network for few-shot learning. In: Proceedings of the CVPR, pp 1199–1208
52. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of the NeurIPS, pp 5998–6008
53. Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: Proceeings of the ICLR
54. Wang P, Zhao J, Zhang X, Li Z, Cheng J, Lui JCS, Towsley D, Tao J, Guan X (2018) MOSS-5: a fast method of approximating counts of 5-node graphlets in large graphs. IEEE Trans Knowl Data Eng 30(1):73–86
55. Wang W, Zheng VW, Yu H, Miao C (2019) A survey of zero-shot learning: settings, methods, and applications. ACM Trans Intell Syst Technol 10(2):13:1-13:37
56. Wilson AG, Adams RP (2013) Gaussian process kernels for pattern discovery and extrapolation. In: Proceedings of the ICML, vol 28, pp 1067–1075
57. Wilson AG, Hu Z, Salakhutdinov R, Xing EP (2016) Deep kernel learning. In: Proceedings of the AISTATS, vol 51, pp 370–378
58. Wilson AG, Izmailov P (2020) Bayesian deep learning and a probabilistic perspective of generalization. In: Proceedings of the NeurIPS
59. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: Proceedings of the ICLR'19
60. Yang Q, Zhang Y, Dai W, Pan SJ (2020) Transfer learning. Cambridge University Press, Cambridge
61. Yang R, Shi J, Xiao X, Yang Y, Bhowmick SS (2020) Homogeneous network embedding for massive graphs via reweighted personalized pagerank. Proc VLDB 13(5):670–683

62. Zhang J, Dong Y, Wang Y, Tang J, Ding M (2019) Prone: fast and scalable network representation learning. In: Proceedings of the IJCAI'19, pp 4278–4284
63. Zhao K, Yu JX, He Z, Rong Y (2023) Learning with small data: subgraph counting queries. In: Proceedings of the DASFAA 2023. Springer, pp 308–319
64. Zhao K, Yu JX, Li Q, Zhang H, Rong Y (2023) Learned sketch for subgraph counting: a holistic approach. VLDB J 1–26
65. Zhao K, Yu JX, Zhang H, Li Q, Rong Y (2021) A learned sketch for subgraph counting. In: Proceedings of the SIGMOD'21
66. Zhou F, Cao C, Zhang K, Trajcevski G, Zhong T, Geng J (2019) Meta-gnn: on few-shot node classification in graph meta-learning. In: Proceedings of the CIKM 2019. ACM, pp 2357–2360