COUNT-GNN: GRAPH NEURAL NETWORKS FOR SUBGRAPH ISOMORPHISM COUNTING

Anonymous authors
Paper under double-blind review

ABSTRACT

The prevalence of graph structures has attracted a surge of research interest in graph data. As many graph-based tasks exploit recurring subgraph patterns on graphs, subgraph isomorphism counting becomes an important problem. Classical methods usually boil down to a backtracking framework that needs to navigate a huge search space with prohibitive computational cost due to the #P-completeness of the problem. Some recent studies resort to graph neural networks (GNNs) to learn a low-dimensional representation for both the query and the input graph, in order to predict the number of subgraph isomorphisms on the input graph. However, typical GNNs employ a node-centric message passing mechanism that receives and aggregates messages on nodes. While effective on node-oriented tasks, they become inadequate in complex structure matching for isomorphism counting. Moreover, given an input graph, the space of possible query graphs is enormous, and different parts of the input graph will be triggered to match different queries. Thus, expecting a fixed representation of the input graph to match diversely structured query graphs is unrealistic. In this paper, we propose a novel GNN called COUNT-GNN for subgraph isomorphism counting, to deal with the above challenges. At the edge level, we resort to an *edge-centric message passing* scheme, where messages on edges are propagated and aggregated based on the edge adjacency. By treating edges as first-class citizens, COUNT-GNN is able to preserve fine-grained structural information, given that an edge is an atomic unit of encoding graph structures. At the graph level, we *modulate the input graph representation* conditioned on the query, so that the input graph can be adapted to each query individually to improve their matching. To demonstrate the effectiveness and efficiency of COUNT-GNN, we conduct extensive experiments on a number of benchmark datasets. Results show that COUNT-GNN achieves superior performance in comparison to the state-of-the-art baselines.

1 INTRODUCTION

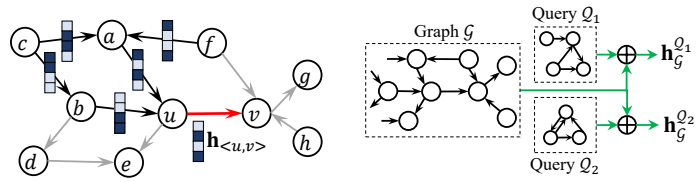
Graph structures are prevalent in real-world scenarios, catalyzing intensive research in network science and graph mining and learning. To discover graph-based insights, much research finds and exploits recurring subgraph patterns on an input graph. For example, on a protein network, we could query for the hydroxy groups which consist of one oxygen atom covalently bonded to one hydrogen atom; on a social network, we could query for potential families in which several users form a clique and two of them are working and the rest are studying. These queries essentially describe a subgraph pattern that repeatedly occurs on different parts of an input graph, which expresses certain semantics such as the hydroxy groups or families. These subgraph patterns are also known as network motifs on homogeneous graphs (Milo et al. (2002)) or meta-structures on heterogeneous graphs (Sun et al. (2011); Fang et al. (2016)). To leverage their expressiveness, more sophisticated graph models (Monti et al. (2018a); Sankar et al. (2019); Wang et al. (2019b)) have also been designed to specifically incorporate motifs or meta-structures.

The need for subgraph patterns in graph-based tasks and models leads to a high demand of *subgraph isomorphism counting* (Liu et al. (2020)), a significant problem yet to be adequately addressed. Classical methods for subgraph isomorphism detection and counting usually resort to search-based algorithms such as backtracking (Ullmann (1976); Cordella et al. (2004); He & Singh (2008)). Although they can exhaustively detect the isomorphisms and return an exact count, their computational

costs are often excessive given that the problem is NP-complete (the counting form is #P-complete). With the rise of graph neural networks (GNNs) (Wu et al. (2020)), some recent approaches for subgraph isomorphism counting also leverage on the powerful graph representations from GNNs (Liu et al. (2020); Zhengdao et al. (2020)). They generally employ GNNs to embed the query and input graph into low-dimensional vectors, which are further fed into a counter module to predict the approximate number of isomorphisms on the input graph. Compared to classical approaches, they can significantly save computational resources and time at the expense of approximation. The empirical erros are usually within a tolerable margin, providing a useful trade-off between the accuracy and computational cost since many applications do not necessarily need an exact count.

However, previous GNN-based isomorphism counting models utilize a node-centric message passing mechanism, which propagates and aggregates messages on nodes. While this mechanism can be effective for node-oriented tasks, it falls short of matching complex structures for isomorphism counting. In particular, they usually rely on message aggregation for local view representations centering on nodes, failing to fundamentally capture the subtle structures especially the link adjacency which is the atomic element in network structures, exposing a bottleneck of prior studies. Thus, as the first challenge, *how do we capture fine-grained structural information* beyond node-centric GNNs? Moreover, even for the same input graph, the space of possible query graphs is enormous. Different queries are often characterized by distinct structures that match with different parts of the input graph. A fixed representation to match with all possible queries is likely to underperform. Thus, as the second challenge, *how do we adapt the input graph to each query individually*, in order to improve the matching of specific structures in every query?

In this paper, we propose a novel model called COUNT-GNN for subgraph isomorphism counting, which copes with the above challenges from both the edge and graph perspectives. To be more specific, at the edge level, COUNT-GNN is built upon an *edge-centric* GNN that propagates and aggregates messages on and for edges based on the edge adjacency, as shown in Fig. 1(a). Given that edges constitute the atomic unit of graph structures, any subgraph is composed of one or more edge chains. Thus, treating them as first class citizens can better capture fine-grained structural information. At the graph level, COUNT-GNN resorts to a modulation mechanism (Perez et al. (2018)) by adapting edge-centric graph representations to each query graph, as shown in Fig. 1(b). As a result, the input graph can be tailored to each query individually, which may differ significantly in their structures. Coupling the two perspectives, COUNT-GNN is able to precisely match complex structures between the input graph and structurally diverse queries.



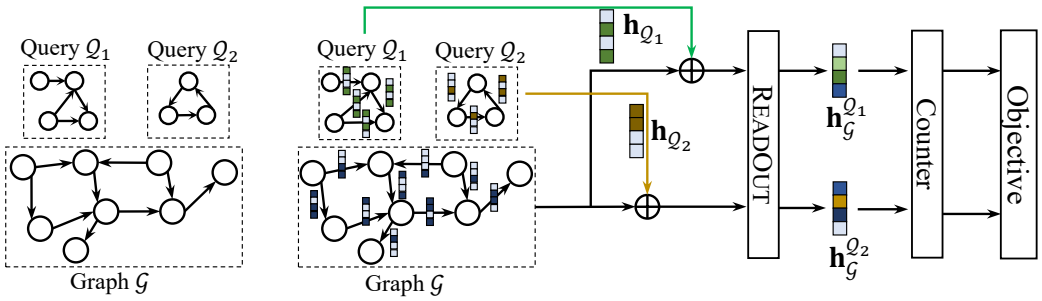
(a) Edge-centric aggregation (b) Modulation conditioned on queries

Figure 1: Illustration of COUNT-GNN.

To summarize, our contributions are three-fold. (1) We propose a novel model COUNT-GNN that capitalizes on edge-centric aggregation to encode fine-grained structural information, to improve structure matching between the queries and input graph from the edge perspective. (2) Moreover, we design an query-conditioned graph modulation in COUNT-GNN, to adapt structure matching to different queries from the graph perspective. (3) Extensive experiments on several benchmark datasets demonstrate that COUNT-GNN can significantly outperform state-of-the-art GNN-based models on subgraph isomorphism counting.

2 PROBLEM FORMULATION

A *graph* $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ is defined by a set of nodes $V_\mathcal{G}$, and a set of edges $E_\mathcal{G}$ between the nodes. In our study we consider the general case of directed edges, where an undirected edge can be treated as two directed edges in opposite directions. In our problem, we further consider *labeled* graphs (also known as heterogeneous graphs), in which there exists a node label function $\ell : V_\mathcal{G} \to L$ and an edge label function $\ell' : E_\mathcal{G} \to L'$, where L and L' denote the set of labels on nodes and edges, respectively. A graph $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ is a *subgraph* of $\mathcal{G}$, written as $\mathcal{S} \subseteq \mathcal{G}$, if and only if $V_\mathcal{S} \subseteq V_\mathcal{G}$ and $E_\mathcal{S} \subseteq E_\mathcal{G}$.



(a) Toy queries/graph (b) Edge-centric aggregation (c) Modulation w.r.t. queries (d) Objective

Figure 2: Overall framework of COUNT-GNN.

We are now ready to present the formal definition of subgraph isomorphism on a labeled graph, followed by our problem formulation.

Definition 1 (Labeled Subgraph Isomorphism)**.** *Consider a subgraph $\mathcal{S}$ of some* input graph*, and a* query graph $\mathcal{Q}$*. The subgraph $\mathcal{S}$ is* isomorphic *to the query $\mathcal{Q}$, written as $\mathcal{S} \simeq \mathcal{Q}$, if there exists a bijection between their nodes, $\psi : V_{\mathcal{S}} \to V_{\mathcal{Q}}$, such that*

- $\forall v \in V_{\mathcal{S}}$, $\ell(v) = \ell(\psi(v))$;
- $\forall e = \langle u, v \rangle \in E_{\mathcal{S}}$, *it must hold that* $e' = \langle \psi(u), \psi(v) \rangle \in E_{\mathcal{Q}}$ *and* $\ell'(e) = \ell'(e')$.

In our problem of *subgraph isomorphism counting*, we are given a query graph $\mathcal{Q}$ and an input graph $\mathcal{G}$. We aim to predict $n(\mathcal{Q}, \mathcal{G})$, the number of subgraphs of $\mathcal{G}$ which are isomorphic to $\mathcal{Q}$, *i.e.*, the cardinality of the set $\{\mathcal{S} | \mathcal{S} \subseteq \mathcal{G}, \mathcal{S} \simeq \mathcal{Q}\}$. Note that this is a non-trivial #P-complete problem (Cordella et al. (2004)). In practice, the query $\mathcal{Q}$ usually has a much smaller size than the input graph $\mathcal{G}$, *i.e.*, $|V_{\mathcal{Q}}| \ll |V|$ and $|E_{\mathcal{Q}}| \ll |E|$, leading to a huge search space and computational cost.

3 THE PROPOSED MODEL: COUNT-GNN

In this section, we first present the overall framework of COUNT-GNN. Next, we illustrate each module as well as the overall objective.

3.1 OVERALL FRAMEWORK

We give an overview of the proposed COUNT-GNN in Fig. 2. Consider some query graphs and an input graph in Fig. 2(a). On both the query and input graphs, we first conduct edge-centric aggregation in which messages on edges are propagated to and aggregated for each edge based on the edge adjacency, as shown in Fig. 2(b). This module operates at the edge level, and enables us to learn edge-centric representations for both input graphs and queries that capture their fine-grained structural information for better structure matching. Furthermore, to be able to match diverse queries with distinct structures, the edge-centric graph representations are modulated conditioned on each query, as shown in Fig. 2(c). The module operates at the graph level, and enables us to adapt the input graph to each query individually to improve the matching of specific structures in each query. Finally, as shown in Fig.(d), a counter module is applied to predict the isomorphism counting on the input graph for a particular query, forming the overall objective.

3.2 EDGE-CENTRIC AGGREGATION

Typical GNNs (Kipf & Welling (2017); Veličković et al. (2018); Hamilton et al. (2017)) and GNN-based isomorphism counting models (Liu et al. (2020); Zhengdao et al. (2020)) resort to the key mechanism of node-centric message passing, in which each node receives and aggregates messages from its neighboring nodes. For the problem of subgraph isomorphism counting, it is crucial to capture fine-grained structural information for more precise structure matching between the query and the input graph. Consequently, we exploit edge-centric message passing, in which each edge receives and aggregates messages from adjacent edges. The edge-centric GNN captures structural information more explicitly given that edges represent the fundamental, atomic unit of graph structures.

To ground our study, we learn a representation vector for each edge by propagating messages on edges. A message can be an input feature vector of the edge in the input layer of the GNN, or an intermediate embedding vector of the edge in subsequent layers. Specifically, given a directed edge $e = \langle u, v\rangle$, we initialize its message as a d_0-dimensional vector $\mathbf{h}^0_{\langle u,v\rangle} = \mathbf{x}_u \parallel \mathbf{x}_{\langle u,v\rangle} \parallel \mathbf{x}_v \in \mathbb{R}^{d_0}$, where $\mathbf{x}_*$ encodes the input features of the corresponding nodes or edges and $\parallel$ denotes the concatenation operator. In general, $\mathbf{h}^0_{\langle u,v\rangle} \neq \mathbf{h}^0_{\langle v,u\rangle}$ for directed edges. Note that, in the absence of input features, we can employ one-hot encoding as the feature vector; it is also possible to employ additional embedding layers to further transform the input features into initial messages.

Given the initial messages, we devise an edge-centric GNN where each edge receives and aggregates messages along the directed edges recursively in multiple layers. Formally, in layer l, the message on a directed edge $\langle u, v\rangle$, denoted $\mathbf{h}^l_{\langle u,v\rangle} \in \mathbb{R}^{d_l}$, is updated as

$$\mathbf{h}^l_{\langle u,v\rangle} = \sigma(\mathbf{W}^l \mathbf{h}^{l-1}_{\langle u,v\rangle} + \mathbf{U}^l \mathbf{h}^{l-1}_{\langle \cdot,u\rangle} + \mathbf{b}^l), \tag{1}$$

where $\mathbf{W}^l$, $\mathbf{U}^l \in \mathbb{R}^{d_l \times d_{l-1}}$ are learnable weight matrices, $\mathbf{b}^l \in \mathbb{R}^{d_l}$ is a learnable bias vector, and σ is an activation function (we use LeakyReLU in the implementation). In addition, $\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} \in \mathbb{R}^{d_{l-1}}$ is the intermediate message aggregated from the preceding edges of $\langle u, v\rangle$, *i.e.*, edges incident on node u from other nodes, which can be formalized as

$$\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} = \textsc{Aggr}(\{\mathbf{h}^{l-1}_{\langle i,u\rangle} | \langle i, u\rangle \in E\}), \tag{2}$$

where E denotes the set of directed edges in the graph (either the query or input graph), and $\textsc{Aggr}(\cdot)$ is an aggregation operator to aggregate messages from the preceding edges. We implement the aggregation operator as a simple mean, although more advanced approaches such as self-attention (Hamilton et al. (2017)) and sum-pooling (Xu et al. (2019)) can also be employed. To boost the message passing capacity, more advanced mechanisms can be imported into the layer-wise edge-centric aggregation, *e.g.*, a residual (He et al. (2016)) can be added to assist the message passing from previous layers to the current layer.

The above layer-wise edge-centric aggregation is applied to each query and input graph. All query graphs share one set of GNN parameters (*i.e.*, $\mathbf{W}^l, \mathbf{U}^l, \mathbf{b}^l$), while all input graphs share another set. On all graphs, the aggregated message on an edge $e = \langle u, v\rangle$ in the last layer is taken as the representation vector of the edge, denoted as $\mathbf{h}_{\langle u,v\rangle} \in \mathbb{R}^d$. Beyond the edge level, COUNT-GNN fuse the edge-centric representations into a whole-graph representation to facilitate structure matching between query and input graphs, which we will elaborate in Sect. 3.3.

3.3 QUERY-CONDITIONED GRAPH MODULATION

Toward structure matching between the query and input graph, we derive a whole-graph representation for each graph by fusing its edge representations. The whole-graph representations will be leveraged for matching the structures between graphs to predict subgraph isomorphisms.

Query graph representation. We employ a typical readout function (Xu et al. (2019); Lee et al. (2019); Yao et al. (2020)) for a query graph, by aggregating all edge representations in the query. Given a query graph $\mathcal{Q}$, its whole-graph representation is computed as

$$\mathbf{h}_{\mathcal{Q}} = \sigma(\mathbf{Q} \cdot \textsc{Aggr}(\{\mathbf{h}_{\langle u,v\rangle} | \langle u, v\rangle \in E_{\mathcal{Q}}\}), \tag{3}$$

where $\mathbf{Q} \in \mathbb{R}^{d \times d}$ is a learnable weight matrix shared by all query graphs, and we use sum for the aggregation. Intuitively, the query graph representation simply pools all edge representations together uniformly.

Input graph representation. A straightforward way for the input graph representation is to rely on Eq. (3) as well, which regards all edges uniformly. However, on an input graph, the space of possible query graphs is enormous. Thus, different queries are often characterized by distinct structures, which implies that different parts of the input graph will be triggered to match different queries. Therefore, aggregating all edges in the input graph uniformly cannot retain sufficiently specific structural properties w.r.t. each query. In other words, using a fixed whole-graph representation for the input graph cannot tailor to each query individually for effective structure matching. Thus, we propose to modulate the input graph conditioned on the query, to adapt its whole-graph

representation to each query uniquely. To this end, we leverage Feature-wise Linear Modulation or FiLM (Perez et al. (2018)) on the edge representations in the input graph, conditioned on the query, in order to retain the most specific structures for the query. The modulation is essentially an affine transformation based on scaling and shifting to adapt the edge representations of the input graph to the query. Specifically, given an input graph $\mathcal{G}$, for each edge $e = \langle u, v\rangle \in E_{\mathcal{G}}$ we modulate its representation $\mathbf{h}_{\langle u,v\rangle}$ into $\tilde{\mathbf{h}}_{\langle u,v\rangle}$, as follows.

$$\tilde{\mathbf{h}}_{\langle u,v\rangle} = (\boldsymbol{\gamma}_{\langle u,v\rangle} + \mathbf{1}) \odot \mathbf{h}_{\langle u,v\rangle} + \boldsymbol{\beta}_{\langle u,v\rangle}, \tag{4}$$

where $\boldsymbol{\gamma}_{\langle u,v\rangle}$, $\boldsymbol{\beta}_{\langle u,v\rangle} \in \mathbb{R}^d$ are FiLM factors for scaling and shifting, respectively, $\odot$ denotes the Hadamard product, and $\mathbf{1} \in \mathbb{R}^d$ is a vector filled with ones to center the scaling factor around one. Note that the FiLM factors $\boldsymbol{\gamma}_{\langle u,v\rangle}$ and $\boldsymbol{\beta}_{\langle u,v\rangle}$ are not directly learnable, but are instead generated by a secondary network (Ha et al. (2017)) conditioned on the original edge representation $\mathbf{h}_{\langle u,v\rangle}$ and the query representation $\mathbf{h}_{\mathcal{Q}}$. More specifically,

$$\boldsymbol{\gamma}_{\langle u,v\rangle} = \sigma(\mathbf{W}_\gamma \mathbf{h}_{\langle u,v\rangle} + \mathbf{U}_\gamma \mathbf{h}_{\mathcal{Q}} + \mathbf{b}_\gamma), \tag{5}$$

$$\boldsymbol{\beta}_{\langle u,v\rangle} = \sigma(\mathbf{W}_\beta \mathbf{h}_{\langle u,v\rangle} + \mathbf{U}_\beta \mathbf{h}_{\mathcal{Q}} + \mathbf{b}_\beta), \tag{6}$$

where $\mathbf{W}_\gamma, \mathbf{U}_\gamma, \mathbf{W}_\beta, \mathbf{U}_\beta \in \mathbb{R}^{d\times d}$ are learnable weight matrices, and $\mathbf{b}_\gamma, \mathbf{b}_\beta \in \mathbb{R}^d$ are learnable bias vectors.

The modulated edge representations can be further fused via a readout function, to result in a modulated whole-graph representation for the input graph tailored toward specific structures in each query. The adaptation to each query individually enables more precise matching between the input graph and the query, which ultimately improve subgraph isomorphism counting downstream. Concretely, consider a query graph $\mathcal{Q}$ and input graph $\mathcal{G}$. We formulate the $\mathcal{Q}$-conditioned representation for $\mathcal{G}$, denoted $\mathbf{h}_{\mathcal{G}}^{\mathcal{Q}} \in \mathbb{R}^d$, by aggregating the modulated edge representations of $\mathcal{G}$ in the following.

$$\mathbf{h}_{\mathcal{G}}^{\mathcal{Q}} = \sigma(\mathbf{G} \cdot \textsc{Aggr}(\{\tilde{\mathbf{h}}_{\langle u,v\rangle} | \langle u, v\rangle \in E_{\mathcal{G}}\}), \tag{7}$$

where $\mathbf{G} \in \mathbb{R}^{d\times d}$ is a learnable weight matrix shared by all input graphs.

3.4 Counter Module and Overall Objective

With the whole-graph representations of the query and input graph, we first capitalize on a counter module to estimate the count of subgraph isomorphisms, and then design the overall objective.

Counter module. We estimate the count of isomorphisms based on the structure matchability between the query and input graph. Formally, given the query graph $\mathcal{Q}$ and input graph $\mathcal{G}$, we predict the number of subgraphs of $\mathcal{G}$ which are isomorphic to $\mathcal{Q}$ by

$$\hat{n}(\mathcal{Q}, \mathcal{G}) = \textsc{ReLU}(\mathbf{w}^\top \textsc{Match}(\mathbf{h}_{\mathcal{Q}}, \mathbf{h}_{\mathcal{G}}^{\mathcal{Q}}) + b), \tag{8}$$

where $\textsc{Match}(\cdot,\cdot)$ outputs a d_m-dimensional vector to represent the matchability between its arguments, and $\mathbf{w} \in \mathbb{R}^{d_m}, b \in \mathbb{R}$ are the learnable weight vector and bias, respectively. Here a ReLU activation is used to ensure that the prediction is non-negative. Note that $\textsc{Match}(\cdot,\cdot)$ can be any function; in our implementation, we adopt a fully connected layer (FCL) and materialize it as $\textsc{Match}(\mathbf{x}, \mathbf{y}) = \text{FCL}(\mathbf{x} \parallel \mathbf{y} \parallel \mathbf{x} - \mathbf{y} \parallel \mathbf{x} \odot \mathbf{y})$.

Overall objective. Based on the counter module, we formulate the overall training objective. Assume a set of training triples $\mathcal{T} = \{(\mathcal{Q}_i, \mathcal{G}_i, n_i) | i = 1, 2, \ldots\}$, where n_i is the ground truth count for query $\mathcal{Q}_i$ and input graph $\mathcal{G}_i$. The ground truth can be evaluated by classical exact algorithms (Cordella et al. (2004)). Subsequently, we minimize the following absolute loss during training.

$$\mathcal{L} = \tfrac{1}{|\mathcal{T}|} \textstyle\sum_{(\mathcal{Q}_i,\mathcal{G}_i,n_i)\in\mathcal{T}} |\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) - n_i| + \lambda \cdot \mathcal{L}_{\text{FiLM}} + \mu \cdot \|\Theta\|_2^2, \tag{9}$$

where $\mathcal{L}_{\text{FiLM}}$ is a regularizer on the FiLM factors and $\|\Theta\|_2^2$ is a L2-norm regularizer on the model parameters, and λ, μ are hyperparameters to control the weight of the regularizers. Specifically, the FiLM regularizer is designed to smooth the modulations to reduce overfitting, by encouraging less scaling and shifting as follows.

$$\mathcal{L}_{\text{FiLM}} = \textstyle\sum_{(\mathcal{Q}_i,\mathcal{G}_i,n_i)\in\mathcal{T}} \sum_{e\in E_{\mathcal{G}_i}} (\|\gamma_e\|_2^2 + \|\beta_e\|_2^2). \tag{10}$$

We also present the training algorithm as well as the complexity analysis in Appendix A.

Table 1: Summary of datasets.

	# Queries	# Graphs	# Triples	Avg($\lvert V_{\mathcal{Q}}\rvert$)	Avg($\lvert E_{\mathcal{Q}}\rvert$)	Avg($\lvert V_{\mathcal{G}}\rvert$)	Avg($\lvert E_{\mathcal{G}}\rvert$)	Avg(Counts)	Max($\lvert L\rvert$)	Max($\lvert L'\rvert$)
SMALL	75	6790	448,140	5.20	6.80	32.62	76.34	14.83	16	16
LARGE	122	3240	395,280	8.43	12.23	239.94	559.68	34.42	64	64
MUTAG	24	188	4,512	3.50	2.50	17.93	39.58	17.76	7	4

4 EXPERIMENTS

In this section, we evaluate the proposed model COUNT-GNN[1] for subgraph isomorphism counting, and further analyze various important aspects of the model.

4.1 EXPERIMENTAL SETUP

Datasets. We conduct the evaluation on three datasets, as summarized in Table 1. In particular, *SMALL* and *LARGE* are two synthetic datasets, which are generated by the query and graph generators presented by Liu et al. (2020). On the other hand, *MUTAG* (Zhang et al. (2018)) is a real-world dataset which consists of 188 nitro compound graphs. These graphs are taken as our input graphs, while we use the query generator (Liu et al. (2020)) to obtain the query graphs. As each dataset consists of multiple query and input graphs, we couple each query graph $\mathcal{Q}$ with an input graph $\mathcal{G}$ to form a training triple $(\mathcal{Q}, \mathcal{G}, n)$ with n denoting the ground-truth count given by an exact algorithm VF2 (Cordella et al. (2004)). More details of the datasets can be found in Appendix B.1.

Baselines. We compare the proposed COUNT-GNN with the state-of-the-art approaches from two main categories. (1) *Conventional GNNs*, including GCN (Kipf & Welling (2017)), GAT (Veličković et al. (2018)), GraphSAGE (Hamilton et al. (2017)), GIN (Xu et al. (2019)) and DiffPool (Ying et al. (2018)). They capitalize on node-centric message passing, followed by a readout function to obtain the whole-graph representation. Except DiffPool which utilizes a specialized hierarchical readout, we employ a sum pooling over the node representations for the readout in other GNNs. (2) *GNN-based isomorphism counting models*, including four variants proposed by Liu et al. (2020), namely RGCN-DN, RGCN-Sum, RGIN-DN, RGIN-Sum, as well as LRP (Zhengdao et al. (2020)). They are purposely designed GNNs for subgraph isomorphism counting, relying on different GNNs (*e.g.*, RGCN (Schlichtkrull et al. (2018)), RGIN (Xu et al. (2019)), or local relational pooling (Zhengdao et al. (2020))) for node representation learning, followed by a specialized readout suited for isomorphism matching, *e.g.*, DiamNet (Liu et al. (2020)). In particular, the two variants RGCN-DN and RGIN-DN utilize DiamNet, whereas RGCN-Sum and RGIN-Sum utilize the simple sum-pooling. Finally, we also compare to a classical approach VF2 (Cordella et al. (2004)), which evaluates exact counts as the ground truth. We provide further details and settings for the baselines in Appendix B.2.

Settings and parameters. For SMALL and LARGE datasets, we randomly sample 5000 triples for training, 1000 for validation, and the rest for testing. For MUTAG, due to its small size, we randomly sample 1000 triples for training, 100 for validation and the rest for testing. We also conduct experiments with different training splits to evaluate the performance in Appendix C.1. We report further parameter settings in Appendix B.3.

Evaluation. We employ mean absolute error (MAE) and Q-error (Zhao et al. (2021)) to evaluate the effectiveness of COUNT-GNN. While the widely used MAE measures the magnitude of error in the prediction, Q-error measures a form of relative error defined by $\max(\frac{n}{\hat{n}}, \frac{\hat{n}}{n})$, where n denotes the ground-truth count and $\hat{n}$ denotes the predicted count. Both metrics are better when smaller: the best MAE is 0 and the best Q-error is 1. We further report the inference time for all the approaches in order to evaluate their efficiency, while training time comparisons are included in Appendix C.2. We repeat all experiments with five runs, and report their average results and standard deviations.

4.2 PERFORMANCE EVALUATION

To comprehensively evaluate the performance, we compare COUNT-GNN with the baselines in two settings: (1) a main setting with triples generated by all the query graphs and input graphs; (2) a

[1]Code and data can be found in Supplementary Materials for review.

Table 2: Effectiveness and efficiency evaluation in the main setting. VF2 generates exact ground-truth counts, thus with the perfect MAE (0) and Q-error (1). Time refers to the total inference time on all test triples, in seconds. Except VF2, the best method is bolded and the runner-up is underlined.

Methods	SMALL			LARGE			MUTAG		
	MAE ↓	Q-error ↓	Time (s) ↓	MAE ↓	Q-error ↓	Time (s) ↓	MAE ↓	Q-error ↓	Time (s) ↓
GCN	14.8 ± 0.5	2.1 ± 0.1	7.9 ± 0.2	33.0 ± 0.4	3.5 ± 0.9	$\underline{29.8} \pm 0.7$	19.9 ± 9.7	4.2 ± 1.5	0.88 ± 0.02
GRAPHSAGE	14.0 ± 2.7	2.5 ± 0.8	$\mathbf{7.0} \pm 0.1$	33.8 ± 1.6	$\underline{3.1} \pm 0.4$	$\mathbf{27.5} \pm 1.3$	13.8 ± 2.8	4.7 ± 0.8	0.88 ± 0.02
GAT	12.2 ± 0.7	2.9 ± 0.5	14.3 ± 0.3	37.3 ± 5.2	6.0 ± 1.2	59.4 ± 0.7	30.8 ± 6.7	6.0 ± 0.3	0.91 ± 0.01
DIFFPOOL	14.8 ± 2.6	2.1 ± 0.4	$\mathbf{7.0} \pm 0.1$	34.9 ± 1.4	3.8 ± 0.7	32.5 ± 0.7	$\underline{6.4} \pm 0.3$	2.5 ± 0.2	0.86 ± 0.00
GIN	12.6 ± 0.5	2.1 ± 0.1	$\underline{7.1} \pm 0.0$	35.8 ± 0.6	4.8 ± 0.2	33.5 ± 0.6	21.3 ± 1.0	5.6 ± 0.7	$\underline{0.41} \pm 0.01$
RGCN-SUM	24.2 ± 6.1	3.7 ± 1.2	13.2 ± 0.1	80.9 ± 26.3	6.3 ± 1.3	61.8 ± 0.2	8.0 ± 0.9	$\mathbf{1.5} \pm 0.1$	0.90 ± 0.01
RGCN-DN	16.6 ± 2.3	3.2 ± 1.3	48.1 ± 0.2	73.7 ± 29.2	9.1 ± 4.2	105.0 ± 0.4	7.3 ± 0.8	2.6 ± 0.2	1.19 ± 0.04
RGIN-SUM	$\underline{10.7} \pm 0.3$	$\underline{2.0} \pm 0.2$	12.2 ± 0.0	33.2 ± 2.2	4.2 ± 1.3	61.4 ± 1.0	10.8 ± 0.9	1.9 ± 0.1	0.45 ± 0.02
RGIN-DN	11.6 ± 0.2	2.4 ± 0.0	49.7 ± 1.8	$\underline{32.5} \pm 1.9$	4.3 ± 2.0	104.0 ± 1.5	8.6 ± 1.9	3.3 ± 0.8	0.73 ± 0.03
COUNT-GNN	$\mathbf{8.5} \pm 0.0$	$\mathbf{1.4} \pm 0.1$	7.9 ± 0.3	$\mathbf{30.9} \pm 4.3$	$\mathbf{2.5} \pm 0.5$	59.2 ± 1.7	$\mathbf{4.2} \pm 0.1$	$\underline{1.8} \pm 0.0$	$\mathbf{0.02} \pm 0.00$
VF2	0	1	1267.5 ± 2.7	0	1	12734.6 ± 5.9	0	1	1.52 ± 0.04

Table 3: Effectiveness and efficiency evaluation in the secondary setting. Time refers to the total inference time on all test triples, in seconds. The better method for each query is bolded.

		SMALL			LARGE			MUTAG		
		MAE ↓	Q-error ↓	Time (s) ↓	MAE ↓	Q-error ↓	Time (s) ↓	MAE ↓	Q-error ↓	Time (s) ↓
$\mathcal{Q}_1$	LRP	11.5 ± 0.4	3.6 ± 0.5	0.13 ± 0.00	126.1 ± 4.9	38.3 ± 1.1	$\mathbf{0.04} \pm 0.00$	12.3 ± 0.2	2.1 ± 0.0	0.01 ± 0.00
	COUNT-GNN	$\mathbf{3.0} \pm 0.2$	$\mathbf{1.4} \pm 0.3$	$\mathbf{0.04} \pm 0.00$	$\mathbf{111.2} \pm 0.8$	$\mathbf{2.9} \pm 0.1$	0.22 ± 0.01	$\mathbf{2.5} \pm 0.2$	$\mathbf{1.2} \pm 0.1$	$\mathbf{0.00} \pm 0.01$
$\mathcal{Q}_2$	LRP	12.6 ± 2.4	4.6 ± 0.9	0.12 ± 0.01	19.8 ± 1.4	3.7 ± 0.6	$\mathbf{0.04} \pm 0.00$	7.8 ± 0.4	2.9 ± 0.8	0.01 ± 0.00
	COUNT-GNN	$\mathbf{4.6} \pm 1.4$	$\mathbf{1.1} \pm 0.2$	$\mathbf{0.05} \pm 0.01$	$\mathbf{4.3} \pm 2.4$	$\mathbf{1.1} \pm 0.1$	0.07 ± 0.00	$\mathbf{5.0} \pm 0.2$	$\mathbf{2.1} \pm 0.1$	$\mathbf{0.01} \pm 0.00$
$\mathcal{Q}_3$	LRP	31.5 ± 3.1	4.1 ± 0.6	0.05 ± 0.01	87.2 ± 2.9	7.1 ± 0.8	$\mathbf{0.04} \pm 0.00$	8.3 ± 0.4	2.8 ± 0.1	0.01 ± 0.00
	COUNT-GNN	$\mathbf{23.2} \pm 2.8$	$\mathbf{1.3} \pm 0.2$	$\mathbf{0.03} \pm 0.00$	$\mathbf{58.0} \pm 2.4$	$\mathbf{1.8} \pm 0.1$	0.08 ± 0.00	$\mathbf{4.3} \pm 0.2$	$\mathbf{1.8} \pm 0.1$	$\mathbf{0.01} \pm 0.00$
Avg.	LRP	18.5	4.1	0.10	77.7	16.4	**0.04**	9.5	2.6	0.01
	COUNT-GNN	**10.3**	**1.3**	**0.04**	**57.8**	**1.9**	0.12	**3.9**	**1.7**	0.01

secondary setting for triples generated by all input graphs associated with only one query graph. Note that the main setting represents the more general scenarios, in which we compare with all baselines except LRP. However, due to the particular design of LRP that requires a number of input graphs coupled with one query graph and the corresponding ground-truth count during training, we use the secondary setting only for this baseline. Our model can flexibly work in both settings.

Main setting. As discussed, we compare COUNT-GNN with all baselines except LRP in this more general scenario, where the triples are generated by coupling every pair of query and input graphs. We report the results in Table 2, and make the following observations. Firstly, in terms of effectiveness measured by MAE and Q-error, COUNT-GNN consistently outperforms all the GNN-based models. The only exception appears on the Q-error of MUTAG, where COUNT-GNN still emerges as a competitive runner-up. This demonstrates the two key modules of COUNT-GNN, namely, edge-centric aggregation and query-conditioned graph modulation, can improve structure matching between input graphs and structurally diverse queries. Secondly, in terms of efficiency, COUNT-GNN achieves 76x$\sim$215x speedup over the classical VF2. Moreover, the speedup over the fastest RGCN/RGIN variant (*i.e.*, RGIN-Sum) are moderate but still range up to 23x whilst reducing the MAE and Q-error by 20% or more in most cases. In contrast, conventional GNNs can run faster than COUNT-GNN on LARGE and comparably on SMALL, but the efficiency comes at the expense of much worse MAE and Q-error than COUNT-GNN, by at least 30% in most cases. Thirdly, VF2 achieves the perfect MAE and Q-error (*i.e.*, 0 and 1 respectively), given that it is an exact method based on which the ground-truth counts are obtained. In summary, COUNT-GNN can achieve very significant speedup while maintaining a strong level of effectiveness.

Secondary setting. We also generate another group of triples for comparison with the baseline LRP, in the so-called secondary setting as discussed earlier. In particular, for each dataset we select three query graphs of different size (see details of the selected queries in Appendix B.1). On each dataset, we couple each query graph with all the input graphs, thus forming 6790/3240/188 triples for each query in SMALL/LARGE/MUTAG, respectively. Besides, we split the triples of SMALL and LARGE in the ratio of 1:1:2 for training, validation and testing respectively, while use the ratio of 1:1:1 for MUTAG. The results are reported in Table 3. We observe that COUNT-GNN persistently

outperforms LRP across the three datasets in terms of effectiveness, significantly reducing MAE by 43% and Q-error by 64% on average. This demonstrates again the power of the two key modules in COUNT-GNN. In terms of efficiency, neither COUNT-GNN nor LRP emerges as the clear winner.

4.3 MODEL ANALYSIS

In this section, we conduct a further analysis on COUNT-GNN. In particular, we present ablation study and parameters sensitivity here, and leave the rest (including comparison with different training size, comparison of offline time) into Appendix C due to the space limitation.

Ablation Study. To evaluate the impact of each component in COUNT-GNN, we conduct an ablation study by comparing COUNT-GNN with its two degenerate variations, including the version by replacing edge-centric aggregation with node-centric GIN aggregation (COUNT-GNN\E), and the version by replacing the query-conditioned modulation with sum-pooling upon the edges (COUNT-GNN\M), and show the results in Fig. 3. We have the following observations. Firstly, the whole model COUNT-GNN generally outperforms the two variations in most cases, except MAE on LARGE dataset. This shows that removing either module from the whole COUNT-GNN would impair the performance, further demonstrating the necessity of edge-centric aggregation mechanism and query-based modulation. Secondly, it is interesting that COUNT-GNN\M is usually better than COUNT-GNN\E in terms of both MAE and Q-error. This demonstrates that the edge-centric may contribute more to the performance boost of subgraph isomorphism counting, possibly due to the key fact that treating the edges as first class citizens for representation learning can capture fine-grained structural information.

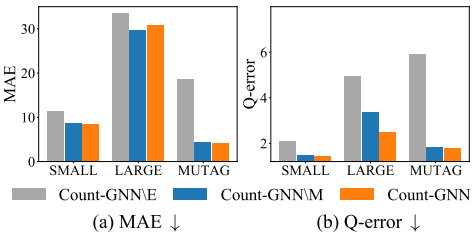


Figure 3: Ablation study of COUNT-GNN.

Parameters Sensitivity. We study the sensitivity of two important hyper-parameters on SMALL dataset. In Fig. 4(a), we increase the total number of GNN layers K for edge-centric aggregation, to check its influence on the performance. As K increases, the performance in terms of MAE and Q-error generally become better, only with one exception on Q-error when $K = 4$. This shows a phenomena that the increase of layers may facilitate the exploitation of long-range structural information, which might further help the model to achieve a clearer view of the structures in the graph. In Fig. 4(b), we show the sensitivity of parameter λ, which weights the regularizer on the FiLM factors in Eq. (9). We observe that λ is a bit sensitive to the performance, and $\lambda = 0.01$ may result an inferior performance. Interval [1e-5, 1e-3] might be a good range for superior performance.

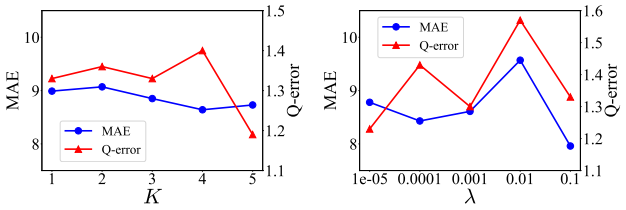


Figure 4: Parameters sensitivity on dataset SMALL.

5 RELATED WORK

Graph representation learning. Graph representation learning (Perozzi et al. (2014); Grover & Leskovec (2016); Tang et al. (2015)) usually capitalizes on sub-structures sampling on graph to represent the local view of graph structures, thus an encoder can be further utilized to embed nodes into low-dimensional representations, in which the graph structures are preserved. More recently, graph neural networks (GNNs) (Kipf & Welling (2017); Hamilton et al. (2017); Veličković et al. (2018); Xu et al. (2019)) arise as a powerful family of representation learning approaches, which rely on the key operator of neighborhood aggregation to pass messages recursively for node representation learning, thus the graph structure and content information can be preserved simultaneously.

Graph isomorphism counting. Graphs usually retain abundant local structures to depict specific frames in accordance with particular semantics, which gives rise to a high demand of subgraph isomorphism counting (Ullmann (1976)). To solve this proble, most traditional methods resort to backtracking (Ullmann (1976); Cordella et al. (2004); He & Singh (2008)). Though they can achieve precise results, the searching space usually grows exponentially as the increase of graph size, resulting in an intractable fact that subgraph isomorphism Counting being an #P-complete problem with high cost (Ullmann (1976)). Subsequently, several approaches (Han et al. (2013); Carletti et al. (2017)) are proposed to utilize some constraints towards reducing the searching space, and others (Yan et al. (2004)) try to filter out unnecessary graphs to speed up the backtracking process. Another line of approaches (Alon et al. (1995); Bressan et al. (2021)) rely on the color coding for subgraph isomorphism counting in polynomial time. They are usually fixed parameter tractable and can only be employed for some limited subcases. Teixeira et al. (2020) transform the subgraph counting to edge sum over a higher-order graph that only provides neighborhood query access for large-real-world input graphs, and work (Pinar et al. (2017)) is built on the idea of cutting a pattern into smaller ones, and using counts of smaller patterns to get larger counts. Though more efforts (Teixeira et al. (2018); Wang et al. (2014)) have also been devoted, these attempts still face the high cost issue.

Recently, a few studies (Liu et al. (2020); Zhengdao et al. (2020)) propose to address the subgraph isomorphism counting from the perspective of machining learning. Liu et al. (2020) propose to incorporate several existing pattern extraction mechanisms such as CNN (LeCun et al. (1998)), GRU (Chung et al. (2014)) and GNNs on both query graph and input graph for structural information exploitation, then a counter module is attached to summarize the number of isomorphisms. Zhengdao et al. (2020) analyze the ability of GNNs in detecting subgraph isomorphism, and propose a Local Relational Pooling model based on the permutations of walks according to BFS to count certain queries on graphs. However, they should create a new model for each query subgraph, limiting their usage. Compared to traditional methods, these GNN-based models usually approximate the counting with a tolerable error, yet can significantly save computation resources and time, providing a trade-off between the accuracy and cost. However, the node-centric nature of these GNNs-based models limits their ability to capture the fine-grained structural information. Bouritsas et al. (2020) propose Graph Substructure Networks (GSN), a topology-aware message passing scheme based on substructure encoding, to serve as structural features to enhance the expressive power of GNNs, yet not targeting at the problem of graph isomorphism counting itself.

Other related studies. Graph similarity search (Bai et al. (2019); Li et al. (2019)) addresses a distinct problem that calculating the similarity between graphs. Though some recent studies (Wang et al. (2019a; 2021b); Bai et al. (2021)) are designed by combining both the traditional models and deep learning models, they cannot be directly employed to cope with the problem of subgraph isomorphism counting since they focus on a distinct problem. Object detection (Redmon et al. (2016); Zhao et al. (2019)) is a hot topic in research field of computer vision, which is a bit similar to subgraph isomorphism counting. Yet these approaches usually do not consider the graph structure stemming from dependencies between instances. Some recent studies (Wang et al. (2021a); Jiang et al. (2020); Yang & Li (2020); Monti et al. (2018b)) resort to edge-oriented neighborhood aggregation to facilitate the node representation learning, or further graph representation learning based on the achieved node representations. But they are not particularly devised for subgraph isomorphism counting.

6 CONCLUSIONS

In this paper, we proposed a novel GNN called COUNT-GNN to address the problem of subgraph isomorphic counting on labeled graphs. COUNT-GNN is equipped with two key modules, namely, edge-centric message passing and query-conditioned graph modulation, to improve structure matching between the query and input graphs. On one hand, the module of edge-centric message passing operates at the edge level, which propagates and aggregates messages on and for edges following the edge adjacency, in order to capture fine-grained structural information. On the other hand, the module of query-conditioned graph modulation operates at the graph level, which adapts the input graph to suit each query individually, in order to improve the matching with specific structures in each query. To demonstrate the effectiveness and efficiency of COUNT-GNN, we conduct extensive experiments on a number of benchmark datasets. Results show that the proposed COUNT-GNN achieves superior performance in comparison to the state-of-the-art baselines.

REFERENCES

Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4): 844–856, 1995.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 384–392, 2019.

Yunsheng Bai, Derek Xu, Yizhou Sun, and Wei Wang. Glsearch: Maximum common subgraph detection via learning to search. In *International Conference on Machine Learning*, pp. 588–598. PMLR, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

Marco Bressan, Stefano Leucci, and Alessandro Panconesi. Faster motif counting via succinct color coding and adaptive sampling. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(6):1–27, 2021.

Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):804–818, 2017.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiao-Li Li. Semantic proximity search on graphs with metagraph-based learning. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 277–288. IEEE, 2016.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pp. 855–864, 2016.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pp. 1024–1034, 2017.

Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 337–348, 2013.

Huahai He and Ambuj K Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 405–418, 2008.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Xiaodong Jiang, Ronghang Zhu, Sheng Li, and Pengsheng Ji. Co-embedding of nodes and edges with graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pp. 3734–3743. PMLR, 2019.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pp. 3835–3845. PMLR, 2019.

Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1959–1969, 2020.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Federico Monti, Karl Otness, and Michael M Bronstein. MotifNet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pp. 225–228. IEEE, 2018a.

Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-primal graph convolutional networks. *arXiv preprint arXiv:1806.00770*, 2018b.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *KDD*, pp. 701–710, 2014.

Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*, pp. 1431–1440, 2017.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Aravind Sankar, Xinyang Zhang, and Kevin Chen-Chuan Chang. Meta-GNN: metagraph neural network for semi-supervised learning in attributed heterogeneous information networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 137–144, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. PathSim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pp. 1067–1077, 2015.

Carlos HC Teixeira, Leornado Cotta, Bruno Ribeiro, and Wagner Meira. Graph pattern mining and learning through user-defined relations. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1266–1271. IEEE, 2018.

Carlos HC Teixeira, Mayank Kakodkar, Vinícius Dias, Wagner Meira Jr, and Bruno Ribeiro. Sequential stratified regeneration: Mcmc for large state spaces with an application to subgraph count estimation. *arXiv preprint arXiv:2012.03879*, 2020.

Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1): 31–42, 1976.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

Hongwei Wang, Hongyu Ren, and Jure Leskovec. Relational message passing for knowledge graph completion. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1697–1707, 2021a.

Pinghui Wang, John CS Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(2):1–27, 2014.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3056–3065, 2019a.

Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5241–5250, 2021b.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *TheWebConf*, pp. 2022–2032, 2019b.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *TNNLS*, (Early Access), 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: A frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 335–346, 2004.

Yulei Yang and Dongsheng Li. Nenn: Incorporate node and edge features in graph neural networks. In *Asian Conference on Machine Learning*, pp. 593–608. PMLR, 2020.

Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6656–6663, 2020.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural Information Processing Systems*, 31:4800–4810, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. A learned sketch for subgraph counting. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 2142–2155, 2021.

Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

Chen Zhengdao, Chen Lei, Villar Soledad, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 2020.

A ALGORITHM AND COMPLEXITY ANALYSIS

Algorithm. We present the algorithm for training of COUNT-GNN in Alg. 1. In line 1, we initial all the parameters, as well as objective $\mathcal{L}$. In lines 3-13, we accumulate the loss for the given training tuples. In particular, in lines 4-8, we conduct the recursive edge-centric aggregation. In lines 5-7, we calculate the edge-centric representation for each edge. Then, in lines 9 and 10, we form the graph representations by aggregating all the inclusive edge-centric representations for query graph and input graph, respectively. In line 11, a counter module is employed to predict the number of subgraphs of $\mathcal{G}_i$ which are isomorphic to $\mathcal{Q}_i$. In line 12, we accumulate the loss. In line 14, we form the overall objective. Finally, in line 15 we optimize the model by minimizing objective $\mathcal{L}$.

Algorithm 1 MODEL TRAINING FOR COUNT-GNN

Input: Training tuples $\mathcal{T} = \{(\mathcal{Q}_i, \mathcal{G}_i, n_i) | i = 1, 2, \ldots\}$, total layers number K, hyper-parameters λ, μ.
Output: Model parameters Θ.

1: $\Theta \leftarrow$ parameters initialization, $\mathcal{L} \leftarrow 0$;
2: **while** not converged **do** ▷ Training iteration
3: **for** each triple $(\mathcal{Q}_i, \mathcal{G}_i, n_i) \in \mathcal{T}$ **do**
4: **for** each layer $l \in \{1, \ldots, K\}$ **do**
5: **for** each directed edge $\langle u, v \rangle \in E_{\mathcal{Q}_i}$ or $E_{\mathcal{G}_i}$ **do**
6: $\mathbf{h}^l_{\langle u,v \rangle} \leftarrow \sigma(\mathbf{W}^l \mathbf{h}^{l-1}_{\langle u,v \rangle} + \mathbf{U}^l \mathbf{h}^{l-1}_{\langle \cdot,u \rangle} + \mathbf{b}^l)$; ▷ Edge-centric aggregation, Eq. (1)
7: **end for**
8: **end for**
9: $\mathbf{h}_{\mathcal{Q}_i} \leftarrow \sigma(\mathbf{Q} \cdot \text{AGGR}(\{\mathbf{h}_e | e \in E_{\mathcal{Q}_i}\})$; ▷ Query graph representation, Eq. (3)
10: $\mathbf{h}^{\mathcal{Q}_i}_{\mathcal{G}_i} \leftarrow \sigma(\mathbf{G} \cdot \text{AGGR}(\{\tilde{\mathbf{h}}_e | e \in E_{\mathcal{G}_i}\})$; ▷ Input graph representation, Eq. (7)
11: $\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) \leftarrow \text{RELU}(\mathbf{w}^\top \text{MATCH}(\mathbf{h}_{\mathcal{Q}_i}, \mathbf{h}^{\mathcal{Q}_i}_{\mathcal{G}_i}) + b)$; ▷ Counter, Eq. (8)
12: $\mathcal{L} \leftarrow \mathcal{L} + |\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) - n_i|$; ▷ Loss accumulation
13: **end for**
14: $\mathcal{L} \leftarrow \mathcal{L} + \lambda \cdot \mathcal{L}_{\text{FiLM}} + \mu \cdot \|\Theta\|_2^2$; ▷ Overall objective, Eq. (9)
15: Update Θ by minimize $\mathcal{L}$;
16: **end while**
17: **return** Θ.

Complexity analysis. The edge-centric aggregation increases the computation cost. Here, given a tuple $(\mathcal{Q}, \mathcal{G}, n)$, we split COUNT-GNN into two parts for complexity analysis, *i.e.*, edge-centric aggregation, and query-conditioned graph modulation. **(1) Edge-centric aggregation.** Supposing the average degree on $\mathcal{Q}$ and $\mathcal{G}$ is $\bar{d}$. In each edge-centric aggregation layer, each edge would access its $\bar{d}$ neighboring edges for aggregation, thus involving complexity $O(\bar{d})$. For query graph $\mathcal{Q}$ with a total of K layers, the complexity for the edge representation learning is $O(\bar{d}^K \cdot |E_{\mathcal{Q}}|)$. Similarly, the complexity for the edge representation learning of input graph $\mathcal{G}$ is $O(\bar{d}^K \cdot |E_{\mathcal{G}}|)$. **(2) Query-conditioned graph modulation.** For query graph $\mathcal{Q}$, the calculation of graph representation involves complexity of $O(|E_{\mathcal{Q}}|)$. For input graph $\mathcal{Q}$, it first calculates the query-conditioned modulation for all edges with a complexity of $O(|E_{\mathcal{G}}|)$; then the calculation of graph representation has complexity of $O(|E_{\mathcal{G}}|)$. The prediction w.r.t. the calculated representation of query graph and input graph has complexity of $O(1)$. In summary, the prediction for tuple $(\mathcal{Q}, \mathcal{G}, n)$ has complexity of $O(\bar{d}^K \cdot |E_{\mathcal{Q}}| + \bar{d}^K \cdot |E_{\mathcal{G}}| + |E_{\mathcal{Q}}| + 2|E_{\mathcal{G}}|)$.

B FURTHER DETAILS OF EXPERIMENTAL SETUP

In this section, we give further details for the experimental setup.

B.1 DETAILS OF DATASETS

Data generation. We resort to the data generators in work (Liu et al. (2020)) to generate the three datasets (for MUTAG, only the query graphs), by using the same parameter settings. The detailed settings in data generation for SMALL and LARGE are illustrated in Table 4. In particular, when to generate one query graph or input graph, we first randomly sample the size parameters from the corresponding sets in Table 4, to constrain the generation of this graph. Note that, with generally

Table 4: Parameters for the data generation of SMALL and LARGE.

	Parameters	SMALL	LARGE
Query graph $\mathcal{Q}$	$\lvert V_{\mathcal{Q}}\rvert$	{3, 4, 8}	{3, 4, 8, 16}
	$\lvert E_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
	$\lvert L_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
	$\lvert L'_{\mathcal{Q}}\rvert$	{2, 4, 8}	{2, 4, 8, 16}
Input graph $\mathcal{G}$	$\lvert V_{\mathcal{G}}\rvert$	{8, 16, 32, 64}	{64, 128, 256, 512}
	$\lvert E_{\mathcal{G}}\rvert$	{8, 16, ..., 256}	{64, 128, ..., 2048}
	$\lvert L_{\mathcal{G}}\rvert$	{4, 8, 16}	{16, 32, 64}
	$\lvert L'_{\mathcal{G}}\rvert$	{4, 8, 16}	{16, 32, 64}

larger parameter sizes, the dataset LARGE would have larger individual graph sizes than dataset SMALL, as illustrated in Table 1.

Query selection for secondary setting in experiments. Let N and E denote the number of nodes and directed edges, respectively; for dataset SMALL, we randomly select three query graphs in the size of (N3, E3), (N4, E4) and (N8, E8), respectively; for dataset LARGE, we randomly select three query graphs in the size of (N4, E4), (N8, E8), and (N16, E16), respectively; for dataset MUTAG, we randomly select three query graphs in the size of (N3, E2), (N4, E3) and (N4, E3), respectively.

B.2 Details and Settings of Baselines

We compare COUNT-GNN with the state-of-the-art approaches from two main categories.

(1) *Conventional GNNs*, including GCN (Kipf & Welling (2017)), GAT (Veličković et al. (2018)), GraphSAGE (Hamilton et al. (2017)), GIN (Xu et al. (2019)) and DiffPool (Ying et al. (2018)). They usually capitalize on node-centric message passing, followed by a readout function to obtain the whole-graph representation.

- GCN (Kipf & Welling (2017)): GCN usually resorts to mean-pooling based node-centric neighborhood aggregation to receive messages from the neighboring nodes for node representation learning.
- GAT (Veličković et al. (2018)): GAT also depends on node-centric neighborhood aggregation for node representation learning, while it can assign different weights to neighbors to reweight their contributions.
- GraphSAGE (Hamilton et al. (2017)): GraphSAGE has a similar neighborhood aggregation mechanism with GCN, while it focuses more on the information from the node itself.
- GIN (Xu et al. (2019)): GIN employs a SUM aggregator to replace the mean-pooling method in GCN to aggregate all the messages from neighboring nodes, which is demonstrated to be more powerful to capture the graph structures.
- DiffPool (Ying et al. (2018)): DiffPool depends on a GNN framework to further build its specific aggregation mechanism, by clustering nodes hierarchically to form the whole-graph representation.

(2) *GNN-based isomorphism counting models*, including four variants proposed by Liu et al. (2020), namely RGCN-DN, RGCN-Sum, RGIN-DN, RGIN-Sum, as well as LRP (Zhengdao et al. (2020)). They are purposely designed GNNs for subgraph isomorphism counting, relying on different GNNs (*e.g.*, RGCN (Schlichtkrull et al. (2018)), RGIN (Xu et al. (2019)), or local relational pooling (Zhengdao et al. (2020))) for node representation learning, followed by a specialized readout suited for isomorphism matching, *e.g.*, DiamNet (Liu et al. (2020)). In particular, the two variants RGCN-DN and RGIN-DN utilize DiamNet, whereas RGCN-Sum and RGIN-Sum utilize the simple sum-pooling.

Model settings. To achieve the optimal performance, we tune the hyper-parameters for all the baselines according to the proposed settings in literature. In particular, for conventional GNN models including GCN (Kipf & Welling (2017)), GAT (Veličković et al. (2018)), GraphSAGE (Hamilton et al. (2017)), GIN (Xu et al. (2019)) and DiffPool (Ying et al. (2018)), we set the number of total

layers as 3, hidden dimension as 128, and dropout rate as 0.2. In particular, for GAT, we use a self-attention mechanism with 4 heads; for GraphSAGE, we use the mean-pooling as the aggregator. For DiffPool, we set the ratio of nodes' number in consecutive layers as 0.1. For GNN-based isomorphism counting models, we follow the hyper-parameter settings in their original papers, with which the models can achieve the optimal performance. In particular, for RGCN-SUM, RGCN-DM, RGIN-SUM and RGIN-DM, we set the number of layers as 3, and the hidden dimension as 128.

B.3 Settings of Count-GNN

We tune several hyper-parameters for Count-GNN to achieve its optimal performance. In particular, we employ a Count-GNN with a total of 3 layers. Besides, on SMALL and LARGE datasets, we set the hidden dimension as 24, due to the fact that Count-GNN performs well even with low hidden dimensions, though it usually performs better with higher dimension which also costs more time. To find a balance between the accuracy and time cost, we choose this moderate dimension. On MUTAG, we set the hidden dimension as 12. In addition, we set the hyper-parameter λ for weighting the FiLM factors in Eq. (9) as 0.0001.

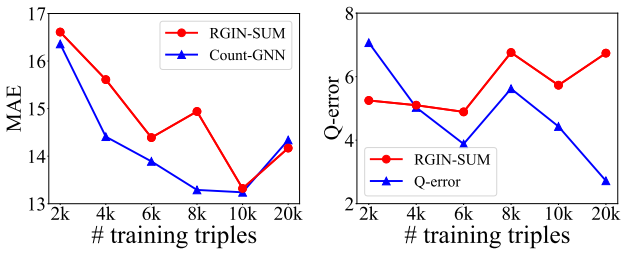
B.4 Other Details

Environment. We implemented the proposed Count-GNN using Pytorch 1.8.1 in Python 3.7.10. All experiments were conducted on a Linux workstation with a 16-core 3.50GHz CPU, 128GB DDR4 memory and one GeForce RTX 2080 Ti GPU.

C Further Model Analysis

C.1 Comparison with Different Training Size

To evaluate the performance tendency of Count-GNN in terms of different training size, we conduct an additional experiment by increasing the training triples from 2000 to 10,000, then to 20,000 on dataset SMALL. A baseline RGIN-SUM (Liu et al. (2020)) is also imported for comparison, which is proved to be competitive in the results of Table 2. Figs. 5(a) and 5(b) show the results of MAE and Q-error, respectively. We have several observations. Firstly, with different training size, the proposed model Count-GNN can consistently outperform baseline RGIN-SUM. The only exceptions lie in MAE with 20k and Q-error with 2k. This demonstrates that the performance of Count-GNN for subgraph isomorphism counting is stably superior to the baselines with different size of labeled data. Only when labeled data is too scarce or too sufficient its performance might be surpassed by the competitive baselines. Secondly, as the number of training triples increases, both MAE and Q-error have a tendency of decrease, showing that more labeled data would generally boost the model performance.



(a) MAE (b) Q-error

Figure 5: Comparison with different training size.

C.2 Comparison of Training Time

We conduct experiments for all the approaches to compare their training time. In particular, we show the training time per epoch for main setting in Table 5 and secondary setting in Table 6, respectively. In accordance with Tables 2 and 3, similar observations can be achieved that (1) in the main setting, our proposed Count-GNN generally occupies relative low training time on dataset SMALL and MUTAG, while having comparable training time with GNN-based isomorphism counting models on dataset LARGE; (2) in the secondary setting, Count-GNN usually costs a bit more training time than LRP.

Table 5: Comparison of training time in main setting.

Methods	SMALL	LARGE	MUTAG
GCN	0.8	1.1	0.35
GRAPHSAGE	0.8	1.0	0.35
GAT	1.0	2.4	0.39
DIFFPOOL	0.8	1.3	0.34
GIN	0.5	1.0	0.19
RGCN-SUM	1.0	2.4	0.38
RGCN-DN	1.5	3.7	0.50
RGIN-SUM	0.7	1.9	0.23
RGIN-DN	1.4	2.9	0.39
COUNT-GNN	0.4	2.5	0.04

Table 6: Comparison of training time in secondary setting with LRP.

		SMALL	LARGE	MUTAG
$\mathcal{Q}_1$	LRP	0.2	0.1	0.01
	COUNT-GNN	0.2	0.7	0.02
$\mathcal{Q}_2$	LRP	0.2	0.1	0.01
	COUNT-GNN	0.3	0.3	0.02
$\mathcal{Q}_3$	LRP	0.1	0.1	0.01
	COUNT-GNN	0.2	0.3	0.02