Learning to Count Isomorphisms with Graph Neural Networks

Xingtong Yu[1*], **Zemin Liu**[2*], **Yuan Fang**[3†], **Xinming Zhang**[1†]

[1] University of Science and Technology of China, China
[2] National University of Singapore, Singapore
[3] Singapore Management University, Singapore
yxt95@mail.ustc.edu.cn, zeminliu@nus.edu.sg, yfang@smu.edu.sg, xinming@ustc.edu.cn

Abstract

Subgraph isomorphism counting is an important problem on graphs, as many graph-based tasks exploit recurring subgraph patterns. Classical methods usually boil down to a backtracking framework that needs to navigate a huge search space with prohibitive computational costs. Some recent studies resort to graph neural networks (GNNs) to learn a low-dimensional representation for both the query and input graphs, in order to predict the number of subgraph isomorphisms on the input graph. However, typical GNNs employ a node-centric message passing scheme that receives and aggregates messages on nodes, which is inadequate in complex structure matching for isomorphism counting. Moreover, on 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, given that an edge is an atomic unit of encoding graph structures, we propose an *edge-centric message passing* scheme, where messages on edges are propagated and aggregated based on the edge adjacency to preserve fine-grained structural information. 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. Finally, we conduct extensive experiments on a number of benchmark datasets to demonstrate the superior performance of Count-GNN.

1 Introduction

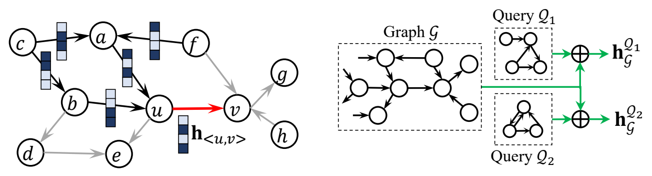
Research in network science and graph mining often finds and exploits recurring subgraph patterns on an input graph. For example, on a protein network, we could query for the hydroxy groups which consist of one oxygen atom covalently bonded to one hydrogen atom; on a social network, we could query for 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, Otness, and Bronstein 2018; Liu et al. 2018; Sankar, Zhang, and Chang 2019; Wang et al. 2019) have also been designed to 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). Classical methods usually employ search-based algorithms such as backtracking (Ullmann 1976; Cordella et al. 2004; He and Singh 2008) to exhaustively detect the isomorphisms and return an exact count. However, their computational costs are often excessive given that the detection problem is NP-complete and the counting form is #P-complete (Cordella et al. 2004). 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; Chen et al. 2020; Xia, Li, and Li 2022). They generally employ GNNs to embed the queries and input graphs 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 at the expense of approximation, providing a useful trade-off between accuracy and cost since many applications do not necessarily need an exact count.

However, previous GNN-based isomorphism counting models adopt a node-centric message-passing scheme, which propagates and aggregates messages on nodes. While this scheme is effective for node-oriented tasks, it falls short of matching complex structures for isomorphism counting. In particular, they rely on message aggregation to generate representations centering on nodes, failing to explicitly and fundamentally capture the complex interactions among nodes. Thus, as the first challenge, *how do we capture fine-grained structural information* beyond node-centric GNNs? Moreover, on an 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 graph representation to match with

[*]Co-first authors with equal contribution. Part of the work was done while at Singapore Management University.

[†]Corresponding authors.



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

Figure 1: Illustration of Count-GNN.

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 approximate 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 edges as first-class citizens can better capture fine-grained structural information. Theoretically, our proposed edge-centric GNN model can be regarded as a generalization of node-centric GNNs, with provably stronger expressive power than node-centric GNNs. At the graph level, Count-GNN resorts to a modulation mechanism (Perez et al. 2018) by adapting the input graph representation to each query graph, as shown in Fig. 1(b). As a result, the input graph can be tailored to each individual query with varying structures. Coupling the two perspectives, Count-GNN is able to precisely match complex structures between the input graph and structurally diverse queries.

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, which is more expressive than node-centric aggregation in theory. (2) Moreover, we design a 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 for isomorphism counting.

2 Related Work

We present the most related studies here, while leaving the rest to Appendix G due to the space limitation.

Graphs usually entail abundant local structures to depict particular semantics, which gives rise to the importance of subgraph isomorphism counting (Ullmann 1976). To solve this problem, most traditional methods resort to backtracking (Ullmann 1976; Cordella et al. 2004; He and Singh 2008). Although they can obtain the exact counts, the search space usually grows intractably as graph size increases. In fact, subgraph isomorphism counting is a #P-complete problem (Ullmann 1976). Subsequently, several approaches (Han, Lee, and Lee 2013; Carletti et al. 2017) are proposed to utilize some constraints to reduce the search space, and others (Yan, Yu, and Han 2004) try to filter out infeasible graphs to speed up the backtracking process. Another line of approaches (Alon, Yuster, and Zwick 1995; Bressan, Leucci, and Panconesi 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. Other studies perform count estimation (Teixeira et al. 2020; Pinar, Seshadhri, and Vishal 2017; Teixeira et al. 2018; Wang et al. 2014), such as in the setting where access to the entire network is prohibitively expensive (Teixeira et al. 2020), or using the counts of smaller patterns to estimate the counts for larger ones (Pinar, Seshadhri, and Vishal 2017). However, these attempts still face high computational costs.

Recently, a few studies (Liu et al. 2020; Chen et al. 2020) propose to address subgraph isomorphism counting from the perspective of machining learning. One study (Liu et al. 2020) proposes to incorporate several existing pattern extraction mechanisms such as CNN (LeCun et al. 1998), GRU (Chung et al. 2014) and GNNs (Wu et al. 2020) on both query graphs and input graphs to exploit their structural match, which is then followed by a counter module to predict the number of isomorphisms. Another work (Chen et al. 2020) analyzes the ability of GNNs in detecting subgraph isomorphism, and proposes a Local Relational Pooling model based on the permutations of walks according to breadth-first search to count certain queries on graphs. However, they need to learn a new model for each query graph, limiting their practical application. Compared to traditional methods, these machine learning-based models can usually approximate the counts reasonably well, and at the same time significantly save computational resources and time, providing a practical trade-off between accuracy and cost. However, these approaches only adopt node-centric GNNs, which limit their ability to capture finer-grained structural information.

There also exist a few recent GNNs based on edge-centric aggregation (Monti et al. 2018; Jiang et al. 2020), node-centric aggregation with the assistance of edge features (Gong and Cheng 2019; Yang and Li 2020; Isufi, Gama, and Ribeiro 2021), or both node- and edge-centric aggregation at the same time (Liu and Song 2022). Except DMPNN (Liu and Song 2022), they are not specifically devised for subgraph isomorphism counting. In particular, they all lack the key module of structural matching between query and input graphs. Besides, the edge-centric approaches do not theoretically justify their enhanced expressive power compared to the node-centric counterparts.

3 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. 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 $E_\mathcal{S} \subseteq E_\mathcal{G}$.

Next, we present the definition of subgraph isomorphism on a labeled graph.

Definition 1 (Labeled Subgraph Isomorphism)**.** *Consider a subgraph $\mathcal{S}$ of some* input graph*, and a* query graph *$\mathcal{Q}$. $\mathcal{S}$ is* isomorphic *to $\mathcal{Q}$, 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 the 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 on $\mathcal{G}$ 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.*, $|V_\mathcal{Q}| \ll |V_\mathcal{G}|$ and $|E_\mathcal{Q}| \ll |E_\mathcal{G}|$, leading to a huge search space and computational cost.

4 Proposed Model: Count-GNN

In this section, we present the overall framework of Count-GNN first, followed by individual modules.

4.1 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 targets 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 representations of the input graph are modulated w.r.t. each query, as shown in Fig. 2(c). The query-conditioned edge representations then undergo a readout operation to fuse into a query-conditioned whole-graph representation for the input graph. The module targets 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. 2(d), a counter module is applied to predict the isomorphism counting on the input graph for a particular query, forming the overall objective.

4.2 Edge-Centric Aggregation

Typical GNNs (Kipf and Welling 2017; Veličković et al. 2018; Hamilton, Ying, and Leskovec 2017) and GNN-based isomorphism counting models (Liu et al. 2020; Chen 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 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 in an explicit and fundamental manner, given that edges represent the atomic unit of graph structures.

More concretely, 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 $e = \langle u, v\rangle$ on a graph (either an input or query graph), we initialize its message as a d_0-dimensional vector

$$\mathbf{h}^0_{\langle u,v\rangle} = \mathbf{x}_u \parallel \mathbf{x}_{\langle u,v\rangle} \parallel \mathbf{x}_v \in \mathbb{R}^{d_0}, \tag{1}$$

where $\mathbf{x}_*$ encodes the input features of the corresponding nodes or edges and $\parallel$ is 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 layer, in which each edge receives and aggregates messages along the directed edges. The edge-centric message passing can be made recursive by stacking multiple layers. Formally, in the l-th layer, the message on a directed edge $\langle u, v\rangle$, *i.e.*, $\mathbf{h}^l_{\langle u,v\rangle} \in \mathbb{R}^{d_l}$, is updated as

$$\mathbf{h}^l_{\langle u,v\rangle} = \sigma(\mathbf{W}^l\mathbf{h}^{l-1}_{\langle u,v\rangle} + \mathbf{U}^l\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} + \mathbf{b}^l), \tag{2}$$

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 materialized as

$$\mathbf{h}^{l-1}_{\langle \cdot,u\rangle} = \text{AGGR}(\{\mathbf{h}^{l-1}_{\langle i,u\rangle} | \langle i, u\rangle \in E\}), \tag{3}$$

where E denotes the set of directed edges in the graph, and $\text{AGGR}(\cdot)$ is an aggregation operator to aggregate messages from the preceding edges. We implement the aggregation operator as a simple mean, although more sophisticated approaches such as self-attention (Hamilton, Ying, and Leskovec 2017) and multi-layer perceptron (Xu et al. 2019) can also be employed. To boost the message passing, 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 multi-layer edge-centric aggregation is applied to each query and input graph in a dataset. 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 this edge, denoted as $\mathbf{h}_{\langle u,v\rangle} \in \mathbb{R}^d$.

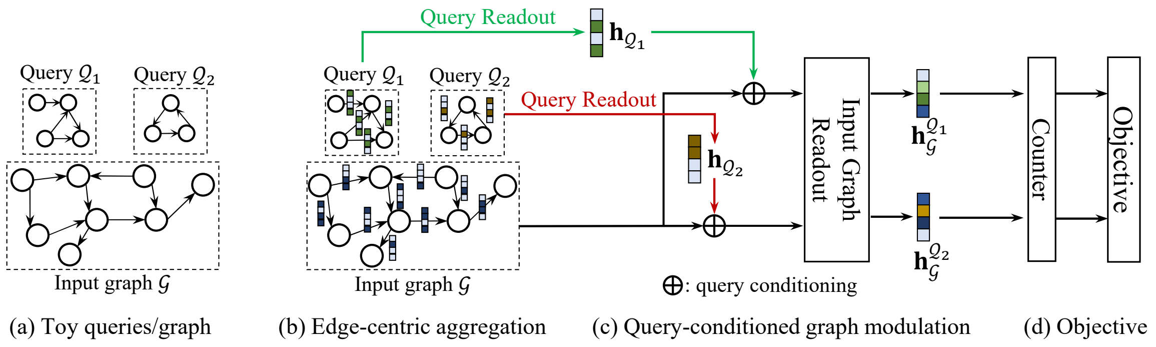

Query $\mathcal{Q}_1$
Query $\mathcal{Q}_2$
Input graph $\mathcal{G}$
(a) Toy queries/graph
Query Readout
$\mathbf{h}_{\mathcal{Q}_1}$
Query Readout
$\mathbf{h}_{\mathcal{Q}_2}$
(b) Edge-centric aggregation
Input Graph Readout
$\mathbf{h}_{\mathcal{G}}^{\mathcal{Q}_1}$
$\mathbf{h}_{\mathcal{G}}^{\mathcal{Q}_2}$
$\oplus$: query conditioning
(c) Query-conditioned graph modulation
Counter
Objective
(d) Objective

Figure 2: Overall framework of Count-GNN.

4.3 Query-Conditioned Graph Modulation

Beyond the edge level, Count-GNN fuses the edge representations into a whole-graph representation to facilitate structure matching between query and input graphs.

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

$$\mathbf{h}_{\mathcal{Q}} = \sigma(\mathbf{Q} \cdot \text{AGGR}(\{\mathbf{h}_{\langle u,v\rangle} | \langle u,v\rangle \in E_{\mathcal{Q}}\})), \tag{4}$$

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. To generate a whole-graph representation for the input graph, a straightforward way is to follow Eq. (4) by regarding all edges uniformly. However, for an input graph, the space of possible query graphs is enormous. In particular, 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 well for effective structure matching. Thus, we propose to modulate the input graph conditioned on the query, to adapt the whole-graph representation of the input graph to each query. To this end, we leverage Feature-wise Linear Modulation (FiLM) (Perez et al. 2018; Liu et al. 2021; Liu, Nguyen, and Fang 2021) on the edge representations in the input graph, conditioned on the query, in order to retain query-specific structures. The modulation is essentially a scaling and shifting transformation to adapt the edge representations of the input graph to the query. 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.

$$\tilde{\mathbf{h}}_{\langle u,v\rangle} = (\gamma_{\langle u,v\rangle} + \mathbf{1}) \odot \mathbf{h}_{\langle u,v\rangle} + \beta_{\langle u,v\rangle}, \tag{5}$$

where $\gamma_{\langle u,v\rangle}$ and $\beta_{\langle u,v\rangle} \in \mathbb{R}^d$ 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 $\gamma_{\langle u,v\rangle}$ and $\beta_{\langle u,v\rangle}$ are not directly learnable, but are instead generated by a secondary network (Ha, Dai, and Le 2017) conditioned on the original edge representation $\mathbf{h}_{\langle u,v\rangle}$ and the query representation $\mathbf{h}_{\mathcal{Q}}$. More specifically,

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

$$\beta_{\langle u,v\rangle} = \sigma(\mathbf{W}_\beta \mathbf{h}_{\langle u,v\rangle} + \mathbf{U}_\beta \mathbf{h}_{\mathcal{Q}} + \mathbf{b}_\beta), \tag{7}$$

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 generate a modulated whole-graph representation for the input graph, which is tailored toward each query to enable more precise matching between the input graph and query. Concretely, consider a query graph $\mathcal{Q}$ and an 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.

$$\mathbf{h}_{\mathcal{G}}^{\mathcal{Q}} = \sigma(\mathbf{G} \cdot \text{AGGR}(\{\tilde{\mathbf{h}}_{\langle u,v\rangle} | \langle u,v\rangle \in E_{\mathcal{G}}\})), \tag{8}$$

where $\mathbf{G} \in \mathbb{R}^{d\times d}$ is a learnable weight matrix shared by all input graphs, and we use sum for the aggregation.

4.4 Counter Module and Overall Objective

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

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

$$\hat{n}(\mathcal{Q},\mathcal{G}) = \text{ReLU}(\mathbf{w}^\top \text{Match}(\mathbf{h}_{\mathcal{Q}}, \mathbf{h}_{\mathcal{G}}^{\mathcal{Q}}) + b), \tag{9}$$

where $\text{Match}(\cdot,\cdot)$ 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 $\text{Match}(\cdot,\cdot)$ can be any function—we adopt a fully connected layer (FCL) such that $\text{Match}(\mathbf{x},\mathbf{y}) = \text{FCL}(\mathbf{x} \parallel \mathbf{y} \parallel \mathbf{x}-\mathbf{y} \parallel \mathbf{x}\odot\mathbf{y})$.

Overall objective. Based on the counter module, we formulate the overall training loss. Assume a set of training triples $\mathcal{T} = \{(\mathcal{Q}_i, \mathcal{G}_i, n_i) \mid i = 1, 2, \ldots\}$, 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 loss:

$$\frac{1}{|\mathcal{T}|} \sum_{(\mathcal{Q}_i,\mathcal{G}_i,n_i)\in\mathcal{T}} |\hat{n}(\mathcal{Q}_i, \mathcal{G}_i) - n_i| + \lambda\mathcal{L}_{\text{FiLM}} + \mu\|\Theta\|_2^2, \quad (10)$$

where $\mathcal{L}_{\text{FiLM}}$ is a regularizer on the FiLM factors and $\|\Theta\|_2^2$ is a L2 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.

$$\mathcal{L}_{\text{FiLM}} = \sum_{(\mathcal{Q}_i,\mathcal{G}_i,n_i)\in\mathcal{T}} \sum_{\langle u,v\rangle\in E_{\mathcal{G}_i}} \|\gamma_{\langle u,v\rangle}\|_2^2 + \|\beta_{\langle u,v\rangle}\|_2^2. \quad (11)$$

We also present the training algorithm and a complexity analysis in Appendix A.

4.5 Theoretical Analysis of Count-GNN

The proposed Count-GNN capitalizes on edge-centric message passing, which is fundamentally more powerful than conventional node-centric counterparts. This conclusion can be theoretically shown by the below lemma and theorem.

Lemma 1 (Generalization)**.** *Count-GNN can be reduced to a node-centric GNN, i.e., Count-GNN can be regarded as a generalization of the latter.* □

In short, Count-GNN can be reduced to a node-centric GNN by removing some input information and merging some edge representations. This demonstrates that Count-GNN is at least as powerful as the node-centric GNNs. We present the proof of Lemma 1 in Appendix B.

Theorem 1 (Expressiveness)**.** *Count-GNN is more powerful than node-centric GNNs, which means (i) for any two non-isomorphic graphs that can be distinguished by a node-centric GNN, they can also be distinguished by Count-GNN; and (ii) there exists two non-isomorphic graphs that can be distinguished by Count-GNN but not by a node-centric GNN.* □

Intuitively, edge-centric GNNs are capable of capturing fine-grained structural information, as any node can be viewed as a collapse of edges around the node. Therefore, by treating edges as the first-class citizens, Count-GNN becomes more powerful. The proof of Theorem 1 can be found in Appendix B.

5 Experiments

In this section, we empirically evaluate the proposed model Count-GNN in comparison to the state of the art.

	SMALL	LARGE	MUTAG	OGB-PPA
# Queries	75	122	24	12
# Graphs	6,790	3,240	188	6,000
# Triples	448,140	395,280	4,512	57,940
Avg($\|V_\mathcal{Q}\|$)	5.20	8.43	3.50	4.50
Avg($\|E_\mathcal{Q}\|$)	6.80	12.23	2.50	4.75
Avg($\|V_\mathcal{G}\|$)	32.62	239.94	17.93	152.75
Avg($\|E_\mathcal{G}\|$)	76.34	559.68	39.58	1968.29
Avg(Counts)	14.83	34.42	17.76	13.83
Max($\|L\|$)	16	64	7	8
Max($\|L'\|$)	16	64	4	1

Table 1: Summary of datasets.

5.1 Experimental Setup

Datasets. We conduct the evaluation on four datasets shown in Table 1. In particular, *SMALL* and *LARGE* are two synthetic datasets, which are generated by the query and graph generators presented by a previous study (Liu et al. 2020). On the other hand, *MUTAG* (Zhang et al. 2018) and *OGB-PPA* (Hu et al. 2020) are two real-world datasets. In particular, MUTAG consists of 188 nitro compound graphs, and OGB-PPA consists of 6,000 protein association graphs. While graphs in MUTAG and OGB-PPA are taken as our input graphs, we use the query generator (Liu et al. 2020) to generate 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 are given in Appendix C.

Baselines. We compare Count-GNN with the state-of-the-art approaches in two main categories. We provide further details and settings for the baselines in Appendix D.

(1) *Conventional GNNs*: GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017), DPGCNN (Monti et al. 2018), 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 (Chen et al. 2020) and DMPNN-LRP (Liu and Song 2022), a better variant of DMPNN. They are purposely designed GNNs for subgraph isomorphism counting, relying on different GNNs such as RGCN (Schlichtkrull et al. 2018), RGIN (Xu et al. 2019) and local relational pooling (Chen et al. 2020) for node representation learning, followed by a specialized readout suited for isomorphism matching. In particular, the two variants RGCN-DN and RGIN-DN utilize DiamNet (Liu et al. 2020), whereas RGCN-Sum and RGIN-Sum utilize the simple sum-pooling.

Finally, we also include a classical approach VF2

Methods	SMALL			LARGE			MUTAG			OGB-PPA		
	MAE↓	Q-error↓	Time/s↓	MAE↓	Q-error↓	Time/s↓	MAE↓	Q-error↓	Time/s↓	MAE↓	Q-error↓	Time/s↓
GCN	$14.8_{0.5}$	$2.1_{0.1}$	$7.9_{0.2}$	$33.0_{0.4}$	$3.5_{1.0}$	$\underline{29.8}_{0.7}$	$19.9_{9.7}$	$4.2_{1.5}$	$0.88_{0.02}$	$36.8_{1.4}$	$2.1_{0.4}$	$12.5_{0.3}$
GraphSAGE	$14.0_{2.7}$	$2.5_{0.8}$	$\mathbf{7.0}_{0.1}$	$33.8_{1.6}$	$\underline{3.1}_{0.4}$	$\mathbf{27.5}_{1.3}$	$13.9_{2.8}$	$4.7_{0.8}$	$0.88_{0.02}$	$32.5_{4.5}$	$2.5_{0.5}$	$\mathbf{11.1}_{0.1}$
GAT	$12.2_{0.7}$	$2.0_{0.5}$	$14.3_{0.3}$	$37.3_{5.2}$	$6.0_{1.2}$	$59.4_{0.7}$	$30.8_{6.7}$	$6.0_{0.3}$	$0.91_{0.01}$	$35.8_{2.4}$	$2.2_{0.6}$	$30.4_{0.8}$
DPGCNN	$16.8_{0.7}$	$2.9_{0.2}$	$21.7_{0.4}$	$39.8_{3.7}$	$5.4_{1.6}$	$64.8_{0.9}$	$27.5_{2.5}$	$4.9_{0.6}$	$1.54_{0.01}$	$38.4_{1.2}$	$2.3_{0.3}$	$19.4_{0.7}$
DiffPool	$14.8_{2.6}$	$2.1_{0.4}$	$\mathbf{7.0}_{0.1}$	$34.9_{1.4}$	$3.8_{0.7}$	$32.5_{0.7}$	$6.4_{0.3}$	$2.5_{0.2}$	$0.86_{0.00}$	$35.9_{4.7}$	$2.7_{0.3}$	$15.4_{2.2}$
GIN	$12.6_{0.5}$	$2.1_{0.1}$	$\underline{7.1}_{0.0}$	$35.9_{0.6}$	$4.8_{0.2}$	$33.5_{0.6}$	$21.3_{1.0}$	$5.6_{0.7}$	$0.41_{0.01}$	$34.6_{1.4}$	$2.5_{0.5}$	$\underline{12.3}_{0.4}$
RGCN-Sum	$24.2_{6.1}$	$3.7_{1.2}$	$13.2_{0.1}$	80.9_{26}	$6.3_{1.3}$	$61.8_{0.2}$	$8.0_{0.8}$	$\mathbf{1.5}_{0.1}$	$0.89_{0.01}$	$34.5_{13.6}$	$4.7_{0.8}$	$33.0_{0.2}$
RGCN-DN	$16.6_{2.3}$	$3.2_{1.3}$	$48.1_{0.2}$	73.7_{29}	$9.1_{4.2}$	$105.0_{0.4}$	$7.3_{0.8}$	$2.6_{0.2}$	$1.19_{0.04}$	$57.1_{15.7}$	$5.0_{1.3}$	$31.2_{0.1}$
RGIN-Sum	$10.7_{0.3}$	$2.0_{0.2}$	$12.2_{0.0}$	$33.2_{2.2}$	$4.2_{1.3}$	$61.4_{1.0}$	$10.8_{0.9}$	$1.9_{0.1}$	$0.45_{0.02}$	$29.1_{1.7}$	$2.2_{0.6}$	$21.0_{1.2}$
RGIN-DN	$11.6_{0.2}$	$2.4_{0.0}$	$49.7_{1.8}$	$32.5_{1.9}$	$4.3_{2.0}$	$104.0_{1.5}$	$8.6_{1.9}$	$3.3_{0.8}$	$0.73_{0.03}$	$35.8_{6.4}$	$4.4_{1.1}$	$28.8_{0.3}$
DMPNN-LRP	$\underline{9.1}_{0.2}$	$\underline{1.5}_{0.1}$	$32.4_{1.4}$	$\mathbf{28.1}_{1.3}$	$3.4_{1.5}$	$184.2_{1.8}$	$\underline{5.4}_{1.8}$	$\underline{1.8}_{1.0}$	$\underline{0.13}_{0.05}$	$\mathbf{25.6}_{4.9}$	$\underline{1.1}_{1.3}$	$63.0_{0.6}$
Count-GNN	$\mathbf{8.5}_{0.0}$	$\mathbf{1.4}_{0.1}$	$7.9_{0.3}$	$\underline{30.9}_{4.3}$	$\mathbf{2.5}_{0.5}$	$59.2_{1.7}$	$\mathbf{4.2}_{0.1}$	$\underline{1.8}_{0.0}$	$\mathbf{0.02}_{0.00}$	$\underline{28.7}_{3.9}$	$\mathbf{1.0}_{0.2}$	$18.1_{0.6}$
VF2	0	1	$1049.2_{2.7}$	0	1	$9270.5_{5.9}$	0	1	$1.30_{0.04}$	0	1	$5836.3_{4.8}$
Peregrine	-	-	$72.4_{2.0}$	-	-	$904.2_{4.5}$	-	-	$0.20_{0.03}$	-	-	$450.1_{3.9}$

Table 2: Evaluation in the main setting. VF2 generates the exact counts, giving a 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.

(Cordella et al. 2004) and a state-of-the-art approach Peregrine (Jamshidi, Mahadasa, and Vora 2020), both of which evaluate the exact counts. Note that there are many other exact approaches, but they are not suitable baselines due to their lack of generality. In particular, many approaches have algorithmic limitations that can only process small queries, *e.g.*, up to 4 nodes in PGD (Ahmed et al. 2015), 5 nodes in RAGE (Marcus and Shavitt 2012) and ORCA (Hočevar and Demšar 2014), and 6 nodes in acc-Motif (Meira et al. 2014). More broadly speaking, since the counting task is #P-complete, any exact algorithm is bound to suffer from prohibitively high running time once the queries become just moderately large. Besides, some approaches cannot handle certain kinds of queries or input graphs. For example, SCMD (Wang et al. 2012) and PATCOMP (Jain et al. 2017) are not applicable to directed graphs, and PGD (Ahmed et al. 2015) and RAGE (Marcus and Shavitt 2012) do not handle graphs with node or edge labels. Although Peregrine does not support directed edges or edge labels either, we run it on our datasets by ignoring edge directions/labels if any, and focus on its time cost only. In addition, there are also a number of statistically approximate methods, but they have similar shortcomings. For example, tMotivo and L8Motif (Bressan, Leucci, and Panconesi 2021) can only handle query graphs with no more than 8 or 16 nodes.

Data splits and settings. For the 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. For OGB-PPA, we divide the triples into training, validation and testing sets with a proportion of 4:1:5. We report the settings of Count-GNN in Appendix E.

Evaluation. We employ mean absolute error (MAE) and Q-error (Zhao et al. 2021) to evaluate the effectiveness of Count-GNN. The widely used metric MAE measures the magnitude of error in the prediction. In addition, Q-error measures a 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.[1] Both metrics are better when smaller: the best MAE is 0 while the best Q-error is 1. We further report the inference time for all the approaches to evaluate their efficiency on answering queries, as well as their training time. We repeat all experiments with five runs, and report their average results and standard deviations.

5.2 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 secondary setting with triples generated by all input graphs associated with only one query graph. Note that the main setting represents a more general scenario, 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.

Testing with 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 compare the aspects of effectiveness and efficiency. In terms of *effectiveness* measured by MAE and Q-error, Count-GNN can generally outperform other GNN-based models. In the several cases where Count-GNN is not the best among the GNN-based models, it still emerges as a competitive runner-up. This demonstrates the two key modules of Count-GNN,

[1] If the ground-truth or predicted count is less than 1, we assume a pseudocount of 1 for the calculation of q-error.

Methods	SMALL	LARGE	MUTAG	OGB-PPA
GCN	0.8	1.1	0.35	1.0
GraphSAGE	0.8	1.0	0.35	0.9
GAT	1.0	2.4	0.39	1.5
DiffPool	0.8	1.3	0.34	1.2
GIN	0.5	1.0	0.19	0.7
RGCN-SUM	1.0	2.4	0.38	1.7
RGCN-DN	1.5	3.7	0.50	2.2
RGIN-SUM	0.7	1.9	0.23	1.3
RGIN-DN	1.4	2.9	0.39	1.8
Count-GNN	0.4	2.5	0.04	1.3

Table 3: Comparison of training time per epoch, in seconds.

		SMALL			LARGE			MUTAG		
		MAE	Q-err	Time	MAE	Q-err	Time	MAE	Q-err	Time
$\mathcal{Q}_1$	LRP	11.5	3.6	0.13	126.1	38.3	**0.04**	12.3	2.1	0.01
	CG	**3.0**	**1.4**	**0.04**	**111.2**	**2.9**	0.22	**2.5**	**1.2**	**0.00**
$\mathcal{Q}_2$	LRP	12.6	4.6	0.12	19.8	3.7	**0.04**	7.8	2.9	0.01
	CG	**4.6**	**1.1**	**0.05**	**4.3**	**1.1**	0.07	**5.0**	**2.1**	**0.01**
$\mathcal{Q}_3$	LRP	31.5	4.1	0.05	87.2	7.1	**0.04**	8.3	2.8	0.01
	CG	**23.2**	**1.3**	**0.03**	**58.0**	**1.8**	0.08	**4.3**	**1.8**	**0.01**
Avg	LRP	18.5	4.1	0.10	77.7	16.4	**0.04**	9.5	2.6	0.01
	CG	**10.3**	**1.3**	**0.04**	**57.8**	**1.9**	0.12	**3.9**	**1.7**	0.01

Table 4: Evaluation in the secondary setting. Time is reported in seconds, on the total inference time on all test triples. CG is the abbreviation of Count-GNN.

namely, edge-centric aggregation and query-conditioned graph modulation, can improve structure matching between input graphs and structurally diverse queries. In terms of *efficiency* measured by the query time, we make three observations. First, Count-GNN achieves 65x$\sim$324x speedups over the classical VF2. While the other exact method Peregrine is orders of magnitude faster than VF2, Count-GNN can still achieve 8x$\sim$26x speedups over Peregrine. Second, Count-GNN is also more efficient than other GNN-based isomorphism counting models. On the one hand, Count-GNN achieves 3.1x$\sim$6.5x speedups over DMPNN-LRP, which is generally the second best GNN-based method after Count-GNN in terms of effectiveness. On the other hand, Count-GNN also obtains consistent and notable speedups over the fastest RGCN/RGIN variant (*i.e.*, RGIN-Sum), while reducing the errors by 20% or more in most cases. Finally, although many conventional GNNs can achieve a comparable or faster query time, they have much worse errors than Count-GNN, by at least 30% in most cases.

Furthermore, we compare the training time of GNN-based approaches in Table 3. Our proposed Count-GNN generally requires relatively low training time on SMALL and MUTAG, while having comparable training time to the baselines on LARGE and OGB-PPA.

Testing with secondary setting. We also generate another group of triples for comparison with the baseline LRP, in the so-called secondary setting due to the design requirement of LRP. In particular, we only evaluate on three datasets SMALL, LARGE and MUTAG, as LRP is out-of-memory on OGB-PPA with large and dense graphs. For each dataset we select three query graphs of different sizes (see Appendix C). 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, while using the ratio of 1:1:1 for MUTAG.

The results are reported in Table 4. We observe that Count-GNN consistently outperforms LRP in terms of effectiveness, significantly reducing MAE by 43% and Q-error by 64% on average. This verifies again the power of the two key modules in Count-GNN. For efficiency, neither Count-GNN nor LRP emerges as the clear winner.

Methods	SMALL		LARGE		MUTAG	
	MAE	Q-error	MAE	Q-error	MAE	Q-error
Count-GNN\E	11.3	2.07	33.58	4.96	18.63	5.92
Count-GNN\M	8.66	1.46	**29.65**	3.34	4.41	1.82
Count-GNN	**8.54**	**1.41**	30.91	**2.46**	**4.22**	**1.76**

Table 5: Ablation study on Count-GNN.

5.3 Model Analysis

To evaluate the impact of each module in Count-GNN, we conduct an ablation study by comparing Count-GNN with its two degenerate variants: (1) Count-GNN\E, which replaces the edge-centric aggregation with the node-centric GIN; (2) Count-GNN\M, which replaces the query-conditioned modulation with a simple sum-pooling as the readout for the input graph. As shown in Table 5, the full model generally outperforms the two variants, further demonstrating the benefit of edge-centric aggregation and query-conditioned modulation. Furthermore, Count-GNN\M is usually better than Count-GNN\E, which implies that edge-centric aggregation may contribute more to the performance boost, possibly due to its more central role in capturing fine-grained structure information by treating edges as the first-class citizen.

We present additional results on scalability, parameter sensitivity and the impact of training size in Appendix F.

6 Conclusions

In this paper, we proposed a novel model called Count-GNN to approximately solve subgraph isomorphic counting on labeled graphs. In terms of modelling, we designed two key modules for Count-GNN, namely, edge-centric message passing and query-conditioned graph modulation, to improve structure matching between the query and input graphs. In terms of theory, we showed that edge-centric message passing is more expressive than its node-centric counterpart. In terms of empirical results, we conducted extensive experiments on several benchmark datasets to demonstrate the effectiveness and efficiency of Count-GNN.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB2103803. Dr. Yuan Fang acknowledges the Lee Kong Chian Fellowship awarded by Singapore Management University for the support of this work. The authors wish to thank Dr. Yuchen Li from Singapore Management University for his valuable comments on this work.

References

Ahmed, N. K.; Neville, J.; Rossi, R. A.; and Duffield, N. 2015. Efficient graphlet counting for large networks. In *IEEE International Conference on Data Mining*, 1–10.

Alon, N.; Yuster, R.; and Zwick, U. 1995. Color-coding. *Journal of the ACM*, 42(4): 844–856.

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

Carletti, V.; Foggia, P.; Saggese, A.; and Vento, M. 2017. 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.

Chen, Z.; Chen, L.; Villar, S.; and Bruna, J. 2020. Can Graph Neural Networks Count Substructures? In *Advances in Neural Information Processing Systems*, 10383–10395.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS Workshop on Deep Learning*.

Cordella, L. P.; Foggia, P.; Sansone, C.; and Vento, M. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10): 1367–1372.

Fang, Y.; Lin, W.; Zheng, V. W.; Wu, M.; Chang, K. C.-C.; and Li, X.-L. 2016. Semantic proximity search on graphs with metagraph-based learning. In *IEEE International Conference on Data Engineering*, 277–288.

Gong, L.; and Cheng, Q. 2019. Exploiting edge features for graph neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9211–9219.

Ha, D.; Dai, A.; and Le, Q. V. 2017. Hypernetworks. In *International Conference on Learning Representations*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Han, W.-S.; Lee, J.; and Lee, J.-H. 2013. Turbo$_{\mathrm{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *ACM SIGMOD International Conference on Management of Data*, 337–348.

He, H.; and Singh, A. K. 2008. Graphs-at-a-time: query language and access methods for graph databases. In *ACM SIGMOD International Conference on Management of Data*, 405–418.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hočevar, T.; and Demšar, J. 2014. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4): 559–565.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*.

Isufi, E.; Gama, F.; and Ribeiro, A. 2021. EdgeNets: Edge varying graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Jain, S.; et al. 2017. Impact of memory space optimization technique on fast network motif search algorithm. In *International Conference on Computer and Computational Sciences*, 559–567.

Jamshidi, K.; Mahadasa, R.; and Vora, K. 2020. Peregrine: a pattern-aware graph mining system. In *European Conference on Computer Systems*, 1–16.

Jiang, X.; Zhu, R.; Li, S.; and Ji, P. 2020. Co-embedding of nodes and edges with graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In *International Conference on Machine Learning*, 3734–3743.

Liu, X.; Pan, H.; He, M.; Song, Y.; Jiang, X.; and Shang, L. 2020. Neural subgraph isomorphism counting. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1959–1969.

Liu, X.; and Song, Y. 2022. Graph convolutional networks with dual message passing for subgraph isomorphism counting and matching. In *AAAI Conference on Artificial Intelligence*, 7594–7602.

Liu, Z.; Fang, Y.; Liu, C.; and Hoi, S. C. 2021. Node-wise Localization of Graph Neural Networks. In *International Joint Conference on Artificial Intelligence*, 1520–1526.

Liu, Z.; Nguyen, T.-K.; and Fang, Y. 2021. Tail-GNN: Tail-node graph neural networks. In *ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1109–1119.

Liu, Z.; Zheng, V. W.; Zhao, Z.; Yang, H.; Chang, K. C.-C.; Wu, M.; and Ying, J. 2018. Subgraph-augmented path embedding for semantic user search on heterogeneous social network. In *International Conference on World Wide Web*, 1613–1622.

Marcus, D.; and Shavitt, Y. 2012. RAGE–a rapid graphlet enumerator for large networks. *Computer Networks*, 56(2): 810–819.

Meira, L. A.; Máximo, V. R.; Fazenda, Á. L.; and Da Conceiçao, A. F. 2014. acc-Motif: Accelerated network motif detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(5): 853–862.

Milo, R.; Shen-Orr, S.; Itzkovitz, S.; Kashtan, N.; Chklovskii, D.; and Alon, U. 2002. Network motifs: simple building blocks of complex networks. *Science*, 298(5594): 824–827.

Monti, F.; Otness, K.; and Bronstein, M. M. 2018. MotifNet: a motif-based graph convolutional network for directed graphs. In *IEEE Data Science Workshop*, 225–228.

Monti, F.; Shchur, O.; Bojchevski, A.; Litany, O.; Günnemann, S.; and Bronstein, M. M. 2018. Dual-primal graph convolutional networks. *arXiv preprint arXiv:1806.00770*.

Perez, E.; Strub, F.; De Vries, H.; Dumoulin, V.; and Courville, A. 2018. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 3942–3951.

Pinar, A.; Seshadhri, C.; and Vishal, V. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *International Conference on World Wide Web*, 1431–1440.

Sankar, A.; Zhang, X.; and Chang, K. C.-C. 2019. Meta-GNN: metagraph neural network for semi-supervised learning in attributed heterogeneous information networks. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 137–144.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, 593–607.

Sun, Y.; Han, J.; Yan, X.; Yu, P. S.; and Wu, T. 2011. PathSim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11): 992–1003.

Teixeira, C. H.; Cotta, L.; Ribeiro, B.; and Meira, W. 2018. Graph pattern mining and learning through user-defined relations. In *IEEE International Conference on Data Mining*, 1266–1271.

Teixeira, C. H.; Kakodkar, M.; Dias, V.; Meira Jr, W.; and Ribeiro, B. 2020. Sequential Stratified Regeneration: MCMC for Large State Spaces with an Application to Subgraph Count Estimation. *arXiv preprint arXiv:2012.03879*.

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

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Wang, J.; Huang, Y.; Wu, F.-X.; and Pan, Y. 2012. Symmetry compression method for discovering network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(6): 1776–1789.

Wang, P.; Lui, J. C.; Ribeiro, B.; Towsley, D.; Zhao, J.; and Guan, X. 2014. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data*, 9(2): 1–27.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous graph attention network. In *The ACM Web Conference*, 2022–2032.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Xia, W.; Li, Y.; and Li, S. 2022. On the substructure countability of graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yan, X.; Yu, P. S.; and Han, J. 2004. Graph indexing: A frequent structure-based approach. In *ACM SIGMOD International Conference on Management of Data*, 335–346.

Yang, Y.; and Li, D. 2020. Nenn: Incorporate node and edge features in graph neural networks. In *Asian Conference on Machine Learning*, 593–608.

Yao, H.; Zhang, C.; Wei, Y.; Jiang, M.; Wang, S.; Huang, J.; Chawla, N.; and Li, Z. 2020. Graph few-shot learning via knowledge transfer. In *AAAI Conference on Artificial Intelligence*, 6656–6663.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, 4800–4810.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, 4438–4445.

Zhao, K.; Yu, J. X.; Zhang, H.; Li, Q.; and Rong, Y. 2021. A Learned Sketch for Subgraph Counting. In *ACM International Conference on Management of Data*, 2142–2155.