Neural Subgraph Counting with Wasserstein Estimator

Hanchen Wang[†§], Rong Hu[†], Ying Zhang[†] (✉), Lu Qin[†], Wei Wang[‡], Wenjie Zhang[§]
[†]University of Technology Sydney, [§]University of New South Wales
[‡]The Hong Kong University of Science and Technology (Guangzhou)
{hanchen.wang,ying.zhang,lu.qin}@uts.edu.au,rong.hu@student.uts.edu.au,weiwcs@ust.hk,wenjie.zhang@unsw.edu.au

ABSTRACT

Subgraph counting is a fundamental graph analysis task which has been widely used in many applications. As the problem of subgraph counting is NP-complete and hence intractable, approximate solutions have been widely studied, which fail to work with large and complex query and data graphs. Alternatively, Machine Learning techniques have been recently applied for this problem, yet the existing ML approaches either only support very small data graphs or cannot make full use of the data graph information, which inherently limits their scalability, estimation accuracies and robustness.

In this paper, we propose a novel approximate subgraph counting algorithm, *NeurSC*, that can exploit and combine information from both the query graphs and the data graphs effectively and efficiently. It consists of two components: (1) an extraction module that adaptively generates simple yet representative substructures from data graph for each query graph and (2) an estimator *WEst* that first computes the representations from individual and joint distributions of query and data graphs and then estimates subgraph counts with the learned representations. Furthermore, we design a novel Wasserstein discriminator in *WEst* to minimize the Wasserstein distance between query and data graphs by updating the parameters in network with the vertex correspondence relationship between query and data graphs. By doing this, *WEst* can better capture the correlation between query and data graphs which is essential to the quality of the estimation. We conduct experiments on 7 large real-life labeled graphs to demonstrate the superior performance of *NeurSC* in terms of estimation accuracy and robustness.

CCS CONCEPTS

• **Information systems** → *Information systems applications*; **Database query processing**.

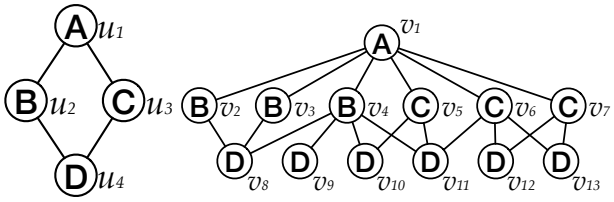
KEYWORDS

Subgraph Counting, Graph Neural Network

ACM Reference Format:
Hanchen Wang[†§], Rong Hu[†], Ying Zhang[†] (✉), Lu Qin[†], Wei Wang[‡], Wenjie Zhang[§]. 2022. Neural Subgraph Counting with Wasserstein Estimator. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3514221.3526163

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526163



(a) Query Graph q **(b) Data Graph G**

Figure 1: Example query graph and data graph.

1 INTRODUCTION

In recent years, graph structured data has been widely used. Graph analysis has also become increasingly ubiquitous and popular in data analytics. One important methodology is to investigate the graphs from a subgraph perspective [56, 64, 121]. At the heart of the subgraph analysis approaches lies the subgraph counting problem, that is, to compute the numbers of subgraphs in a data graph G that match the given query graph q. In this paper, we focus on the counting of subgraph isomorphism matches. For example, in Figure 1, $\{(u_1, v_1), (u_2, v_4), (u_3, v_5), (u_4, v_{10})\}$ is a subgraph isomorphism match of query graph q in data graph G, and there are three subgraph matches of q in G.

Subgraph counting is paramount to the query optimizer in estimating the execution cost of a query plan for the subgraph matching queries [48, 65]. These query optimizers are employed in the graph database systems like Neo4j, Oracle PGX and Amazon Neptune. Subgraph counting is also relevant and widely applied for tasks in bioinformatics [9, 38, 52, 76, 80, 90, 99, 101], computer vision [31, 114, 115], social network analysis [17, 77, 82, 100], designing graph kernels [85, 90] and determining frequent subgraph patterns [75]. Several approaches [34, 35, 35, 58, 59, 63, 64, 72] aim to obtain the exact subgraph counts. However, it has been proven that determining the subgraph existence in a graph (*i.e.,* subgraph isomorphism matching) is NP-complete [11, 23]. Computing the exact subgraph counts is even harder, especially with large query and data graphs. As a result, these exact solutions can only support the small query and data graphs, which precludes their applicability in many scenarios. For example, exact subgraph counting method cannot solve the graph frequency mining problem on the million-scale social networks within a week [5, 17]. Given the importance, popularity and hard tractability of subgraph counting, approximate subgraph counting approaches have been proposed in recent years [9, 12, 13, 17, 18, 52, 80, 90, 101, 117]. These methods enable the subgraph counting on relatively larger graphs, and hence are widely applied in the aforementioned applications. Please refer to survey papers [73, 78] and tutorial [83] for more details.

Existing non-learning-based subgraph counting algorithms are classified into two categories: summary-based techniques [14, 69,

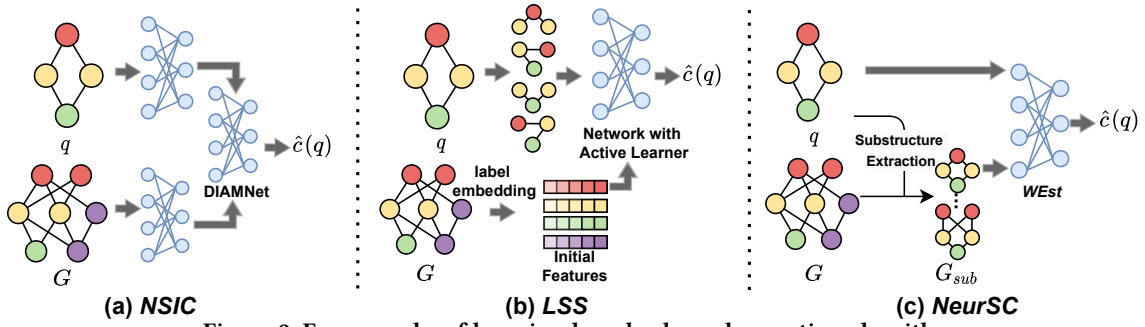


Figure 2: Frameworks of learning-based subgraph counting algorithms

86] or sampling-based techniques [17, 50, 92, 119] in [73]. Summary-based techniques decompose the query graph into small subgraphs, and aggregate the subgraph counts of these small subgraphs for estimation. These methods assume that the counts of the smaller subgraphs are independent, which is impractical for real large graphs and incurs inaccurate estimation. Sampling-based techniques sample subgraphs from data graph, count the subgraph matchings between query graph and sampled subgraphs, and aggregate the counts for estimation. These methods usually suffer from *sampling failure*, *i.e.,* failure to generate the valid sample from data graph that matches the query graph, especially on complex query and data graphs. Motivated the above limitations, machine learning techniques are employed for subgraph counting problem.

Recently, several learning-based models have been proposed for subgraph analysis tasks. The model in [120] utilizes the graph neural networks to perform the subgraph counting with small motifs as queries. [120] has scalability and efficiency issue due to the complexity of its network, despite of its expressive power. *NeuroMatch* [55] and *LMKG* [20] are proposed for the subgraph containment problem, *i.e.,* predicting whether the given query graph q is a subgraph of the data graph G. These algorithms perceive the presence or absence problem as a binary classification problem and cannot be trivially extended to the subgraph counting problem which is usually regarded as a regression problem.

As with the problem setting in this paper, the model proposed in *NSIC* [53] and *LSS* proposed in [117] are designed for subgraph counting. Their frameworks are illustrated in Figure 2 (a) and (b). *NSIC* [53] in Figure 2 (a) first encodes both query and data graphs into vectors with recurrent neural networks (RNNs) or GNNs and then predicts the subgraph counts with the proposed network DIAMNet which is based on RNN. However, because (1) the model encodes both full-sized query and data graphs, and (2) the model has a large parameter space, *NSIC* suffers from severe efficiency and scalability issues. Besides, when the data graph is large, the estimation result of *NSIC* is intrinsically determined by the data graph, and the query graphs become indistinguishable in the model, which is demonstrated in Section 6.2. Therefore, *NSIC* can only handle the subgraph counting problem on very small data graphs, *e.g.,* graphs with thousands of vertices.

LSS (in Figure 2 (b)) with an active learner (AL) is proposed in [117]. *LSS* decomposes the given query graph q into smaller substructures and encodes these substructures into vector representations by GNN. Then *LSS* estimates the subgraph counts of query graph q by aggregating the representations of all substructures with a network using self-attention mechanism. Though *LSS* achieves state-of-the-art accuracy in estimating the subgraph counts, it still has two main limitations. Firstly, *LSS* only applies the graph neural network on the substructures obtained from the query graph but cannot fully exploit the data graph information. *LSS* only utilizes the data graph information to initialize the features for query vertices. Such initialization methods in *LSS* either only exploit the label frequencies in data graph or directly use the embeddings of labels in data graph produced by task-independent network embedding methods, such as `DeepWalk` [74], `node2vec` [26], and `ProNE` [113]. This initialization method ignores the query structural information. Besides, this initialization can barely leverage the data graph information and might be easily influenced by irrelevant data vertices. Secondly, the substructures of query graphs are obtained by a k-hop breadth-first search (BFS) starting from each query vertex u, where k is a fixed constant 3 in *LSS* for all queries regardless of their sizes. Nevertheless, if the diameter of the query graph is smaller than or equal to the given k, all generated substructures will be the same as the query graph q. Such situation can be quite common given the fact that the query graphs are generally small (*i.e.,* with less than 32 vertices) and dense (*e.g.,* the average degree of query vertices is greater than 4) in many real-life applications. When all substructures are the same as q, *LSS* needs to compute the representations of the same substructures multiple times, which reduces both efficiency and accuracy of *LSS*.

The circumstance that the existing learning-based subgraph counting methods either only support exceedingly small data graphs or cannot fully exploit the data graph information motivates us to design a novel model. In this paper, we propose a Neural Subgraph Counting algorithm, namely *NeurSC* (in Figure 2 (c)). *NeurSC* has two main components: (1) the extraction approach which generates the candidate substructures, and (2) the learning-based estimator named *WEst* which produces the subgraph counts prediction with input query graph and candidate substructures. *NeurSC* first extracts the simple yet representative candidate substructures from data graph G based on the query graph q. Afterwards, query graph q and its corresponding candidate substructures are fed into the estimator for subgraph counting. Therefore, *NeurSC* can adaptively capture the representative information in both query and data graphs and avoid the influence from the unpromising data vertices (*i.e.,* data vertices cannot be matched to any query vertex). Besides, since there could exist multiple candidate substructures, *NeurSC* could make a trade-off between efficiency and accuracy by adjusting the number of substructures involved in the computation.

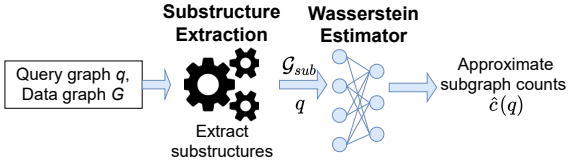


Figure 3: Framework of *NeurSC*

Table 1: The summary of notations

Notation	Definition
q, G, G_{sub}	query graph, data graph and candidate substructure.
V, E, L	vertex set, edge set and label set.
f_l, f_{iso}	label mapping function and subgraph isomorphism function.
$e(u,v), N(v)$	an edge between node u, v and the neighbor set of v.
$d(u)$	degree of node u.
f_θ, f_ω	estimation network and Wasserstein discriminator.
$CS, CS(u)$	candidate set and candidate set of query vertex u.
$c(q), \hat{c}(q)$	ground-truth and estimated subgraph counts of q.

We devised a novel GNN-based estimator, namely *WEst*, to predict the subgraph counts with the input query graph and candidate substructures. *WEst* consists of the intra-graph neural network, inter-graph neural network and the Wasserstein discriminator. With the intra and inter-GNNs, *WEst* is able to learn representations and capture the label and structural information within and between the query graph and its candidate substructures. Furthermore, the Wasserstein discriminator is proposed for adversarial training on *WEst*, which leverages the vertex correspondence relationship between query graph and candidate substructures to minimize the Wasserstein distance between the representations of these graphs. By utilizing the vertex correspondence information, the discriminator improves the representation quality and the estimation accuracy.

Contributions. Our contributions are summarized as follows:

- We propose a learning-based subgraph counting method *NeurSC* that estimates the subgraph counts with both individual and joint distributions of both query and data graphs.
- *NeurSC* extracts the substructures from data graph adaptively according to the query graph to reduce the sizes of graphs that should be processed, and hence not only avoids the influence from the unpromising data vertices and edges during the representation learning process but also improves the efficiency and scalability of the model.
- We design a Wasserstein discriminator in our model to leverage the vertex correspondence information to further improve the estimation accuracy.
- We conduct experiments on 7 real-life data graphs and demonstrate that NeurSC could significantly improve estimation accuracy and robustness. For instance, *NeurSC* could reduce up to 95% of the error compared to the state-of-the-art technique *LSS* in the experiments.

Roadmap. Section 2 gives the background and problem statement. In Section 3, we present an overview of *NeurSC*. The candidate substructure extraction and the learning-based estimator are introduced in Section 4 and Section 5 respectively. Section 6 reports the empirical results, Section 7 reviews the related works, and Section 8 concludes the paper.

Algorithm 1: Framework of *NeurSC*

Input: query graph $q = (V(q), E(q))$, data graph $G = (V(G), E(G))$.
Output: Estimated subgraph counts $\hat{c}(q)$.

```
for each u ∈ V(q) do
    CS(u) ← generate candidate vertex set for query vertex u
CS(q) = ⋃_{u∈V(q)} CS(u)
G_sub = {G_sub^(i)} ← candidate substructure(s) with vertex set CS(q)
X_q ← initial features of vertices in query graph q
for i = 1, ..., |G_sub| do
    X_sub^(i) ← initial features of vertices in substructure G_sub^(i)
    c'_i(q) = WEst(q, G_sub^i, X_q, X_sub^(i))
ĉ(q) = Σ_{i=1}^{|G_sub|} c'_i(q)
return ĉ(q)
```

2 PRELIMINARIES

2.1 Background

We follow the typical setting of the subgraph counting problem and focus on the undirected labeled graphs. In this paper, the query graph and data graph are denoted as q and G respectively. We use $V(q)$ and $V(G)$ to denote *query vertices* and *data vertices*. $E(q)$ and $E(G)$ are used to denote the sets of edges in the query graph and data graph respectively. Both query graph q and data graph G share the same label function f_l which maps a vertex $v \in V$ into a label $l \in L$, where L is the label set. Next, we give a formal definition of subgraph counting and related preliminaries.

2.2 Problem Statement

Definition 1 (Subgraph Isomorphism [89]). *Given $q = (V, E)$ and $G = (V', E')$, a subgraph isomorphism is an **injective function** f_{iso} from V to V' such that $\forall u \in V, f_l(u) = f_l(f_{iso}(u))$; and $\forall e(u,u') \in E, e(f_{iso}(u), f_{iso}(u')) \in E'$.*

With given q and G, the subgraph isomorphism counting is to estimate the number of subgraph isomorphisms from q to G without enumeration of all possible mappings.

We aim to develop a learning-based model to perform the subgraph isomorphism counting with better accuracy. The accuracy is measured by *q-error* [66], a widely used error function in various related works [32, 73, 87, 88, 103, 105, 109, 117]. Formally, *q-error* is defined as $q\text{-}error = max(\frac{max(1,c)}{max(1,\hat{c})}, \frac{max(1,\hat{c})}{max(1,c)})$, where $\hat{c}$ is the estimation of the subgraph counts and c is the ground truth counts.

The frequently used notations are summarized in the Table 1. In this paper, we might refer to subgraph isomorphism counting as subgraph counting when there is no ambiguity. Please note that the subgraph counting can also be performed based on homomorphism. Though *NeurSC* can naturally handle the subgraph homomorphism counting, we still focus on subgraph isomorphism counting in this paper because of its popularity and wide applications.

3 OVERVIEW

In this section, we introduce the framework of our proposed model *NeurSC* designed for subgraph counting problem. The framework of our proposed *NeurSC* is illustrated in Figure 3 and Algorithm 1.

NeurSC first selects the candidate data vertices according to the input query graph and generates the candidate sets (lines 1 and 2). With the candidate set, the model generates the candidate substructure(s) and initializes the features for query graph and substructure(s) as the input of the estimation network (lines 3 to 7). The estimation network is called Wasserstein Estimator, namely *WEst*, which takes the initial features of both query graph and extracted substructure as input, and outputs a real number as the approximate subgraph counts of query graph on the candidate substructure(line 8). We sum up the subgraph counts estimations of q on all substructure(s) (line 9) as the subgraph counts estimation of q on the original data graph G. We also design a Wasserstein discriminator in the training process to optimize the parameters in the graph neural networks and discriminator in a unified manner. The discriminator leverages the vertex correspondence information to improve the estimation accuracy of *WEst*. The detailed architecture of *WEst* is introduced in Section 5. The following Section 4 introduces the substructure extraction procedure of *NeurSC*.

4 SUBSTRUCTURE EXTRACTION

Motivation. The structural and attribute information within both query and data graph serves a critical role in subgraph counting. However, the existing learning-based subgraph counting solutions cannot fully use these information while having great efficiency, accuracy and scalability to support the query on large graphs. For example, LSS [117] only applies the graph neural networks on query graphs; NSIC [53] applies the graph neural networks on both query and the entire data graph, which makes it time prohibitive for queries on large scale data graphs. Besides, the learning model applied on the entire data graph will inevitably produce representations of unpromising data vertices which cannot be matched to query vertices. For example, data vertices $\{v_2, v_3, v_7, v_8, v_9, v_{12}, v_{13}\}$ in Figure 1b cannot be matched to any query vertex in Figure 1a, however their representations are still computed used for subgraph counts estimation in [53], which causes deterioration of estimation accuracy and efficiency of [53]. Motivated by these limitations, a substructure extraction module is utilized to enable our model to capture the structure and label information in the data graph and reduce the number of data vertices involved in the computation.

(1) Candidate Filtering. *NeurSC* first reduces the size of data graph by generating the *complete candidate set* which is defined in Definition 2.

Definition 2 (Complete Candidate Vertex Set *CS*). *Given q and G, a complete candidate vertex set $CS(u)$ of $u \in V(q)$ is a set of data vertices such that for each $v \in V(G)$, if (u, v) exists in a match from q to G, then $v \in CS(u)$.*

Once the *complete candidate set* is obtained, *NeurSC* can exploit the label and topology information of the data graph while averting the effect of the unpromising data vertices.

There are many existing candidate set generation methods in subgraph matching literature. Our proposed model adopts the candidate set generation method from GraphQL [33] which is proven to have the best pruning power, *i.e.,* generates the candidate set with the smallest number of vertices $\left|\bigcup_{u \in V(q)} CS(u)\right|$, in the experimental survey work [89]. Particularly, *NeurSC* generates the candidate set with two stages: local pruning and global refinement. Local pruning filters out the invalid nodes based on the *profile* of the neighborhood graph of the query vertex u, which is the lexicographically ordered labels of u and neighbors of u within distance r (hops of neighbors). If the profile of query vertex u is a sub-sequence of that of data vertex v, then v is added to $CS(u)$. After the candidate set $CS(u)$ has been generated for all vertices $u \in V(q)$, global refinement prunes $CS(u)$ as follows: given $v \in CS(u)$, the algorithm first builds a bipartite graph B_u^v between $N(u)$ and $N(v)$ by adding $e(u', v')$ if $v' \in CS(u')$, where $u' \in N(u)$ and $v' \in N(v)$. Global refinement then checks whether there is a semi-perfect matching in B_v^u, *i.e.,* whether all vertices in $N(u)$ have at least one edge in B_v^u. If not, v will be removed from $CS(u)$. These procedures could be conducted multiple times to obtain a more compact candidate set.

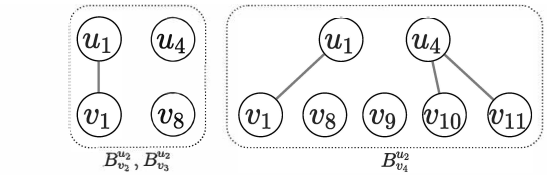


Figure 4: Illustration of the construction of bipartite graphs for candidate filtering.

Example 1. *Consider the query graph and data graph in Figure 1. The profile of the query vertices u_2 is $\{A, B, D\}$. The profiles of v_2 and v_3 are both $\{A, B, D\}$ and that of v_4 is $\{A, B, D, D, D, D\}$. They all subsume the profile of u_2, hence we have $CS(u_2) = \{v_2, v_3, v_4\}$. Similarly, local pruning can build the following candidate sets: $CS(u_1) = \{v_1\}$, $CS(u_3) = \{v_5, v_6, v_7\}$ and $CS(u_4) = \{v_{10}, v_{11}\}$. Global refinement is then applied to build a bipartite graphs B^{u2}_{v2}, B^{u2}_{v3} and B^{u2}_{v4} (See Figure 4). Since $v_8 \notin CS(u_4)$, we cannot find a semi-perfect matching in B^{u2}_{v2} and B^{u2}_{v3}. Hence v_2 and v_3 should be removed from $CS(u_2)$. Finally, candidate set for u_2 is obtained as $CS(u_2) = \{v_4\}$. In the same way, we eventually obtain the following candidate sets for all query vertices: $CS(u_1) = \{v_1\}$, $CS(u_3) = \{v_5, v_6\}$, $CS(u_4) = \{v_{10}, v_{11}\}$.*

NeurSC can terminate the estimation early once there exists an empty candidate set for any query vertex u or the size of candidate set $|\bigcup_{u \in V(q)} CS(u)|$ is smaller than the size of query graph $|V(q)|$.

(2) Substructure Extraction. Suppose we get the candidate vertex set $CS(q) = \bigcup_{u \in V(q)} CS(u)$ of query graph q. We need to extract a subgraph from the data graph G, based on the candidate set $CS(q)$. To avoid possible ambiguity, we refer to the extracted subgraph as the *candidate substructure* of the data graph, denoted by G_{sub}. In *NeurSC*, G_{sub} is the *induced subgraph* of G with vertex set $CS(q)$.

Definition 3 (Induced Subgraph). *Given a graph $G = (V, E)$, a subset $S \subset V$ of vertex set of G. The induced subgraph G_{sub} is the graph whose vertex set is S and whose edge set consists of all the edges in E that have both endpoints in S. The induced subgraph G_{sub} shares the same label mapping function f_l as G.*

With the candidate sets obtained in Example 1, the induced subgraph G_{sub} can be constructed with vertex set $\{v_1, v_4, v_5, v_6, v_{10}, v_{11}\}$, the labels of the vertices and the edges between these vertices in data graph in Figure 1b. The constructed G_{sub} is shown in Figure 6.

Please note that the obtained candidate substructure is required to be a *connected* graph. Therefore, if G_{sub} is not a connected graph, it will be divided into several connected graphs. Without loss of

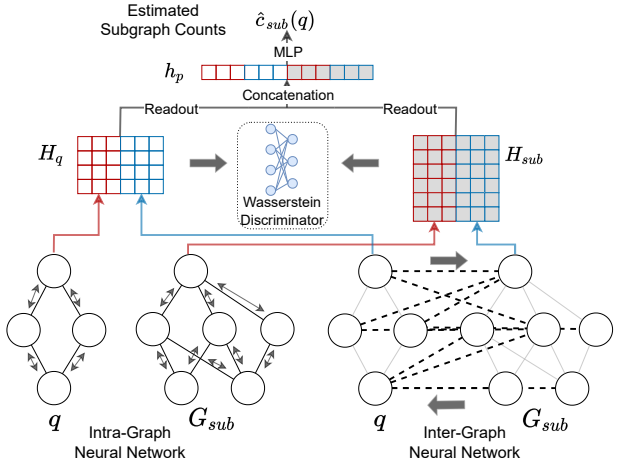


Figure 5: Illustration of *WEst* which computes the vertex representations with intra and inter-graph neural networks, estimates the subgraph counts and updates the parameters with Wasserstein discriminator.

generality, the set of candidate substructure(s) is denoted as $\mathcal{G}_{sub}$ in which each element $G_{sub}^{(i)}$ is a connected graph, and the number of candidate substructures is $|\mathcal{G}_{sub}|$. If the number of vertices or edges in $G_{sub}^{(i)}$ is smaller than that of the query graph, *NeurSC* would skip the following computation on this candidate substructure since a query graph cannot be isomorphic to a smaller graph.

5 WASSERSTEIN ESTIMATOR

With the input query graph q and the obtained substructure G_{sub}, the Wasserstein Estimator, namely *WEst*, is proposed to estimate the subgraph counts. The architecture of *WEst* is demonstrated in Figure 5. *WEst* consists of i) an intra-graph neural network which captures the structural information within q and G_{sub}, ii) an inter-graph attentive network which aggregates features between query vertex u and its candidate vertices $v' \in CS(u)$ and iii) a Wasserstein discriminator which boosts the estimation performance of *WEst* by leveraging the vertex correspondence between q and G_{sub}. The high-level forward propagation procedure is listed in Algorithm 2. For simplicity, we omit the index i of candidate substructure $G_{sub}^{(i)}$ in Algorithm 2 and the following subsections when the context is clear. In the rest of the paper, we refer to the network in Algorithm 2 that outputs the subgraph counts estimation as the *estimation network* with parameters θ, denoted by f_θ. In the following subsections, we introduce each component of *WEst* in detail.

5.1 Feature Initialization

We convert the input query graph and candidate substructure pair (q, G_{sub}) to the feature vectors for vertices by exploiting the property values in these graphs. We choose the label and degree which are the most widely used graph properties [28, 29, 40, 84] to build the feature vectors. Specifically, since each label is assigned with a unique integer, *e.g.*, the label of u_1 is 1 ($f_l(u_1) = 1$), and degree is an integer, we use a function to transfer the integers to multi-hot vectors. Considering that the number of labels and the maximum degree in the graph are relatively small, we simply use the binary

Algorithm 2: Forward Propagation of *WEst*

Input: query graph q, candidate substructure G_{sub}, initial features $\boldsymbol{X}_q$ and $\boldsymbol{X}_{sub}$, readout function $Readout(\cdot)$.
Output: Estimated subgraph counts $\hat{c}_{sub}(q)$.

$\boldsymbol{H}_q^{(0)} = \boldsymbol{X}_q$, $\boldsymbol{H}_{sub}^{(0)} = \boldsymbol{X}_{sub}$
// Intra-graph neural network.
for $k = 1, ..., K$ **do**
  for $u \in V(q)$ **do**
    $\boldsymbol{h}_u^{(k)} = MLP^{(k)}((1+\epsilon^{(k)})\boldsymbol{h}_u^{(k-1)}, \sum_{u' \in N_q(u)} \boldsymbol{h}_{u'}^{(k)})$
  for $v \in V(G_{sub})$ **do**
    $\boldsymbol{h}_v^{(k)} = MLP^{(k)}((1+\epsilon^{(k)})\boldsymbol{h}_v^{(k-1)}, \sum_{v' \in N_{sub}(v)} \boldsymbol{h}_{v'}^{(k)})$
$\boldsymbol{H}_q^{intra} = \{\boldsymbol{h}_u^{(K)}; \forall u \in V(q)\}$, $\boldsymbol{H}_{G_{sub}}^{intra} = \{\boldsymbol{h}_v^{(K)}; \forall v \in V(G_{sub})\}$
// Inter-graph neural network.
$G_B \leftarrow$ generate a bipartite graph on vertex set $V(q) \cup V(G_{sub})$
for $k' = 1, ..., K'$ **do**
  for $u \in V(G_B)$ **do**
    $\boldsymbol{h}_u^{(k')} = \alpha_{uu}^{(k')} \boldsymbol{\Theta}^{(k')} \boldsymbol{h}_u^{(k'-1)} + \sum_{v \in N_{G_B}(u)} \alpha_{uv}^{(k')} \boldsymbol{\Theta}^{(k')} \boldsymbol{h}_v^{(k')}$
$\boldsymbol{H}_q^{inter} = \{\boldsymbol{h}_u^{(K')}; \forall u \in V(q)\}$, $\boldsymbol{H}_{G_{sub}}^{inter} = \{\boldsymbol{h}_v^{(K')}; \forall v \in V(G_{sub})\}$
$\boldsymbol{H}_q = \{\boldsymbol{h}_u^{intra} \| \boldsymbol{h}_u^{inter}; \forall u \in V(q)\}$
$\boldsymbol{H}_{G_{sub}} = \{\boldsymbol{h}_v^{intra} \| \boldsymbol{h}_v^{inter}; \forall v \in V(G_{sub})\}$
$\boldsymbol{h}_q = Readout(\boldsymbol{H}_q)$, $\boldsymbol{h}_{G_{sub}} = Readout(\boldsymbol{H}_{G_{sub}})$
// Predict the subgraph counts on G_{sub} with MLP.
$\hat{c}_{sub}(q) = MLP(\boldsymbol{h}_q \| \boldsymbol{h}_{G_{sub}})$
return $\hat{c}_{sub}(q)$

encoding function to convert the decimal digits into a binary number and directly use this as a multi-hot vector. We add leading 0*s* to ensure all the vectors have the same length. In subgraph matching, the neighbor information is also important. Based on this intuition, the neighborhood label and degree information are also converted to vectors by the binary encoding function. The obtained vectors are compressed into one by mean pooling which outputs the average of input vectors. Therefore, the input feature vector for vertex v in either query graph or candidate substructure can be obtained with the equation:

$$\boldsymbol{x}_v = f_b(deg_v) \parallel f_b(f_l(v)) \parallel_{i=1}^{k} MP_{\forall v' \in N^{(i)}(v)}(f_b(deg_{v'}) \parallel f_b(f_l(v'))), \tag{1}$$

where $\parallel$ indicates the concatenation operation, f_b is the binary encoding function, MP is mean pooling and $N^{(i)}(v)$ is i-hop neighbors of v. Eventually, the initial features $\boldsymbol{X}_q = \{\boldsymbol{x}_u \in \{0,1\}^{dim_0}; u \in V(q)\}$ and $\boldsymbol{X}_{sub} = \{\boldsymbol{x}_v \in \{0,1\}^{dim_0}; v \in V(G_{sub})\}$ for query graph and substructures are generated as the input of *WEst*, where dim_0 is the dimension of the initial features.

5.2 Intra-Graph Neural Network

Motivation. The topological structure and attribute information within query graph and candidate substructures is important for estimation of the subgraph counting. To capture such information, graph neural network (GNN) is utilized to generate the intra-graph representations for vertices in the query graph and candidate substructures. Through our investigation, graph neural networks like

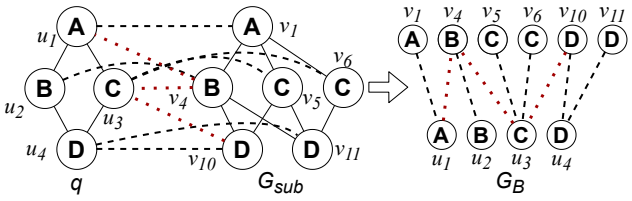


Figure 6: Illustrating the construction of a bipartite graph for inter-graph neural network.

GCN [43], GAT [91] and GraphSAGE [27] enjoy popularity in various applications, but they have limitations in structure preserving ability. Recently, many graph neural networks [61, 62, 67, 93, 108] have been proposed with equally expressive power as Weisfeiler-Lehman (WL) test [25, 49] . These networks are also widely used in applications in which the graph structural information is crucial, such as graph classification [61, 95, 108] , motif count estimation [120] and subgraph isomorphism counting [53, 117]. Among these networks, GNN in [61] could achieve better expressive power than graph isomorphism network (GIN) [108]. However, our preliminary experiment suggests that GNN in [61] is more time consuming with a limited improvement of estimation accuracy compared to GIN in our framework. As a result, GIN is exploited as the intra-graph neural network.

GIN adopts the conventional framework of graph neural networks with an *aggregate-and-combine* scheme as follow:

$$\boldsymbol{h}_u^{(k)} = \mathcal{COM}^{(k)}(\boldsymbol{h}_u^{(k-1)}, \mathcal{AGG}^{(k)}\{\boldsymbol{h}_{u'}^{(k)}; u' \in N(u)\}), \tag{2}$$

where $\mathcal{AGG}$ is the aggregation that iteratively updates the representation of a vertex by aggregating the representations of its neighbors, and $\mathcal{COM}$ is the combine operation that updates the representation of vertex u by the aggregated representations and its own representation $\boldsymbol{h}_u^{(k-1)}$ from the previous layer. $\boldsymbol{h}_u^{(0)}$ is the initial feature input $\boldsymbol{x}_u$ of vertex u. To achieve expressive power as WL-test, $\mathcal{AGG}$ and $\mathcal{COM}$ should be injective (Theorem 3 in [108]). In GIN, $\mathcal{AGG}$ and $\mathcal{COM}$ is modeled as multilayer perceptrons (MLP) [22]. Therefore, $\boldsymbol{h}_u^{(k)} \in \mathbb{R}^{dim_k}$ can be computed by GIN formulated as follows:

$$\boldsymbol{h}_u^{(k)} = \sigma(MLP^{(k)}((1+\epsilon^{(k)})\boldsymbol{h}_u^{(k-1)}, \sum_{u' \in N(u)} \boldsymbol{h}_{u'}^{(k)})), \tag{3}$$

where ϵ is a learnable parameter or a fixed scalar to adjust the importance of the representations from the previous layer and obtained by neighbor aggregation, σ is the nonlinear activation function such as ReLU [68]. By applying GIN, we can obtain the intra-graph representations $\boldsymbol{h}_u^{intra}$ and $\boldsymbol{h}_v^{intra}$ with dimension dim_K which are the outputs of the K-th layer.

5.3 Inter-Graph Neural Network

Motivation. The intra-graph representation $\boldsymbol{h}^{intra}$ could capture the structural information *within* the query graph and candidate substructures. The interrelationship *between* query graph and candidate substructures is also paramount to find exact subgraph counts [15, 28, 29, 40, 84]. Therefore, the graph neural network is designed to capture the inter-graph information to further improve the estimation accuracy.

One immediate solution is to perform neighbor aggregation between each pair of vertices between $V(q)$ and $V(G_{sub})$. Under scrutiny, this solution clearly incurs unnecessary time cost and might result in an unsatisfactory performance because of the aggregation between query and data vertices that are dissimilar with each other. Therefore, we first build a bipartite graph G_B with the vertex set $V(G_B) = V(q) \cup V(G_{sub})$ and then apply an attentive network on G_B. If a vertex $v \in V(G_{sub})$ is a candidate vertex of query vertex $u \in V(q)$, there will be an edge between u and v in G_B, *i.e.,* $E(G_B) = \{(u,v); u \in V(q) \wedge v \in V(G_{sub}) \wedge v \in CS(u))\}$. Nevertheless, the built bipartite graph may not be a connected graph. In that case, we would randomly add edges between $V(q)$ and $V(G_{sub})$ to link the connected components of G_B (See Figure 6).

It is intuitive that each candidate data vertex has different importance to query vertex in terms of the subgraph isomorphism counting. For example, in Figure 6, v_5 can be more important to u_3 than v_6, since v_5 is connected to more candidate vertices and involved in two subgraph isomorphisms. Therefore, the attentive graph neural network is applied on the bipartite graph G_B with the learnable attention coefficient on each edge $e \in E(G_B)$ which approximates the importance of each neighbor of query vertex. Suppose there are K' layers in the attentive graph neural network, the k-th layer is formulated in the following Equation 4.

$$\boldsymbol{h}_u^{(k)} = \sigma(\alpha_{uu}^{(k)} \boldsymbol{\Theta}^{(k)} \boldsymbol{h}_u^{(k-1)} + \sum_{v \in N_{G_B}(u)} \alpha_{uv}^{(k)} \boldsymbol{\Theta}^{(k)} \boldsymbol{h}_v^{(k)}), \tag{4}$$

where $\boldsymbol{\Theta}^{(k)}$ is the weight matrix parameter in k-th layer, σ denotes the activation function, and $\alpha_{uv'}$ is the attention coefficient between vertices u and v' which is computed as:

$$\alpha_{uv} = \frac{exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{\Theta}_a \boldsymbol{h}_u] \parallel [\boldsymbol{\Theta}_a \boldsymbol{h}_v]))}{\Sigma_{v' \in N_{G_B}(u)} exp(\text{LeakeyRelu}(\boldsymbol{a}[\boldsymbol{\Theta}_a \boldsymbol{h}_u] \parallel [\boldsymbol{\Theta}_a \boldsymbol{h}_{v'}]))}, \tag{5}$$

where $\boldsymbol{a}$ is a learnable attention weight vector, and $\parallel$ is the concatenation operation. The output vectors of the last layer of the attentive network are used as the inter-graph representations, *i.e.,* $\boldsymbol{h}_u^{inter} = \boldsymbol{h}_u^{(K')} \in \mathbb{R}^{dim_{K'}}$. Please note that different from the original GAT [91], our attentive network does not include the self loop but focuses on the message passing between the neighbors in different vertex sets. This distinction would help our inter-graph neural network focus more on the interrelationship between the graphs when learning the representations.

5.4 Readout and Prediction

Once obtained the intra-graph and inter graph representations $\boldsymbol{h}_u^{intra}$ and $\boldsymbol{h}_u^{inter}$ by applying the corresponding graph neural networks in Equation 3 and 4, *WEst* produces the final representations by concatenating these two vectors, *i.e.,* $\boldsymbol{h}_u = \boldsymbol{h}_u^{intra} \| \boldsymbol{h}_u^{inter}$ with dimension $dim_K + dim_{K'}$, for each vertex $u \in V(q)$ and $v \in V(G_{sub})$. To compute the representation vectors for query graph and candidate substructure, *WEst* aggregates the corresponding vertex representations with a Readout function:

$$\boldsymbol{h}_q = Readout(\boldsymbol{H}_q), \quad \boldsymbol{h}_{G_{sub}} = Readout(\boldsymbol{H}_{G_{sub}}), \tag{6}$$

where $\boldsymbol{H}_q$ (*resp.* $\boldsymbol{H}_{G_{sub}}$) is the matrix of representations of all vertices in query graph q (*resp.* substructure G_{sub}) and $Readout(\cdot)$ is a permutation invariant and injective function: sum pooling.

In the prediction procedure, the final representation for prediction is generated by concatenating $\boldsymbol{h}_q$ and $\boldsymbol{h}_{G_{sub}}$, *i.e.,* $\boldsymbol{h}_p = \boldsymbol{h}_q \| \boldsymbol{h}_{G_{sub}}$. Then $\boldsymbol{h}_p$ with dimension $2*(dim_K + dim_{K'})$ is fed into an MLP which outputs a one-dimension scalar $\hat{c}_i(q)$ as the estimated subgraph counts of q on $G_{sub}^{(i)}$. *WEst* aggregates the subgraph count estimations by summation of the estimations on all substructures in $\mathcal{G}_{sub}$, *i.e.,* $\hat{c}(q) = \sum_{i=1}^{|\mathcal{G}_{sub}|} \hat{c}_i(q)$, to predict the subgraph counts of query graph q on data graph G.

5.5 Wasserstein Discriminator

Motivation. Learned by the intra and inter-graph neural networks, the query vertex and its matched/candidate data vertices are not guaranteed to be close with each other in the representation space, which is counter-intuitive and might reduce the estimation accuracy. Therefore, a discriminator is designed to minimize the distance between the representations of corresponding query and data vertices in terms of a specific distance metric. The Wasserstein distance [94], which is also known as Earth Mover's distance that is widely used in the database literature [36, 39, 54, 71], has better property compared to other popular metrics such as Kullback-Leibler (KL) divergence [44] and Jensen-Shannon (JS) divergence [57]. In some cases, two probability distributions converge under Wasserstein distance but do not converge under KL and JS divergences, which can cause instability in model training. Please refer to [7, 8] for examples and analysis. As a result, the Wasserstein discriminator is proposed for adversarial training on *WEst*, which minimizes the Wasserstein distance between learned representations of q and G_{sub}.

Wasserstein distance [94] is defined as follows:

Definition 4 (Wasserstein Distance.). *Given random variables μ and ν that are subject to probability distributions $\mathbb{P}_q$ and $\mathbb{P}_g$ respectively, the Wasserstein-1 distance W_1 between distributions $\mathbb{P}_q$ and $\mathbb{P}_g$ is defined as:*

$$W_1(\mathbb{P}_q, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_q, \mathbb{P}_g)} \mathbb{E}_{(\mu,\nu)\sim\gamma}[\|\mu - \nu\|], \tag{7}$$

where $\Pi(\mathbb{P}_q, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(\mu, \nu)$ whose marginals are respectively $\mathbb{P}_q$ and $\mathbb{P}_g$.

Here, γ indicates how much "mass" must be transported from μ to ν to transform the distributions $\mathbb{P}_q$ into the distribution $\mathbb{P}_g$. Then the Wasserstein-1 (W-1) distance is the "cost" of the optimal transport plan. Given two graphs g_s and g_t, the element in optimal transport plan $\gamma_{i,j}$ is the probability the $v_i \in g_s$ matches $v_j \in g_t$. The optimal transport plan γ, which yields the vertex correspondence across the graphs, is used in many learning-based graph alignment works [60, 106, 107]. However, in our problem, it is not necessary to compute the exact optimal transport due to its extra time cost and limited improvement to the estimation accuracy. Therefore, in *WEst*, we design a network aiming to minimize the W-1 distance between q and G_{sub} as the discriminator.

It is obvious that the infimum in Equation 7 is highly intractable. With the Kantorovich-Rubinstein duality [24, 94], the Wasserstein-1 distances can be formulated as:

$$W_1(\mathbb{P}_q, \mathbb{P}_g) = \sup_{\|f_\omega\|_L \le 1} \mathbb{E}_{\mu\sim\mathbb{P}_q}[f_\omega(\mu)] - \mathbb{E}_{\nu\sim\mathbb{P}_g}[f_\omega(\nu)], \tag{8}$$

where the supremum is over all the 1-Lipschitz functions $f_\omega : M \to \mathbb{R}$ which map the μ, ν in variable space M to the real space $\mathbb{R}$. Our discriminator takes the representations of vertices in the correspondence set as the input and models the function f_ω as a neural network parameterized by ω. As in Wasserstein GAN [8], f_ω is enforced to be 1-Lipschitz by clamping the weights to a fixed box, *e.g.,* $\omega \in [-0.01, 0.01]^{d_\omega}$. Therefore, the Wasserstein distance between query graph q and substructure G_{sub} is minimized when f_ω is optimized to maximize $\mathcal{L}_w(q, G_{sub})$ (minimize $-\mathcal{L}_w(q, G_{sub})$) in the following equation proposed in [21]:

$$\mathcal{L}_w(q, G_{sub}) = \sum_{u \in V'(q)} f_\omega(\boldsymbol{h}_u) - \sum_{v \in V'(G_{sub})} f_\omega(\boldsymbol{h}_v), \tag{9}$$

where $V'(q)$ and $V'(G_{sub})$ are the correspondence vertex sets from q and G_{sub} in optimal transport plan when the W-1 distance is minimized. However, we still need to find the correspondence vertex sets $V'(q)$ and $V'(G_{sub})$. Computing the exact correspondence relationship is obviously inefficient and expendable due to the quadratic time complexity. A method to generate an approximation of the correspondence pairs is proposed in [21]. $f_\omega(\boldsymbol{h}_u)$ and $f_\omega(\boldsymbol{h}_v)$ are calculated for each vertex in two graphs. Then the vertices in $V(q)$ that minimize $f_\omega(\boldsymbol{h}_u)$ and the vertices in $V(G_{sub})$ that maximize $f_\omega(\boldsymbol{h}_v)$ will be the selected vertex pairs. Though this approximation method is efficient with linear time complexity, it totally ignores the correspondence relationship in making its selection.

To address this issue, in *WEst*, we select the vertex pairs with the help of the candidate set. Particularly, *WEst* first calculates $f_\omega(\boldsymbol{h}_u)$ and $f_\omega(\boldsymbol{h}_v)$ for each vertex in two graphs. Then starting from the query vertex u' that minimizes $f_\omega(\boldsymbol{h}_{u'})$, the data vertex v' that belongs to the candidate set of u', *i.e.,* $v' \in CS(u')$, and also maximizes $f_\omega(\boldsymbol{h}_{v'})$ is selected. The selection iterates over all query vertices u in ascending order of $f_\omega(\boldsymbol{h}_u)$ to generate the correspondence vertex sets $V'(q)$ and $V'(G_{sub})$. In selecting the query vertex u', if the data vertex $\text{argmax}_{v' \in CS(u')} f_\omega(\boldsymbol{h}_{v'})$ has been selected, the data vertex v'' with the second largest value of $f_\omega(\boldsymbol{h}_{v''})$ will be selected and continue in the same manner. If all data vertices in $CS(u')$ have been selected, we will change the corresponding vertex of preselected query vertex to ensure that the corresponding vertex for each query vertex is from its own candidate set. Due to the properties of candidate set $CS(u)$ and candidate substructure G_{sub} mentioned in Section 4 that (1) for all query vertex u, $|CS(u)| \ge 1$; (2) the size of candidate substructure is greater than query graph in terms of number of vertices, *i.e.,* $|V(G_{sub})| \ge |V(q)|$, *WEst* can always find the corresponding vertex for each query vertex. Finally, the vertex sets $V'(q)$ and $V'(G_{sub})$ are obtained, and the Wasserstein-1 distance between can be minimized by minimizing $-\mathcal{L}_w$ in Equation 9. As in generative adversarial networks (GANs), training *WEst* with the discriminator introduced in this subsection can be regarded as a two-player game. The learning objectives and training procedure are introduced in the next subsection.

5.6 Learning Objectives and Training

There are two learning objectives in *WEst*, one is to reduce the differences between learned counts $\hat{c}(q)$ and the ground-truth counts $c(q)$, and another objective is described in Equation 9 that aims to explore a better lower bound of the Wasserstein distance. To optimize *WEst* for better performance on subgraph matching counts prediction, the loss function is designed based on *q-error* discrepancy. Specifically, the loss function is defined as:

Algorithm 3: Training Procedure of *WEst*

Input: training query graph set Q_t, data graph G, estimation network f_θ , discriminator f_ω, learning rates α_θ, α_ω, batch size n_{batch}, number of training iterations $iter_\omega$.

1 Initialize optimizers opt_θ, opt_ω with learning rates α_θ, α_ω.
2 Separate Q_t into batches $\{Q_b = \{q^{(i)}\}\}$ with n_{batch} query graphs.
3 **for** $Q_b \in Q_t$ **do**
4 **for** $q^{(i)} \in Q_b$ **do**
5 Generate $\mathcal{G}_{sub}^{(i)}$ for $q^{(i)}$
6 $X_q^{(i)} \leftarrow$ initial features of vertices in query graph $q^{(i)}$
7 **for** $j = 1, ..., |\mathcal{G}_{sub}^{(i)}|$ **do**
8 $X_{sub}^{(j)} \leftarrow$ initial features of vertices in $G_{sub}^{(j)}$
9 $\boldsymbol{H}_q^{(i)}, \boldsymbol{H}_{sub}^{(j)}, \hat{c}_j(q^{(i)}) \leftarrow f_\theta(q^{(i)}, G_{sub}^{(j)}, X_q^{(i)}, X_{sub}^{(j)})$
10 **for** $iter_\omega$ **do**
11 Sample $V'(q^{(i)})$ and $V'(G_{sub}^{(j)})$
12 Update ω by opt_ω minimizing $-\mathcal{L}_w$ in Eq. 9
13 Compute $\mathcal{L}_w(q^{(i)}, G_{sub}^{(j)})$
14 $\hat{c}(q^{(i)}) = \sum_{j=1}^{|\mathcal{G}_{sub}|} \hat{c}_j(q^{(i)})$
15 Compute $\mathcal{L}_c(q^{(i)})$ using Eq. 10
16 Update θ by opt_θ with $\sum_{q^{(i)} \in Q_b} \mathcal{L}(q^{(i)})$ defined in Eq. 11

$$\mathcal{L}_c(q) = max(\frac{c(q)}{\hat{c}(q) + \varepsilon}, \frac{\hat{c}(q)}{c(q)}), \tag{10}$$

where ε is a scalar with a small value (*e.g.* $\varepsilon = 10^{-9}$) that avoids the predicted subgraph counts $\hat{c}(q)$ approaching 0 when the ground-truth counts $c(q)$ are small. Finally, we define the overall loss function to optimize the parameters in *WEst* as follows:

$$\mathcal{L}(q) = (1 - \beta)\mathcal{L}_c(q) - \frac{\beta}{|\mathcal{G}_{sub}|} \sum_{G_{sub}^{(i)} \in \mathcal{G}_{sub}} \mathcal{L}_w(q, G_{sub}^{(i)}), \tag{11}$$

where $\beta \in (0, 1)$ is a coefficient to balance two loss functions.

With the loss functions defined in Equations (9) to (11), we can introduce *one epoch* of the training procedure of *WEst* illustrated in Algorithm 3. In detail, in each training epoch, the training query graph set Q_t are separated into batches $\{Q_b\}$. Then training procedure iterates over all batches. For each query graph $q^{(i)} \in Q_b$, *WEst* generates the candidate substructures $\mathcal{G}_{sub}^{(i)} = \{G_{sub}^{(j)}; j = 1, ..., |\mathcal{G}_{sub}^{(i)}|\}$ and computes the vertex representations $\boldsymbol{H}_q^{(i)}, \boldsymbol{H}_{sub}^{(j)}$ and predicted subgraph counts $\hat{c}_j(q^{(i)}$ (lines 5 to 9). Next, the parameters of the discriminator f_ω are updated by minimizing the $-\mathcal{L}_w$ for $iter_\omega$ iterations (lines 10 to 12). At the end of all iterations, $\mathcal{L}_w(q^{(i)}, G_{sub}^{(j)})$ and $\mathcal{L}_c(q^{(i)})$ are computed for each query graph and its substructures (lines 13 to 15). After such computations on all query graphs in batch Q_b are done, the parameters θ of the estimation network of *WEst* is updated by minimizing $\sum_{q^{(i)} \in Q_b} \mathcal{L}(q^{(i)})$ with the loss function defined in Equation (11) (line 16). Please note that when updating θ, the parameters ω in discriminator f_ω are fixed.

There is one problem with the loss function in Equation (9) that it could lead to a trivial case that all representations have the same value. In that case, *WEst*'s estimation performance would be degraded. To solve this problem, we first train the estimation network in *WEst* with the loss function in Equation (10), and then apply the training procedure in Algorithm 3 to train the discriminator and tune the parameters in the estimation network for extra epochs.

5.7 Analysis

We analyze the expressive power of the neural network estimator *WEst* and the time complexity of *NeurSC* in this subsection.

Expressive Power. Here we prove that the estimation network of *WEst* is as powerful as the Weisfeiler-Lehman (WL) [49] isomorphism test. The Weisfeiler-Lehman (WL) algorithm is employed for the graph isomorphism problem. There are several variants of the WL algorithm. The 1-dimensional WL algorithm is the most standard variant that assigns a color to each node, aggregates the colors of vertices and their neighbors and refines colors until convergence. Due to the space limit, please refer to [49] for the formal definition. The expressive power of the intra-graph neural network GIN has been proven in the following lemma.

Lemma 5.1 ([108]). *For all $K \in \mathbb{Z}^+$, there exist parameters of K layered GINs such that if the degrees of the nodes are bounded by a constant and the size of the support of node features is finite, for any graphs g_1 and g_2, if the 1-WL algorithm outputs that g_1 and g_2 are "non-isomorphic" within K rounds, the embeddings $\boldsymbol{h}_{g_1}$ and $\boldsymbol{h}_{g_2}$ computed by the GIN are different.*

The proof of Lemma 5.1 can be found in [81, 108]. Recall that the estimation network of *WEst* consists of the intra-graph GIN and inter-graph attentive network, whose output representation is the concatenation of the embeddings produced by these two networks, *i.e.,* $\boldsymbol{h}_u = \boldsymbol{h}_u^{intra} \| \boldsymbol{h}_u^{inter}$.

Proposition 5.2. *If $\boldsymbol{h}_1 \in \mathbb{R}^{dim}$ and $\boldsymbol{h}_2 \in \mathbb{R}^{dim}$ are different, then for any $\boldsymbol{h}_3, \boldsymbol{h}_4 \in \mathbb{R}^{dim'}$, $\boldsymbol{h}_1 \| \boldsymbol{h}_3$ and $\boldsymbol{h}_2 \| \boldsymbol{h}_4$ are different, where $\|$ is the concatenation operation.*

We can easily derive from Lemma 5.1 and Proposition 5.2 that:

Theorem 5.3. *For all $K, K' \in \mathbb{Z}^+$, there exist parameters of the estimation network in WEst with K layered intra-GNN and K' layered inter-GNN such that if the degrees of the nodes are bounded by a constant and the size of the support of node features is finite, for any graphs g_1 and g_2, if the 1-WL algorithm outputs that g_1 and g_2 are "non-isomorphic" within K rounds, the embeddings $\boldsymbol{h}_{g_1}$ and $\boldsymbol{h}_{g_2}$ computed by WEst are subsequently different.*

The proof of Theorem 5.3 is immediate. It can be concluded from Theorem 5.3 that the estimation network of *WEst* is as powerful as the 1-WL isomorphism test.

Time Complexity. The time complexity of *NeurSC* for subgraph counting consists of the time cost for candidate set generation, candidate substructure extraction, intra-graph neural network and inter-graph neural network. Please note that the Wasserstein discriminator is not involved in the query response process but only in the training procedure. The time complexity of candidate set generation is determined by the neighbor radius r. When $r = 1$, *i.e.,* compare the 1-hop neighbors' labels, then the time complexity of the candidate set generation is $O(|V(q)| \times |V(G)| + \sum_{u \in V(q)} \sum_{v \in V(G)} (d(u) \times d(v) + O(d(u), d(v))))$, where $O(\cdot)$ is the time complexity to check the existence of a semi-perfect match [89].

With the obtained candidate sets, the time complexity of computing the induced subgraph is $O(max_{v \in CS(q)} d(v) \times |CS(q)|)$. The time complexities of intra and inter-graph neural networks used in *WEst* are both $O(|E|)$ [104], where $|E|$ denotes the number of edges in the graph. The intra-GNN is applied on the query graph q and candidate substructure G_{sub}, and inter-GNN is applied on the constructed bipartite graph G_B according to q and G_{sub}. Therefore, the intra-GNN has time complexity $O(|E(q)|+|E(G_{sub})|)$, and the inter-GNN has time complexity $O(|E(G_B)|)$ which is equivalent to $O(|CS(q)|)$. Finally, we can obtain the overall time complexity of *NeurSC* as $O(|V(q)| \times |V(G)| + |CS(q)| + |E(q)| + |E(G_{sub})|)$. Therefore, with a fixed-size query graph, *NeurSC*'s time complexity is linear with the number of vertices of the data graph. Please note that the time complexity of *WEst* is also influenced by its hyperparameters such as the hidden dimensions, since they are adjustable and not related to the graph properties, we omit them in time complexity analysis.

5.8 Discussion

Same as other learning-based methods for subgraph counting [52, 117] and cardinality estimation on *RDBMS* [51, 88, 102, 109**?**], the error of *NeurSC* cannot be bounded due to the utilization of neural networks. The architecture of *NeurSC* enables the trade-off between efficiency and accuracy. Suppose there are more than more candidate substructures obtained by substructure extraction, *i.e.,* $|\mathcal{G}_{sub}| > 1$, the query time can be reduced by only performing *WEst* on a randomly sampled subset of $\mathcal{G}_{sub}$, denoted by $\mathcal{G}'_{sub} = \{G^{(i)}_{sub}; i = 1, ..., |\mathcal{G}'_{sub}|\}$, and $|\mathcal{G}'_{sub}| < |\mathcal{G}_{sub}|$. The estimation of subgraph counts of q is then computed as $\hat{c}'(q) = \frac{|\mathcal{G}_{sub}|}{|\mathcal{G}'_{sub}|} \sum_{i=1}^{|\mathcal{G}'_{sub}|} \hat{c}_i(q)$, *i.e.,* multiplying the sum of the estimated subgraph counts of q on the selected substructures in $\mathcal{G}'_{sub}$ by $\frac{|\mathcal{G}_{sub}|}{|\mathcal{G}'_{sub}|}$. Since the subset $\mathcal{G}'_{sub}$ is generated by uniformly sampling $G^{(i)}_{sub} \in \mathcal{G}_{sub}$, each substructure $G^{(i)}_{sub}$ has the same probability of $\frac{|\mathcal{G}'_{sub}|}{|\mathcal{G}_{sub}|}$ to be selected into the subset. Let's denote $r_s = |\mathcal{G}'_{sub}|/|\mathcal{G}_{sub}|$ as the sample rate. Then, the expectation of $\hat{c}'(q)$ is computed as:

$$\mathbb{E}[\hat{c}'(q)] = \frac{1}{r_s} \sum_{i=1}^{|\mathcal{G}'_{sub}|} \hat{c}_i(q) = \frac{1}{r_s} \sum_{i=1}^{|\mathcal{G}_{sub}|} r_s \cdot \hat{c}_i(q) = \sum_{i=1}^{|\mathcal{G}_{sub}|} \hat{c}_i(q) \qquad (12)$$

As introduced in Section 5.4, *NeurSC* predicts the subgraph counts as $\hat{c}(q) = \sum_{i=1}^{|\mathcal{G}_{sub}|} \hat{c}_i(q)$. Therefore, $\hat{c}'(q)$ is an unbiased estimator of $\hat{c}_i(q)$. By adjusting the sample rate r_s, *NeurSC* can make a trade-off between efficiency and accuracy of subgraph counting. In contrast, *LSS* uses fixed number of substructures (the number of query vertices) for subgraph counting which cannot be adjusted. We demonstrate the detailed empirical evaluation in Section 6.4.

6 EXPERIMENT

6.1 Experiment Setup

Dataset. We use 7 real-life datasets for this experimental study. In this paper, we focus on subgraph counting with the *connected* query graphs. The subgraph counts of a disconnected graph can be obtained by multiplying the estimated counts of its connected components. Specifically, the statistics of data graphs used in this work are listed in Table 2. Human is used in both [73, 89] , HPRD and DBLP are used in [89], and the other data graphs are used in both [89, 117]. For fair comparison, we also use the same query graphs used in [89, 117]. The details of the query graphs are demonstrated in Table 3. We denote the set of query graphs with i vertices as Q_i. We use the method GraphQL [33], with the implementation in [2], to obtain the ground-truth subgraph isomorphism counts. The query graphs whose ground-truth counts can be computed within 30 minutes are selected for evaluation in our experiment.

Table 2: Statistics of Data Graphs

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	d
Yeast	3,112	12,519	71	8.0
Human	4.674	86,282	44	36.9
HPRD	9,460	34,998	307	7.4
Wordnet	76,853	120,399	5	3.1
DBLP	317,080	1,049,866	15	6.6
EU2005	862,664	16,138,468	40	37.4
Youtube	1,134,890	2,987,624	25	5.3

Table 3: Details of Query Graphs

Dataset	# Queries	Query Sizes	Counts Range
Yeast	1,632	{4, 8, 16, 24, 32}	$[10^0, 10^{11}]$
Human	339	{4, 8, 16}	$[10^0, 10^{10}]$
HPRD	1,000	{4, 8, 16}	$[10^0, 10^4]$
Wordnet	600	{4, 8}	$[10^1, 10^9]$
DBLP	600	{4, 8}	$[10^3, 10^8]$
EU2005	372	{4, 8}	$[10^4, 10^9]$
Youtube	811	{4, 8, 16}	$[10^0, 10^{11}]$

Compared Methods. To demonstrate the performance of our proposed *NeurSC*, we use the approaches introduced in the recently developed benchmarking framework G-CARE [73] including *CharacteristicSets (CSet)* [69], *SumRDF* [86], *Correlated Sampling (CS)* [92], *WanderJoin (WJ)* [50], *Join Sampling with Upper Bounds (JSUB)* [119]. We also include the comparison with the learning-based approach in [53] and the state-of-the-art Learned Sketch for Subgraph Counting (*LSS*) [117]. Please note that *LSS* rather than *ALSS* , *i.e.,* without active learner, is compared in this experiment, because *LSS* is used as the default method for evaluations in the original paper [117]. The effectiveness comparison of the active learner can be found in Figure 10 of [117]. Since the model in [53] is not named in the origin paper, we name it as *NSIC* by its title Neural Subgraph Isomorphism Counting. In this experiment, we choose the Dynamic Intermedium Attention Memory Network (DIAMNet) as the interaction network, and RGIN and RGCN as the graph learning models of *NSIC*, denoted as *NSIC-I* and *NSIC-C* respectively.

The implementation of G-CARE and *NSIC* are from [3] and [4] respectively. We implement *LSS* ourselves since the code is currently unavailable. We keep the default settings of the methods in G-CARE and *NSIC*. Particularly, we set the time out limit as 5 minutes for methods in G-CARE. For *LSS*, we use the enhanced label embedding produced by `ProNE` [1, 113] as the initial features of query vertices. We set training epoch = 50, learning rate = 10^{-3}, batch size = 2 and Adam penalty = 10^{-5} for *LSS*, which are in the tuning range given in [117]. We keep other settings such as layers of neural networks, hidden dimensions and dropout rate same as in [117].

To investigate the effectiveness of each component of *NeurSC*, we include the following variants in the experiments: (1) a variant which only has the intra-graph neural network, denoted by *NeurSC-I*, (2) a variant which has dual networks (both intra and inter-GNN) but does not have Wasserstein discriminator, denoted by *NeurSC-D*.

Experiment Settings. We introduce the hyperparameter settings of *NeurSC* in this paragraph. The dimension of initial features $dim_0 = 64$. The intra-GNN and inter-GNN both have 2 layers and output the representations with dimension $dim_{K/K'} = 128$. The prediction of subgraph isomorphism counts is then computed by a 4-layer MLP with the learned representations. Wasserstein discriminator is modeled as a 3-layer MLP whose output dimension is 1. The discriminator is trained *once* for each input pair, *i.e.,* $iter_\omega = 1$. Adam [42] is selected as the optimizers for both *WEst* and Wasserstein discriminator. The learning rate is set to 10^{-3}. The batch size is set to 20. The loss balance coefficient β in Equation (11) is tuned in range $[0.5, 0.99]$, and the training epochs $\in [30, 150]$ for different data graphs. To evaluate the performance of *NeurSC*, we split the query graphs into 80% for training and 20% for testing. *NeurSC*'s performance on the entire query set is obtained by 5-fold validation. Our *NeurSC* is built with C++ and Python, the extraction module is implemented with C++ while *WEst* is coded with PyTorch framework in Python.

Experiment Environment. The experiments are conducted on the servers running RHEL 7.7 system, which have Intel Xeon Gold 6238R 2.2GHz 28cores CPU and NVIDIA Quadro RTX 6000 Passive GPU with 180GB RAM (Six Channel).

Evaluation Metrics. In this experiment, we evaluate the performance of the compared methods with the estimation accuracy and efficiency. Particularly, we measure the accuracy using *q-error* [66] defined as $max(\frac{max(1,c)}{max(1,\hat{c})}, \frac{max(1,\hat{c})}{max(1,c)})$, where $\hat{c}$ is the estimated subgraph counts and c is the ground-truth counts. *q-error* is also used for evaluation in [73, 117]. To measure the efficiency of the compared methods, we report the average elapsed time for queries.

6.2 Accuracy Comparison

The results of accuracy comparison between *NeurSC* and the baseline methods are demonstrated in Figure 7, in which the *q-error* distribution of each method is shown in the box-plot. We represent the under/overestimation explicitly on the y-axis of the result figures. In our box-plot, the upper bound (*resp.,* lower bound) of the box is 75% (*resp.,* 25%) percentile of the *q-error*, the whiskers contain *all q-errors* from minimum (underestimate) to maximum (overestimate), and the line in the box is the median. Overall, *NeurSC* consistently outperforms the other compared methods. Specifically, NeurSC could reduce up to 95% of the *q-error* compared to the state-of-the-art technique *LSS*.

Compare with Methods in G-CARE. In our experiment, we observe comparable results to [117] in methods surveyed in G-CARE [73] are outperformed by the learning-based methods. Among these methods, sampling-methods (*CS, WJ, JSUB*) suffer from the *sampling failure*, *i.e.,* failure to generate valid sample from data graph that (partially) matches the query graph, on complex query and data graphs, which leads to underestimated results for these methods. Besides the summary-based method *CSet* and *SumRDF*

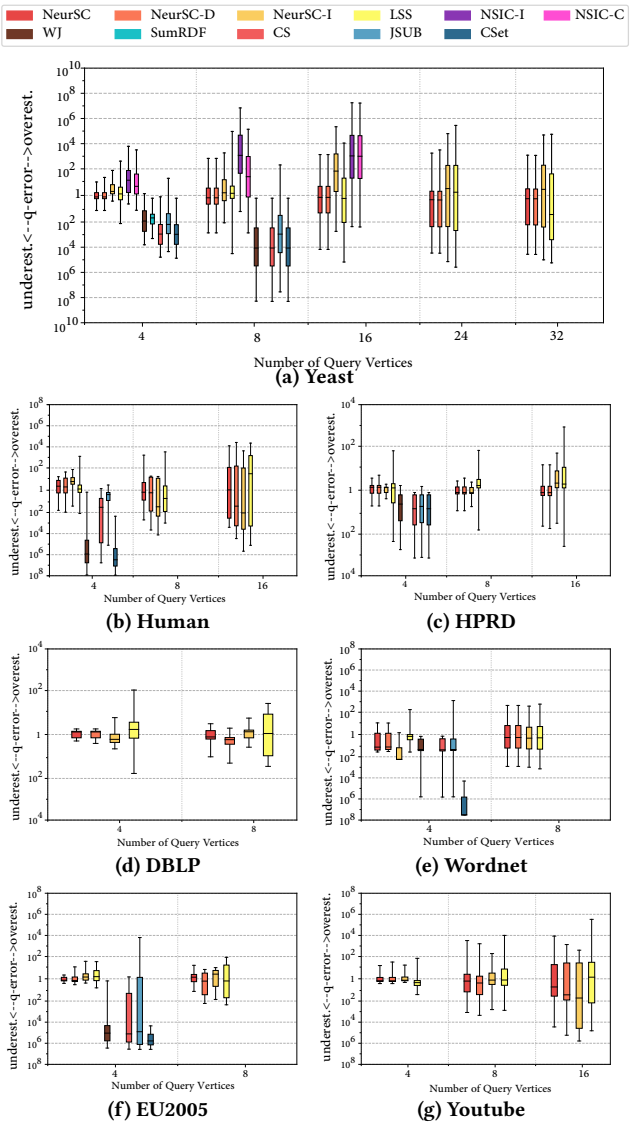


Figure 7: *q-error* Evaluation

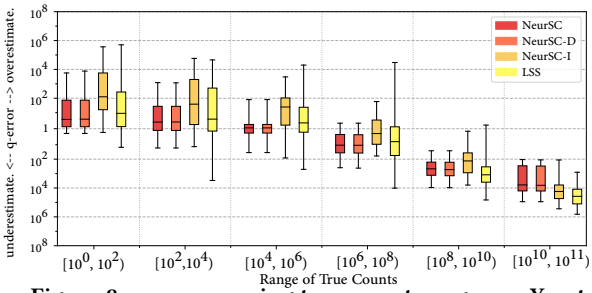
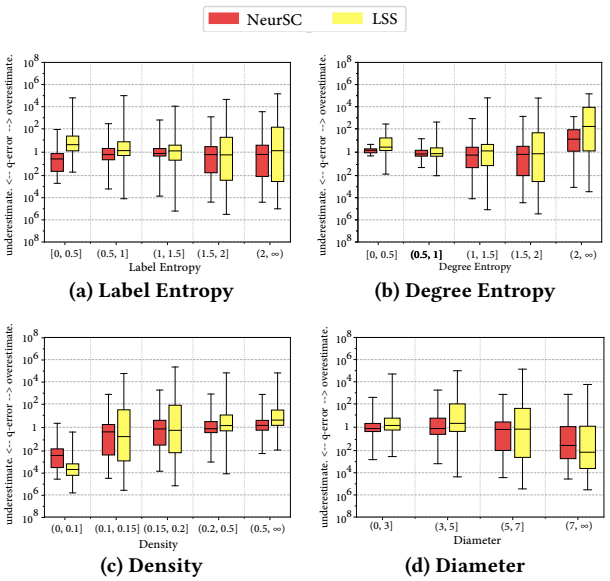


Figure 8: *q-error* varying true count ranges on Yeast

underestimate the subgraph counts because of their assumption of distributions of the subgraph matches. *SumRDF* consistently time out on the large query graphs because it needs to search for exact matches on the summarized data graph.

Compare with Learning-based Methods. The *sampling failure* occurs since the sampling-based methods cannot explore enough



(a) Label Entropy (b) Degree Entropy

(c) Density (d) Diameter

Figure 9: *q-error* varying query characteristics on Yeast

valid samples on graphs with complex label and topology distributions. Conversely, neural networks are capable of modeling complex data distributions. As a result, the learning-based methods generally deliver better performance.

NSIC is designed for relatively small graphs. Therefore, when predicting the subgraph counts for the data graphs used in this paper, *NSIC* easily exceeds 5 minutes. As a result, we only report the results of *NSIC* on Yeast data graph with query sets Q_4, Q_8 and Q_{16} in Figure 7a. We observed that despite the high time complexity, *NSIC* also fails to distinguish the differences between the query graphs. Therefore, as shown in Figure 7a, both *NSIC-I* and *NSIC-C* output similar estimated counts for almost all given query graphs, which leads to the poor estimation performance of NSIC.

Among the learning-based models, *NeurSC* consistently outperforms *LSS* on all data graphs with all query sets. Especially on Yeast data graph with Q_{32}, the first/third quartile and the maximum/minimum of *q-error* are all two orders of magnitude less than that of *LSS*. This performance improvement emphasizes the importance of data graph information, which is sufficiently exploited in *NeurSC*, for subgraph counting. Intuitively, the query graphs with different subgraph counts in data graph should have different candidate substructures. *NeurSC* is able to capture these differences and exploit the substructures' label, topology and size information with intra and inter-GNNs, and hence achieve better estimation accuracy. Meanwhile, *LSS* only learns the representations on the query graph and somehow misses some information in data graph.

The close observations on the 1,632 queries on Yeast are shown in Figure 8 and Figure 9. Figure 8 demonstrates the *q-error* distribution of compared methods on Yeast varying the range of ground-truth counts. The results show that *NeurSC* outperforms *LSS* on all ground-truth count ranges. Furthermore, we evaluate the performance of *NeurSC* varying the characteristics of queries such as label entropy, degree entropy, density and diameter. Specifically, The label entropy is defined as $\sum_{l \in L_q} p(l)log(p(l))$, where $p(l)$ denotes the fraction that a query vertex is attached to label l, and degree

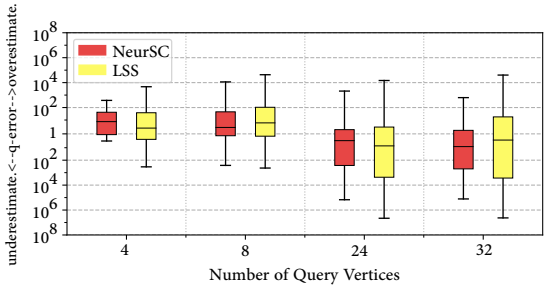


Figure 10: Robustness evaluation

entropy is defined as $\sum_{d \in d_q} p(d)log(p(d))$, where $p(d)$ is the fraction that a query vertex has degree of d. The results are shown in Figure 9. Generally, both *NeurSC* and *LSS* have better performance on queries with lower degree entropy, larger density and smaller diameter. Meanwhile, *NeurSC* consistently outperforms *LSS*, and the advancement of *NeurSC* becomes greater on the queries with large label and degree entropy. These results not only prove the benefits of learning the individual and joint distributions of both query and data graphs, but also indicate that *NeurSC* has better ability on solving complex queries.

Compare with Variants. Figure 7 also illustrates the performance comparison between *NeurSC* and its variants *NeurSC-D* and *NeurSC-I*. Though in some query sets, *e.g.*, Q_8 on Youtube in Figure 7g, surprisingly, *NeurSC-I* has the best accuracy, while *NeurSC* outperforms its variants in most cases. Compared to the performance of *NeurSC-D*, *NeurSC* reduces the min/max *q-error* values on most query sets, and reduces the range of quantiles in some cases: see Figures 7b and 7f. The performance gap between the two variants is also significant, we can find in Figure 7a that *NeurSC-D* predicts the subgraph counts much more accurately than *NeurSC-I*. We can conclude from these results that (1) the Wasserstein discriminator boosts the estimation accuracy of *NeurSC* on most query sets in most data graphs (2) the inter-graph neural network plays a critical role in estimating the subgraph counts.

Robustness Evaluation. We also conduct an experiment to evaluate the robustness of *NeurSC* and *LSS* on Yeast data graph. We train these two models on Yeast with the query graphs with 16 vertices (Q_{16}), then evaluate the *q-error* of trained models with query sets with $\{4, 8, 24, 32\}$ vertices which are unseen during the training phase. The results are shown in Figure 10. Since both models are trained on Q_{16}, in general, their predictions are both overestimated on Q_4 and Q_8 and underestimated on Q_{24} and Q_{32}. Compared with *LSS*, *NeurSC* has better performance in term of *q-error*. The superiority of *NeurSC* indicates its greater robustness. This superiority comes from the adaptive utilization of data graph information in *NeurSC*. Therefore, *NeurSC* could make estimations based on both query graph and corresponding candidate substructures for different queries adaptively and distinguishably. Given the fact that it is impractical to train the model for each query set with different number of vertices, *NeurSC* is both more applicable and robust for real-life subgraph counting tasks.

Effectiveness Evaluation of Substructure Extraction. To test the effectiveness of the substructure extraction, the performance following variants of *NeurSC* and *NSIC* [52] are evaluated on Yeast: (1) *NeurSC w/o SE* which is our model without the substructure

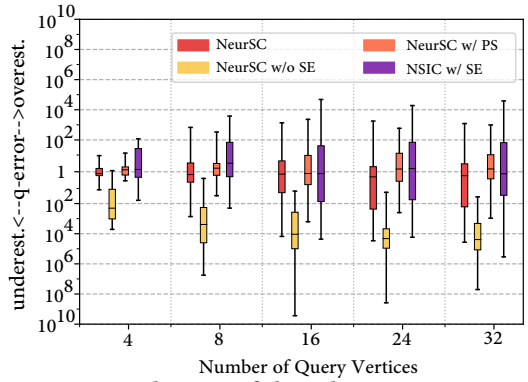


Figure 11: Evaluation of the substructure extraction

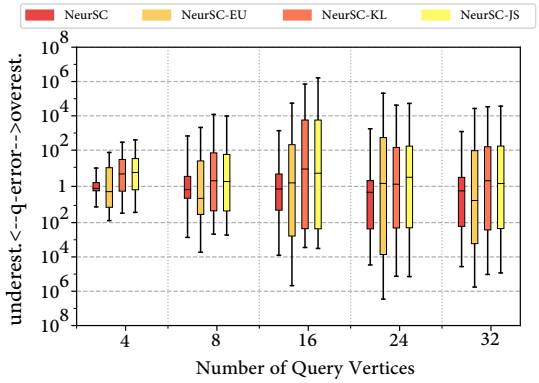


Figure 12: Evaluation varying distance metrics

extraction. Please note that without the substructure extraction module, *NeurSC w/o SE* estimates the subgraph counts only based on the intra-graph neural network. (2) *NeurSC w/ PS* is a variant of *NeurSC* with the 'perfect' substructure(s) constructed from the ground-truth matches. (3) *NSIC w/ SE* is a variant of *NSIC* with substructure extraction module. Figure 11 illustrates the results. It can be found that, with only intra-graph neural networks on query and entire data graph, *NeurSC w/o SE* cannot distinguish the differences between query graphs and hence incurs unsatisfactory accuracy. *NeurSC w/ PS* achieves better accuracy in terms of *q-error* compared to NeurSC. However, it is time consuming to obtain the 'perfect' substructure(s). Meanwhile, *NeurSC* consistently outperforms *NSIC w/ SE*. Specifically, the max/min *q-error* of *NSIC w/ SE* is nearly two orders of magnitudes greater than that of *NeurSC* on Q_{32}. These observations prove the necessity of substructure extraction and the contribution and advancement of our carefully designed model based on the substructure extraction module.

Evaluation varying Distance Metrics in Discriminator. We evaluate the effectiveness of Wasserstein distance by replacing it with other three popular distance metrics: Euclidean distance, Kullback-Leibler (KL) divergence and Jensen-Shannon (JS) divergence in the discriminator. Specifically, the variants *NeurSC-EU*, *NeurSC-KL* and *NeurSC-JS* select the correspondence query and data vertex pairs that are closest to each other in representation space and minimize the distance between these representations as training objective according to Euclidean distance, KL-divergence and JS-divergence respectively. The results are illustrated in Figure 12. Among three variants, *NeurSC-KL* and *NeurSC-JS* have similar performances which are better than *NeurSC-EU* in terms of

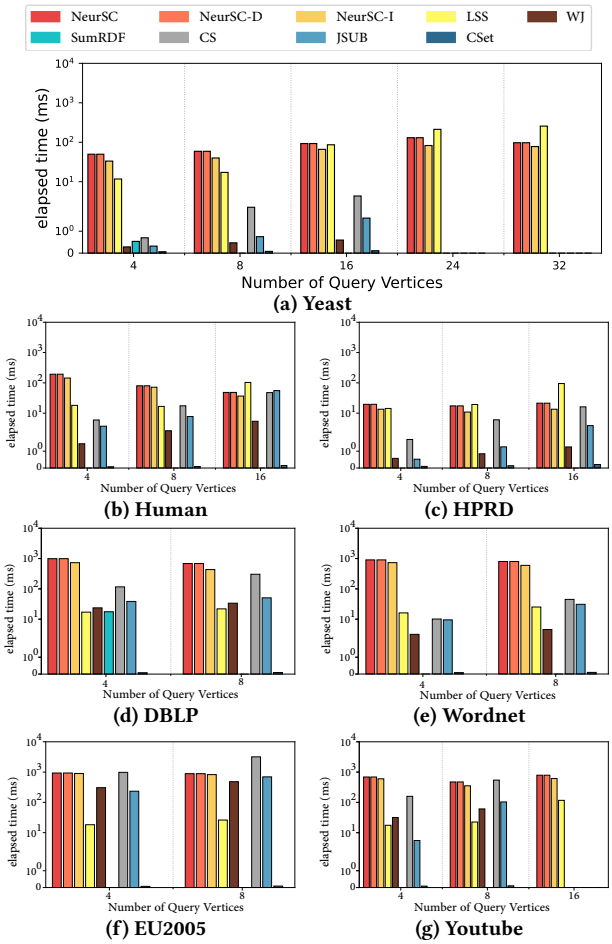


Figure 13: Query Time Comparison

q-error. *NeurSC* outperforms the compared variants on all query sets, which demonstrates the superiority of Wasserstein distance.

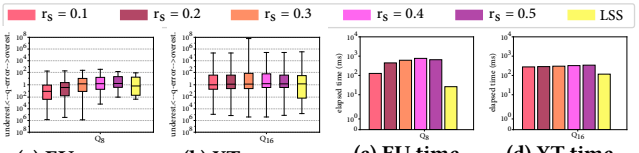
6.3 Efficiency Comparison

In this subsection, we evaluate the efficiency of *NeurSC* and its variants, *LSS* and methods surveyed in G-CARE. Specifically, we report the query processing time of compared methods and the training time of *NeurSC* and *LSS*.

Query Processing Time. The results of average query processing time comparison are illustrated in Figure 13. In data graphs such as Yeast (Figure 13a), Human (Figure 13b) and HPRD (Figure 13c), the learning-based methods do not have many advantages because summary-based and sampling-based methods are efficient on the small data graph. *CSet* is consistently the most efficient because it performs neither matching nor sampling for the online query. When comparison is between *NeurSC* and *LSS* in Figure 13a, it can be observed that *LSS* is more efficient when the query graph is small (Q_4) while *NeurSC* is more efficient when the number of query vertices is large (Q_{32}). This is because that (1) with the growth of the query graph size, the number and size of substructures to be processed by *LSS* both grows linearly, which leads to the quadratic time growth, (2) the time cost of *NeurSC* does not increase and can even reduce with the decreasing size of candidate substructures.

Table 4: Training time (second) for one epoch.

Data Graph	LSS	NeurSC-I	NeurSC-D	NeurSC
Yeast	8.53	58.63	67.43	109.71
Human	14.51	107.11	152.05	257.61
HPRD	10.43	8.80	28.68	88.80
DBLP	13.21	360.96	472.59	793.45
Wordnet	13.96	207.54	248.12	451.09
Youtube	9.47	217.60	263.89	609.22
EU2005	12.99	391.58	499.38	1023.91



(a) EU *q-error* **(b) YT *q-error*** **(c) EU time** **(d) YT time**

Figure 14: Trade-off between efficiency and accuracy

For queries on data graphs like EU2005 (Figure 13f) and Youtube (Figure 13g), *NeurSC* could produce the estimation within around 1 second which is similar to *WJ* and *CS*. In these graphs, *LSS* is usually more efficient than *NeurSC* since it only needs to embed the substructures from query graph, whose time cost is independent to the size of data graph. However, this architecture of *LSS* also limits its estimation accuracy and robustness, which has been discussed in Section 6.2. Furthermore, the theoretical analysis in Section 5.7 indicates that the time cost of *NeurSC* should be linear with the number of data vertices. In practice, since the filtering and extraction process is quite efficient, the time cost is usually determined by the size of candidate substructures. Therefore, the exact time cost grows sub-linearly *w.r.t.* the size of data graph. For example, EU2005 is more than 200 times and 10,000 larger than Yeast in terms of number of vertices and edges. But the average query processing time for Q_4 on EU2005 is only 10 times greater than that on Yeast, which validates the scalability of *NeurSC*.

Training time. We compare the training time for one epoch of *LSS*, *NeurSC* and its two variants for a query set with 4 vertices which are reported in Table 4. *LSS* is the most efficient *w.r.t.* the training time. The training time of *NeurSC* is acceptable given its superior estimation accuracy. Like the query processing time, the training time also grows sub-linearly to the sizes of data graphs, which again validates the scalability of *NeurSC*. Please note that though *NeurSC* costs more time for training than its variants, as mentioned in Section 5.6, we would train *NeurSC* without Wasserstein discriminator (*i.e., NeurSC-D*) for several epochs and then involve the discriminator to tune the parameters. Hence, the exact training time of *NeurSC* is less than the multiplication of the number of epochs and the training time for one epoch reported in Table 4.

6.4 Trade-Off Analysis

As mentioned in Section 5.8, *NeurSC* can make a trade-off between efficiency and accuracy by adjusting the sample rate r_s on the candidate substructures. The number of sample substructures $|\mathcal{G}'_{sub}| = \lceil r_s \times |\mathcal{G}_{sub}| \rceil$, where $\lceil \cdot \rceil$ is the ceiling function. The subgraph counts estimation on the data graph is obtained by dividing the summation of the subgraph counts on these sampled substructures computed by *WEst*, *i.e.,* $\sum_{G^{(i)}_{sub} \in \mathcal{G}'_{sub}} \hat{c}_i(q)$, by sample rate r_s. We vary the sample rate r_s from 0.1 to 0.5 with step width 0.1 and show both the *q-error* distribution and query processing time for Q_{16} on Youtube (YT) and Q_8 on EU2005 (EU). The results are demonstrated in Figure 14. We find that when $r_s = 0.4$, the *q-error* of *NeurSC* on EU2005 is close to that of LSS (see Figure 14a). Surprisingly, when $r_s = 0.1$, the *NeurSC* outperforms LSS *w.r.t.* the *q-error* on Youtube (see Figure 14b). Meanwhile, the query time of *NeurSC* is only 2× of that of LSS (see Figure 14d). These results indicate that *NeurSC* can fit the requirements in various applications.

7 RELATED WORKS

Subgraph Matching. There are two major categories of subgraph matching methods. The backtracking search based algorithms [10, 15, 16, 19, 28, 29, 37, 40, 84], which follow the *preprocessing- enumeration* framework to obtain matches for query graphs. The other category is the join-based methods [45, 46, 110]. Among these methods, it has been proved that the WCOJ [70] based database system LogicBlox [6] outperformed other database systems. Besides, there are several algorithms [79, 116, 118] utilize indexing-enumeration framework. Please refer to survey papers [47, 89] for more details.

Subgraph Counting. Subgraph counting [78] has been extensively studied for either exact or approximate solutions. There are two categories for exact counting approaches: enumeration and analytic approaches. The enumeration approaches [63, 64] enumerate all subgraphs to obtain the exact counts. The analytic approaches [34, 35, 58, 59, 72] obtain the counts for query graphs by leveraging the counts of smaller size queries in an analytical way.

Graph Neural Netoworks. Graph neural networks (GNNs) have shown enourmous success. Typically, GNNs learn the node representations with iterative aggregation of neighborhood information for each node, e.g., GCN [43], GAT [91] and GraphSAGE [27]. Despite these models, there have been a great number of GNNs proposed for wide range of real-life applications [30, 41, 93, 96, 97, 108, 112], please refer to [104] for comprehensive survey.

Machine Learning Methods for Subgraph Matching/Counting. Recently, there have been several research attempts [18, 53, 55, 98, 117] to develop subgraph matching and subgraph counting algorithms on a learning basis. These methods focus on the approximation of counting of isomorphic subgraphs [18, 53, 117] or producing approximate matching results [55].

8 CONCLUSION

In this paper, we proposed a learning-based model *NeurSC* to estimate the subgraph isomorphism counts. *NeurSC* first adaptively extracts the candidate substructures for each query graph and then estimates the subgraph counts with a carefully-designed neural network *WEst*. *WEst* has intra and inter-graph neural networks which could capture the information within and between the query graph and candidate substructures. In addition, *WEst* exploits the Wasserstein estimator to leverage the node correspondence information. Our extensive experiments demonstrated the effectiveness, robustness, scalability and flexibility of our proposed *NeurSC* on subgraph counting task.

ACKNOWLEDGMENTS

Ying Zhang is the corresponding author who is supported by ARC DP210101393, Lu Qin is supported by ARC FT200100787 and DP21010 -1347, Wei Wang is supported by HKUST(GZ) Grant No: G0101000028, Wenjie Zhang is supported by ARC DP200101116 and FT210100303.

REFERENCES

[1] 2019. https://github.com/THUDM/ProNE.
[2] 2020. https://github.com/RapidsAtHKUST/SubgraphMatching.
[3] 2020. https://github.com/yspark-dblab/gcare.
[4] 2020. https://github.com/HKUST-KnowComp/NeuralSubgraphCounting.
[5] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*. IEEE, 1–10.
[6] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1371–1382.
[7] Martín Arjovsky and Léon Bottou. 2017. Towards Principled Methods for Training Generative Adversarial Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
[8] Martín Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. *CoRR* abs/1701.07875 (2017).
[9] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*. IEEE, 91–100.
[10] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1199–1214.
[11] Manuel Bodirsky. 2015. Graph homomorphisms and universal algebra course notes. *TU Dresden* (2015).
[12] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting graphlets: Space vs time. In *Proceedings of the tenth ACM international conference on web search and data mining*. 557–566.
[13] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: Fast Motif Counting via Succinct Color Coding and Adaptive Sampling. *Proccedings of the VLDB Endowment* 12, 11 (2019), 1651–1663.
[14] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proceedings of the 2019 International Conference on Management of Data*. 18–35.
[15] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 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 (2017), 804–818.
[16] Vincenzo Carletti, Pasquale Foggia, and Mario Vento. 2015. VF2 Plus: An Improved version of VF2 for Biological Graphs. In *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9069)*, Cheng-Lin Liu, Bin Luo, Walter G. Kropatsch, and Jian Cheng (Eds.). Springer, 168–177.
[17] Xiaowei Chen and John CS Lui. 2018. Mining graphlet counts in online social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 4 (2018), 1–38.
[18] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025* (2020).
[19] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 10 (2004), 1367–1372.
[20] Angjela Davitkova, Damjan Gjurovski, and Sebastian Michel. 2021. LMKG: Learned Models for Cardinality Estimation in Knowledge Graphs. *arXiv preprint arXiv:2102.10588* (2021).
[21] Ji Gao, Xiao Huang, and Jundong Li. 2021. Unsupervised Graph Alignment with Wasserstein Distance Discriminator. In *27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery.
[22] Matt W Gardner and SR Dorling. 1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment* 32, 14-15 (1998), 2627–2636.
[23] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
[24] Nathael Gozlan, Cyril Roberto, Paul-Marie Samson, and Prasad Tetali. 2017. Kantorovich duality for general transport costs and applications. *Journal of Functional Analysis* 273, 11 (2017), 3327–3405.
[25] Martin Grohe. 2017. *Descriptive complexity, canonisation, and definable graph structure theory*. Vol. 47. Cambridge University Press.
[26] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.
[27] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.
[28] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1429–1446.
[29] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. 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*. 337–348.
[30] Yu Hao, Xin Cao, Yufan Sheng, Yixiang Fang, and Wei Wang. 2021. KS-GNN: Keywords Search over Incomplete Graphs via Graphs Neural Network. *Advances in Neural Information Processing Systems* 34 (2021).
[31] Zaïd Harchaoui and Francis Bach. 2007. Image classification with segmentation graph kernels. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.
[32] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050.
[33] Huahai He and Ambuj K Singh. 2008. 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*. 405–418.
[34] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.
[35] Tomaž Hočevar and Janez Demšar. 2017. Combinatorial algorithm for counting small induced graphs and orbits. *PloS one* 12, 2 (2017), e0171428.
[36] Kai Huang, Haibo Hu, Shuigeng Zhou, Jihong Guan, Qingqing Ye, and Xiaofang Zhou. 2021. Privacy and efficiency guaranteed social subgraph matching. *The VLDB Journal* (2021), 1–22.
[37] Alpár Jüttner and Péter Madarasi. 2018. VF2++—An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242 (2018), 69–81.
[38] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. 2004. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* 20, 11 (2004), 1746–1758.
[39] Duck Hoon Kim, Il Dong Yun, and Sang Uk Lee. 2004. A new attributed relational graph matching algorithm using the nested structure of earth mover's distance. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 1. IEEE, 48–51.
[40] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2021. Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching. In *Proceedings of the 2021 International Conference on Management of Data*. 925–937.
[41] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D Yoo. 2019. Edge-labeling graph neural network for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11–20.
[42] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[43] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
[44] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.
[45] Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable Subgraph Enumeration in MapReduce. *Proc. VLDB Endow.* 8, 10 (2015), 974–985.
[46] Longbin Lai, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2016. Scalable Distributed Subgraph Enumeration. *Proc. VLDB Endow.* 10, 3 (2016), 217–228.
[47] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of the VLDB Endowment* 6, 2 (2012), 133–144.
[48] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.
[49] AA Leman and B Weisfeiler. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya* 2, 9 (1968), 12–16.
[50] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*. 615–629.
[51] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*. 53–59.
[52] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD*

International Conference on Knowledge Discovery & Data Mining. 1959–1969.

[53] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural Subgraph Isomorphism Counting. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 1959–1969.

[54] Yangwei Liu, Hu Ding, Danyang Chen, and Jinhui Xu. 2017. Novel geometric approach for global alignment of PPI networks. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[55] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural Subgraph Matching. *arXiv preprint arXiv:2007.03092* (2020).

[56] Shmoolik Mangan and Uri Alon. 2003. Structure and function of the feed-forward loop network motif. *Proceedings of the National Academy of Sciences* 100, 21 (2003), 11980–11985.

[57] Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT press.

[58] Dror Marcus and Yuval Shavitt. 2010. Efficient counting of network motifs. In *2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*. IEEE, 92–98.

[59] Dror Marcus and Yuval Shavitt. 2012. Rage–a rapid graphlet enumerator for large networks. *Computer Networks* 56, 2 (2012), 810–819.

[60] Hermina Petric Maretic, Mireille El Gheche, Matthias Minder, Giovanni Chierchia, and Pascal Frossard. 2020. Wasserstein-based graph alignment. *arXiv preprint arXiv:2003.06048* (2020).

[61] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably Powerful Graph Networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2156–2167.

[62] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. 2018. Invariant and Equivariant Graph Networks. In *International Conference on Learning Representations*.

[63] Brendan D McKay and Adolfo Piperno. 2013. Nauty and Traces user's guide (Version 2.5). *Computer Science Department, Australian National University, Canberra, Australia* (2013).

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

[65] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment* 2, 1 (2009), 982–993.

[66] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *Proc. VLDB Endow.* 2, 1 (2009), 982–993.

[67] Christopher Morris, Gaurav Rattan, and Petra Mutzel. 2020. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems* 33 (2020).

[68] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Icml*.

[69] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 984–994.

[70] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, Jan Van den Bussche and Marcelo Arenas (Eds.). ACM, 111–124.

[71] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. 2017. Matching node embeddings for graph similarity. In *Thirty-first AAAI conference on artificial intelligence*.

[72] Mark Ortmann and Ulrik Brandes. 2016. Quad census computation: Simple, efficient, and orbit-aware. In *International Conference and School on Network Science*. Springer, 1–13.

[73] Yeonsu Park, Seongyun Ko, Sourav S. Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. 2020. G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1099–1114.

[74] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[75] Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. 2021. MaNIACS: Approximate Mining of Frequent Subgraph Patterns through Sampling. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1348–1358.

[76] N Pržulj, Derek G Corneil, and Igor Jurisica. 2006. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics* 22, 8 (2006), 974–980.

[77] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2466–2478.

[78] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. 2021. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.

[79] Carlos R Rivero and Hasan M Jamil. 2017. Efficient and scalable labeled subgraph matching using SGMatch. *Knowledge and Information Systems* 51, 1 (2017), 61–87.

[80] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding network motifs using MCMC sampling. In *Complex Networks VI*. Springer, 13–24.

[81] Ryoma Sato. 2020. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078* (2020).

[82] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. 2013. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM international conference on data mining*. SIAM, 10–18.

[83] Comandur Seshadhri and Srikanta Tirthapura. 2019. Scalable subgraph counting: the methods behind the madness. In *Companion Proceedings of The 2019 World Wide Web Conference*. 1317–1318.

[84] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment* 1, 1 (2008), 364–375.

[85] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*. PMLR, 488–495.

[86] Giorgio Stefanoni, Boris Motik, and Egor V Kostylev. 2018. Estimating the cardinality of conjunctive queries over RDF data using graph summarisation. In *Proceedings of the 2018 World Wide Web Conference*. 1043–1052.

[87] Ji Sun and Guoliang Li. 2019. An end-to-end learning-based cost estimator. *Proceedings of the VLDB Endowment* 13, 3 (2019), 307–319.

[88] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *Proceedings of the 2021 International Conference on Management of Data*. 1745–1757.

[89] Shixuan Sun and Qiong Luo. 2020. In-Memory Subgraph Matching: An In-depth Study. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1083–1098.

[90] Vladimir Vacic, Lilia M Iakoucheva, Stefano Lonardi, and Predrag Radivojac. 2010. Graphlet kernels for prediction of functional residues in protein structures. *Journal of Computational Biology* 17, 1 (2010), 55–72.

[91] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[92] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. 2015. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1530–1541.

[93] Clément Vignac, Andreas Loukas, and Pascal Frossard. 2020. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[94] Cédric Villani. 2009. *Optimal transport: old and new*. Vol. 338. Springer.

[95] Hanchen Wang, Defu Lian, Wanqi Liu, Dong Wen, Chen Chen, and Xiaoyang Wang. 2021. Powerful graph of graphs neural network for structured entity analysis. *World Wide Web* (2021), 1–21.

[96] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. 2021. Binarized graph neural network. *World Wide Web* 24, 3 (2021), 825–848.

[97] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Xuemin Lin. 2021. GoGNN: graph of graphs neural network for predicting structured entity interactions. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 1317–1323.

[98] Hanchen Wang, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang, and Xuemin Lin. 2022. Reinforcement Learning Based Query Vertex Ordering Model for Subgraph Matching. *arXiv preprint arXiv:2201.11251* (2022).

[99] Jianxin Wang, Yuannan Huang, Fang-Xiang Wu, and Yi Pan. 2012. Symmetry compression method for discovering network motifs. *IEEE/ACM transactions on computational biology and bioinformatics* 9, 6 (2012), 1776–1789.

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

[101] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2017. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2017), 73–86.

[102] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, and Yoshiharu Ishikawa. 2021. Consistent and flexible selectivity estimation for high-dimensional data. In *Proceedings of the 2021 International Conference on Management of Data*. 2319–2327.

[103] Peizhi Wu and Gao Cong. 2021. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation. In *Proceedings of the 2021 International Conference on Management of Data*. 2009–2022.

[104] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[105] Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. 2020. BayesCard: Revitilizing Bayesian Frameworks for Cardinality Estimation. *arXiv preprint arXiv:2012.14743* (2020).

[106] Hongteng Xu, Dixin Luo, and Lawrence Carin. 2019. Scalable Gromov-Wasserstein learning for graph partitioning and matching. *Advances in neural information processing systems* 32 (2019), 3052–3062.

[107] Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin Duke. 2019. Gromov-wasserstein learning for graph matching and node embedding. In *International conference on machine learning*. PMLR, 6932–6941.

[108] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[109] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *Proceedings of the VLDB Endowment* 14, 1 (2020), 61–73.

[110] Zhengyi Yang, Longbin Lai, Xuemin Lin, Kongzhang Hao, and Wenjie Zhang. 2021. Huge: An efficient and scalable subgraph enumeration system. In *Proceedings of the 2021 International Conference on Management of Data*. 2049–2062.

[111]]yang13deep Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. [n. d.]. Deep Unsupervised Cardinality Estimation. *Proceedings of the VLDB Endowment* 13, 3 ([n. d.]).

[112] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[113] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. 2019. ProNE: Fast and Scalable Network Representation Learning.. In *IJCAI*, Vol. 19. 4278–4284.

[114] Luming Zhang, Mingli Song, Zicheng Liu, Xiao Liu, Jiajun Bu, and Chun Chen. 2013. Probabilistic graphlet cut: Exploiting spatial structure cue for weakly supervised image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1908–1915.

[115] Luming Zhang, Mingli Song, Qi Zhao, Xiao Liu, Jiajun Bu, and Chun Chen. 2012. Probabilistic graphlet transfer for photo cropping. *IEEE Transactions on Image Processing* 22, 2 (2012), 802–815.

[116] Shijie Zhang, Shirong Li, and Jiong Yang. 2009. GADDI: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. 192–203.

[117] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. In *Proceedings of the 2021 International Conference on Management of Data*. 2142–2155.

[118] Peixiang Zhao and Jiawei Han. 2010. On graph query optimization in large networks. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 340–351.

[119] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data*. 1525–1539.

[120] Chen Zhengdao, Chen Lei, Villar Soledad, and Joan Bruna. 2020. Can Graph Neural Networks Count Substructures? *Advances in neural information processing systems* (2020).

[121] Dongxiao Zhu and Zhaohui S Qin. 2005. Structural comparison of metabolic networks in selected single cell organisms. *BMC bioinformatics* 6, 1 (2005), 1–12.