

DeSCo: Towards Generalizable and Scalable Deep Subgraph Counting

Tianyu Fu¹, Chiyue Wei¹, Yu Wang¹, Rex Ying²
¹Tsinghua University, ²Yale University

ABSTRACT

Subgraph counting is the problem of counting the occurrences of a given query graph in a large target graph. Large-scale subgraph counting is useful in various domains, such as motif counting for social network analysis and loop counting for money laundering detection on transaction networks. Recently, to address the exponential runtime complexity of scalable subgraph counting, neural methods are proposed. However, existing neural counting approaches fall short in three aspects. Firstly, the counts of the same query can vary from zero to millions on different target graphs, posing a much larger challenge than most graph regression tasks. Secondly, current scalable graph neural networks have limited expressive power and fail to efficiently distinguish graphs in count prediction. Furthermore, existing neural approaches cannot predict the occurrence position of queries in the target graph.

Here we design DeSCo, a scalable neural deep subgraph counting pipeline, which aims to accurately predict the query count and occurrence position on any target graph after one-time training. Firstly, DeSCo uses a novel *canonical partition* and divides the large target graph into small neighborhood graphs. The technique greatly reduces the count variation while guaranteeing no missing or double-counting. Secondly, *neighborhood counting* uses an expressive subgraph-based heterogeneous graph neural network to accurately perform counting in each neighborhood. Finally, *gossip propagation* propagates neighborhood counts with learnable gates to harness the inductive biases of motif counts. DeSCo is evaluated on eight real-world datasets from various domains. It outperforms state-of-the-art neural methods with 137× improvement in the mean squared error of count prediction, while maintaining the polynomial runtime complexity.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Graph-based database models**.

KEYWORDS

subgraph counting, graph mining, graph neural network

1 INTRODUCTION

Given a *query* graph and a *target* graph, the problem of subgraph counting is to count the number of *patterns*, defined as subgraphs of the target graph, that are graph-isomorphic to the query graph [64].

Subgraph counting is crucial for domains including biology [1, 6, 8, 70, 74], social science [41, 61, 76, 81], risk management [3, 63], and software analysis [78, 86]. For example, in brain networks, subgraph counting is used to identify important functional motifs and understand how the brain evolves [72]. In social networks,

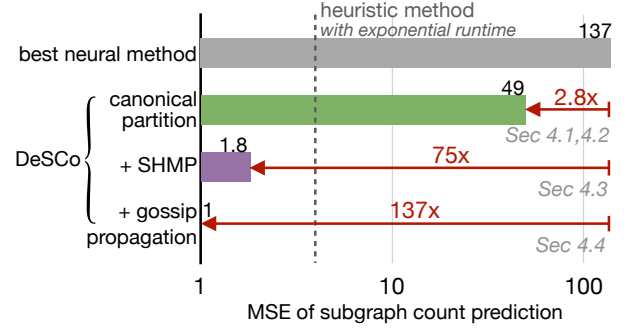


Figure 1: DeSCo Pipeline reduces the mean square error (MSE) of subgraph count prediction with three components: canonical partition, subgraph-based heterogeneous message passing (SHMP) and gossip propagation. The MSE is evaluated and averaged on eight real-world datasets.

counts of stars, holes, or paths are used to characterize circles of friends [18].

While being an essential method in graph and network analysis, subgraph counting is a #P-complete problem [77]. Due to the computational complexity, existing exact counting algorithms are restricted to small query graphs with no more than 5 vertices [2, 57, 60]. The commonly used VF2 [21] algorithm fails to even count a single query of a 5-node chain within a week’s time budget on a large target graph Astro [44] with nineteen thousand nodes.

Luckily, approximate counting of query graphs is sufficient in most real-world use cases [39, 42, 65]. Heuristic methods can scale to large targets by substructure sampling, random walk, and color-based sampling, allowing estimation of the frequency of query graph occurrences. However, they still cannot scale to large queries. Very recently, Graph Neural Networks (GNNs) are employed as a deep learning-based approach to scale the query graphs in subgraph counting [20, 46, 94]. The target graph and the query graph are embedded via a GNN, which predicts the motif count through a regression task.

However, there exist several major challenges with existing heuristic and GNN approaches: 1) The number of graph structures and count variation both grow super-exponentially with respect to the graph size [62, 69], resulting in large approximation error [64]. For different large target graphs, the counts of the same query can vary from zero to millions, making the task much harder than most graph regression tasks [71], which only predict a single-digit number with a small upper bound. 2) The expressive power of commonly used message passing GNNs is limited by the Weisfeiler-Lehman (WL) test [20, 43, 87]. Certain structures are not distinguishable with these GNNs, let alone counting them, resulting in the same count prediction for different queries. 3) Furthermore, most existing

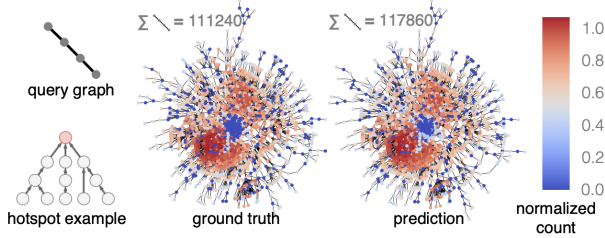


Figure 2: The total count and the position distribution of the query graph over the CiteSeer Citation Network. The figure compares between ground truth and DeSCo predictions. The hotspots are where the 4-chain patterns appear most often in CiteSeer.

approximate heuristic and GNN methods only focus on estimating the total count of a query in the target graph [15, 20, 46], but not the occurrence positions of the patterns, as shown in Figure 2. Yet such position distribution information is crucial in various applications [10, 26, 36, 75, 91].

Proposed work. To resolve the above challenges, we propose DeSCo, a GNN-based model that learns to predict both pattern counts and occurrence positions on any target graph. The main idea of DeSCo is to leverage the local information of neighborhood patterns to predict query counts and occurrences in the entire target graph. DeSCo first uses *canonical partition* to decompose the target graph into small neighborhoods. The local information is then encoded using a GNN with *subgraph-based heterogeneous message passing*. Finally, we perform *gossip propagation* to use inductive biases to improve counting accuracy over the entire graph. Our contributions are four-fold.

Canonical partition. Firstly, we propose *canonical partition* that divides the problem into subgraph counting for individual neighborhoods. We theoretically prove that no pattern will be double counted or missed for all neighborhoods. The algorithm allows the model to make accurate predictions on large target graphs with high count variation. Furthermore, we can predict the pattern position distribution for the first time, as shown in Figure 2. In this citation network, the hotspots represent overlapped linear citation chains, indicating original publications that motivate multiple future directions of incremental contributions [30, 90], which shed light on the research impact of works in this network.

Subgraph-based heterogeneous message passing. Secondly, we propose a general approach to enhance the expressive power of any MPNNs by encoding the subgraph structure through heterogeneous message passing. The message type is determined by whether the edge presents in a certain subgraph, e.g., a triangle. We show that this architecture outperforms expressive GNNs, including GIN [87] and ID-GNN [93], while maintaining the polynomial runtime complexity for scalable subgraph counting.

Gossip propagation. We further improve the count prediction accuracy by utilizing two inductive biases of the counting problem: homophily and antisymmetry. Real-world graphs share similar patterns among adjacent nodes, as shown in Figure 2. Furthermore, since canonical count depends on node indices, there exists antisymmetry due to canonical partition. Therefore, we propose a *gossip propagation* phase featuring a learnable gate for propagation to leverage the inductive biases.

Generalization Framework. We propose a generalization framework that uses the carefully designed synthetic dataset to enable model generalization to different real-world datasets. After training on the synthetic dataset, the model can directly perform subgraph counting inference with high accuracy on real-world datasets.

To demonstrate the effectiveness of DeSCo, we compare it against state-of-the-art GNN-based subgraph counting methods [20, 46, 47], as well as approximate heuristic method [15] on eight real-world datasets from various domains. DeSCo achieves 137 \times mean square error reduction of count predictions for both small and large targets, as shown in Figure 1. To the best of our knowledge, it is also the first approximate method to accurately predict pattern position distribution as illustrated in Figure 2. DeSCo also maintains polynomial runtime efficiency, demonstrating orders of magnitude speedup over the heuristic [15] and exact methods [21, 73].

2 RELATED WORKS

There has been an extensive line of work to solve the subgraph counting problem.

Exact counting algorithms. Exact methods generally count subgraphs by searching through all possible node combinations and finding the matching pattern. Early methods usually focus on improving the matching phase [21, 52, 84]. Recent approaches emphasize the importance of pruning the search space and avoiding double counting [23, 49, 50, 68]. However, exact methods still scale poorly in terms of query size (often no more than five nodes) despite much effort [19, 60].

Approximate heuristic methods. To further scale up the counting problem, approximate counting algorithms sample from the target graph to estimate pattern counts. Strategies like path sampling [40, 80], random walk [67, 89], substructure sampling [29, 39], and color coding [14, 16] are used to narrow the sample space and provides better error bound. However, large and rare queries are still hard to find in the vast sample space, leading to large approximation error [15].

GNN-based approaches. Recently, GNNs have been used to attempt counting large queries. [46, 92] use GNNs to embed the query and target graph, and predict subgraph counts via embeddings. [20] theoretically analyzes the expressive power of GNNs for counting and proposes an expressive GNN architecture. [94] proposes an active learning scheme for the problem. [47] proposes expensive edge-to-vertex dual graph transformation to enhance the model expressive power for subgraph counting. Unfortunately, large target graphs have extremely complex structures and a high variation of pattern count, so accurate prediction remains challenging.

3 PRELIMINARY

Let $G_t = (V_t, E_t)$ be a large *target* graph with vertices V_t and edges E_t . Let $G_q = (V_q, E_q)$ be the *query* graph of interest. The *subgraph counting problem* $C(G_q, G_t)$ is to calculate the size of the *set of patterns* $\mathcal{P} = \{G_p | G_p \subseteq G_t\}$ in the target graph G_t that are isomorphic to the query graph G_q , that is, \exists bijection $f: V_p \mapsto V_q$ such that $(f(v), f(u)) \in E_q$ iff $(v, u) \in E_p$, denoted as $G_p \cong G_q$.

Subgraph counting includes induced and non-induced counting depending on whether the pattern G_p is restricted to induced subgraph [64]. A $G_p = (V_p, E_p)$ is induced subgraph of G_t if $\forall e \in E_t \leftrightarrow$

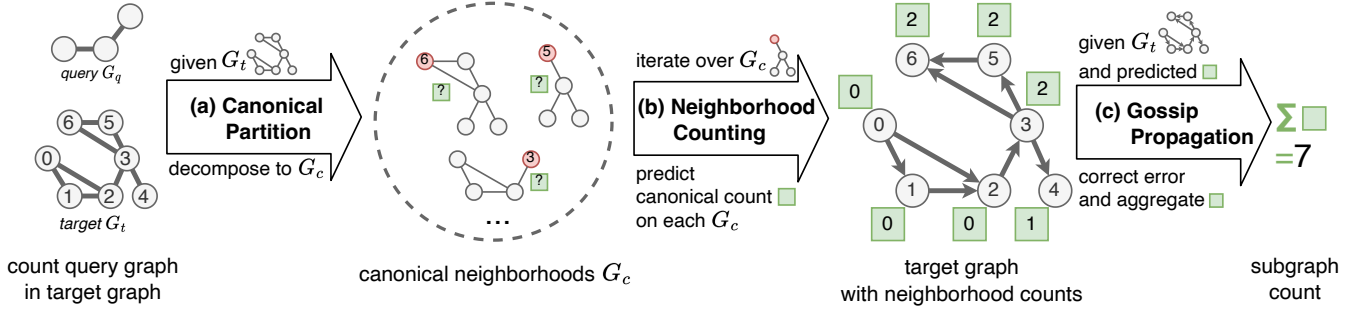


Figure 3: DeSCo Pipeline in 3 steps. (a) Step 1. Canonical Partition: Given *query* and *target*, decompose *target* into multiple node-induced subgraphs, i.e., *canonical neighborhoods*, based on node indices. Each neighborhood contains a *canonical node* that has the greatest index in the neighborhood. **(b) Step 2. Neighborhood Counting:** Predict the *canonical counts* of each neighborhood via an expressive GNN, and assign the count of the neighborhood to the corresponding *canonical node*. Neighborhood counting is the local count of queries. **(c) Step 3. Gossip Propagation:** Use GNN prediction results to estimate *canonical counts* on the *target* graph through learnable gates.

$e \in E_p$, denoted as $G_p \subseteq G_t$. Without loss of generality, we focus on the connected, induced subgraph counting problem, following modern mainstream graph processing frameworks [32, 59] and real-world applications [52, 85]. It is also possible to obtain non-induced occurrences from induced ones with a transformation [28]. Our GNN approach can natively support graphs with node features and edge directions. But in alignment with exact and heuristic methods, we use undirected graphs without node features in experiments to investigate the ability to capture graph topology.

4 DESCOT PIPELINE

In this section, we introduce the pipeline of DeSCo as shown in Figure 3. To perform subgraph counting, DeSCo first performs **canonical partition** to decompose the target graph to many canonical neighborhood graphs. Then, **neighborhood counting** uses the subgraph-based heterogeneous GNN to embed the query and neighborhood graphs and performs a regression task to predict the canonical count on each neighborhood. Finally, **gossip propagation** propagates neighborhood count predictions over the target graph with learnable gates to further improve counting accuracy. We will first introduce the model objective before elaborating on each step.

4.1 Canonical Count Objective

Motivation. For commonly seen node-level tasks such as node classification, each node is responsible for predicting its own node value. However, for subgraph counting, since each pattern contains multiple nodes, it is unclear which node should be responsible for

predicting the pattern’s occurrence. As illustrated in Figure 4, the ambiguity can lead to missing or double-counting of the motif, especially for queries with symmetric nodes, e.g. triangle. So we propose the canonical count objective to eliminate the ambiguity by assigning a specific canonical node responsible for each pattern. The canonical node is used to represent the pattern position. The canonical count is used as the local count prediction objective for the GNN and gossip propagation.

To break the symmetry, we randomly assign node indices on the target graph and define the *canonical node*.

DEFINITION 4.1 (CANONICAL NODE). Canonical node v_c is the node with the largest node index in the pattern.

$$v_c = \max_I V_p \quad (1)$$

Based on the index, we assign the count of the k -node pattern to its *canonical node* and define *canonical count*.

DEFINITION 4.2 (CANONICAL COUNT). Canonical count C_c equals the number of patterns that share the same canonical node.

$$C_c(G_q, G_t, v_c) = |\{G_p \subseteq G_t | G_p \cong G_q, v_c = \max_I V_p\}| \quad (2)$$

The canonical count $C_c(G_q, G_t, v_c)$ differs from the regular count C_c , as it takes an additional variable - a node v_c from the target graph. As shown in Figure 4(c), a pattern is only counted by its canonical node in C_c . So the summation of C_c over all nodes equals the count of all patterns, C , as stated in Lemma 4.1 and proven in Appendix A.1.

LEMMA 4.1. The subgraph count C of query in target equals the summation of the canonical count of query in target for all target nodes.

$$C(G_q, G_t) = \sum_{v_c \in V_t} C_c(G_q, G_t, v_c) \quad (3)$$

Advantage. By predicting the canonical count of each node, DeSCo can naturally get the pattern position distribution.

Lemma 4.1 allows the decomposition of the counting problem into multiple canonical count objectives. We use the following canonical partition to minimize the overhead for the decomposition.

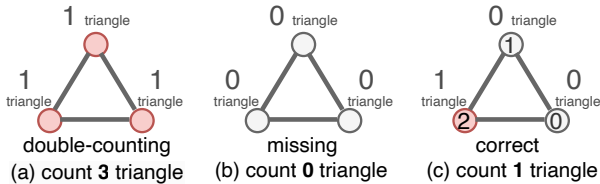


Figure 4: When counting, (a) double-counts and (b) misses the triangle in the neighborhoods due to symmetry. (c) DeSCo uses the canonical node to break symmetry and correctly count the triangle. ① are the node indices.

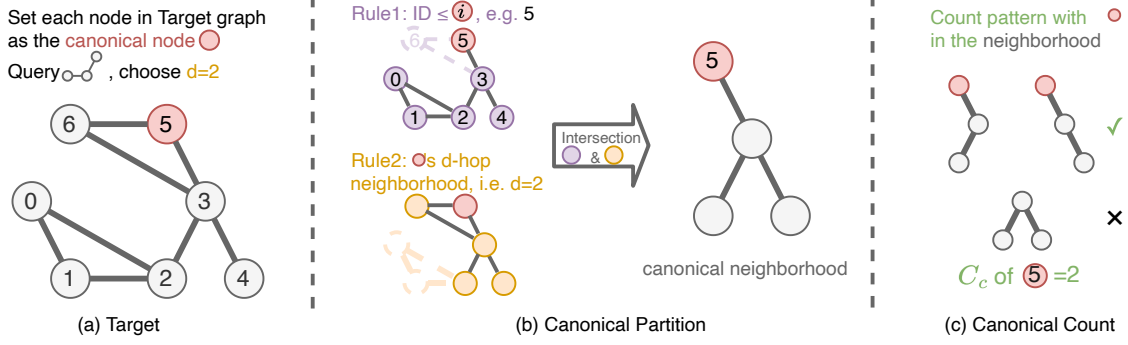


Figure 5: An example of canonical partition and canonical count. (a) Choose node 5 from the target graph as the *canonical node* (red circle). (b) *Canonical partition* generates the corresponding *canonical neighborhood* graph. It performs an ID-restricted breadth-first search to find the induced neighborhood that complies with both Rule 1 and Rule 2. (c) The corresponding *canonical count* is defined by the number of patterns containing the canonical node in the canonical neighborhood. DeSCo’s *neighborhood counting* phase predicts the canonical count for each canonical neighborhood.

4.2 Canonical Partition

Motivation. In Lemma 4.1, each canonical count C_c is obtained with the entire target graph G_t . In order to overcome the high computational complexity, we partition the target to reduce the graph size for the canonical count. We observe that each canonical count only depends on some local neighborhood structure as shown in Figure 5(c). So we propose *canonical partition* to efficiently get the small neighborhood.

Unique challenges of partition for canonical count. Commonly used graph partition strategies include cutting edges [5] and taking d-hop neighborhoods [33]. However, edge-cutting breaks the pattern structure, leading to incorrect count; D-hop neighborhoods guarantee correctness, yet are unnecessarily large since patterns exist in many overlapping neighborhoods.

Thus, we define *canonical partition*. It neglects the neighborhood structure that does not influence the canonical count of each node. Canonical partition uses node indices to filter nodes as illustrated in Figure 5(a), (b).

DEFINITION 4.3 (CANONICAL PARTITION). Canonical partition \mathcal{P} crops the index-restricted d-hop neighborhood around the center node from the target graph. $\mathcal{D}(G_t, v_i, v_c)$ means the shortest distance between v_i and v_c on G_t .

$$\begin{aligned} \mathcal{P}(G_t, v_c, d) &= G_c, \\ \text{s. t. } G_c &\subseteq G_t, V_c = \{v_i \in V_t \mid \mathcal{D}(G_t, v_i, v_c) \leq d, v_i \leq v_c\} \end{aligned} \quad (4)$$

The graph G_c obtained by canonical partition is called the *canonical neighborhood*. Canonical neighborhoods can correctly substitute the target graph in canonical count as proven in Appendix A.2. Thus, we derive Theorem 1.

THEOREM 1. The subgraph count of query in target equals the summation of the canonical count of query in canonical neighborhoods for all target nodes. Canonical neighborhoods are acquired with canonical partition \mathcal{P} , given any d greater than the diameter of the query.

$$\begin{aligned} C(G_q, G_t) &= \sum_{v_c \in V_t} C_c(G_q, \mathcal{P}(G_t, v_c, d), v_c), \\ d &\geq \max_{v_i, v_j \in V_q} \mathcal{D}(G_q, v_i, v_j) \end{aligned} \quad (5)$$

In DeSCo, given the target graph G_t , it iterates over all nodes v of the target G_t and divides it into a set of canonical neighborhoods G_{v_c} with *canonical partition*. In practice, we set d as the maximum diameter of query graphs to meet the requirements of Theorem 1. See Appendix A.3 for the implementation of $\mathcal{P}(G_t, v_c, d)$.

Advantage. Canonical partition dramatically reduces the worst and average complexity of the real-world subgraph counting problem by a factor of $1/10^{70}$ and $1/10^{11}$, as discussed in Appendix A.4. Furthermore, diverse target graphs can have similar and limited kinds of canonical neighborhoods. So it boosts the generalization power of DeSCo as shown in Section 5.4.

This divide-and-conquer scheme not only greatly reduces the complexity of each GNN prediction, but also makes it possible to predict the count distribution over the entire graph. After the canonical partition, DeSCo uses the following model to predict the canonical count for each decomposed neighborhood.

4.3 Neighborhood Counting

After canonical partition, GNNs are used to predict the *canonical count* $C_c(G_q, G_{v_c}, v_c)$ on any canonical neighborhood G_{v_c} in the *neighborhood counting* stage. The canonical neighborhood and the query are separately embedded using GNNs. The embeddings are passed to a multilayer perceptron to predict the canonical count.

Motivation. Previous work [20] shows message passing (MP) GNNs confuse certain graph structures and harm the counting accuracy. To enhance GNN’s expressive power while remaining scalable, we propose the Subgraph-based Heterogeneous Message Passing (SHMP) framework. Inspired by [53], SHMP incorporates subgraph information to boost the expressive power. In the meantime, SHMP avoids using super-node [53] or message permutation [20] that are computationally expensive during message passing.

Neighborhood counting with SHMP. To embed the input graph, SHMP uses small subgraph structures to categorize edges into different edge types, and uses different learnable weights for each edge type.

DEFINITION 4.4 (SUBGRAPH-BASED HETEROGENEOUS MESSAGE PASSING). The SHMP computes each node’s representation with equation 6. Here k denotes the layer; ϕ_h^k denotes the message function of the h -th edge type; $N_h(i)$ denotes nodes that connect to node i with

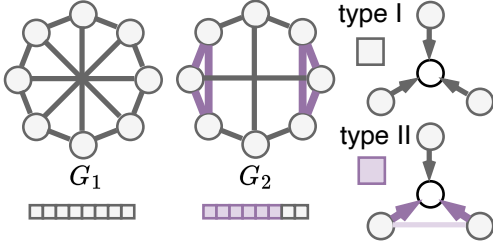


Figure 6: Proposed SHMP. Embedded with regular MP, graphs G_1 and G_2 are indistinguishable. While embedded with SHMP, G_2 is successfully distinguished with six type II node embeddings, demonstrating better expressive power of SHMP.

the h -th edge type; AGG and AGG' are the permutation invariant aggregation function such as sum, mean, or max.

$$\begin{aligned} \mathbf{x}_i^{(k)} &= \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, AGG'_{h \in H} (M_h) \right) \\ M_h &= AGG_{j \in N_h(i)} (\phi_h^{(k)} (\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i})) \end{aligned} \quad (6)$$

Note that MP defined by major GNN frameworks [27, 79] is just a special case of SHMP if only one edge type is derived with the subgraph structure. We prove that SHMP can exceed the upper bound of MP in terms of expressiveness in Appendix B.1.

For example, Figure 6 demonstrates that triangle-based heterogeneous message passing has better expressive power. Regular MPGNNs fail to distinguish different d -regular graphs G_1 and G_2 because of their identical type I messages and embeddings, which is a common problem of MPGNNs [93]. SHMP, however, can discriminate the two graphs by giving different embeddings. The edges are first categorized into two edge types based on whether they exist in any triangles (edges are colored purple if they exist in any triangles). Since no triangles exist in G_2 , all of its nodes still receive type I messages. While some nodes of G_1 now receive type II messages with two purple messages and one gray message in each layer. As a result, the model acquires not only the adjacency information between the message sender and receiver, but also information among their neighbors. Such subgraph structural information improves expressiveness by incorporating high-order information in both the query and the target. In DeSCO, the canonical node of the neighborhood is also treated as a special node type in the heterogeneous message passing.

Advantage. The triangle-based SHMP reduces the typical error of MPGNNs by 68% as discussed in Appendix B.2, while remaining polynomial runtime complexity of $O(V + E^{3/2})$ as discussed in Appendix F. The comparison with other expressive GNNs are shown in Table 5 and Appendix B.3.

The summation of the neighborhood counts (the predicted canonical counts of all canonical neighborhoods) can serve as the final subgraph count prediction. The counts also show the position of patterns. But to further improve counting accuracy, we pass the neighborhood counts to the *gossip propagation* stage.

4.4 Gossip Propagation

Given the count predictions \hat{C}_c output by the GNN, DeSCO uses **gossip propagation** to improve the prediction quality, enforcing different homophily and antisymmetry inductive biases for different queries. Gossip propagation uses another GNN to model the error

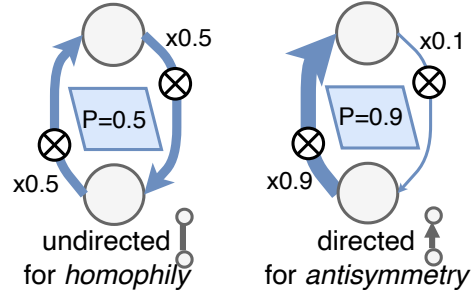


Figure 7: Proposed learnable gates in the gossip propagation model balance the influence of *homophily* and *antisymmetry* by controlling message directions.

of neighborhood count. It uses the predicted \hat{C}_c as input, and the canonical counts C_c as the supervision for corresponding nodes in the target graph.

Motivation. To further improve the counting accuracy, we identify two inductive biases: *Homophily* and *Antisymmetry*. 1) *Homophily*: Adjacent nodes within graphs share similar graph structures, resulting in analogous canonical counts (Figure 2). This phenomenon, termed *homophily* of canonical counts, stands out. 2) *Antisymmetry*: Nodes with similar neighborhood structures, per Definition 4.2, exhibit higher canonical counts for those with larger node indices. See right part of Figure 3 for an example. Details are in Appendix C.

We observe a negative correlation between *Antisymmetry* ratio and *Homophily* in different queries, as depicted in Figure 14 in Appendix C. This observation inspires us to learn this relationship within models.

The edges' direction in message passing can control the *homophily* and *antisymmetry* properties of the graph. With undirected edges, message propagation is a special low-pass filter [56], enhancing the homophily property of the node values. With directed edges pointing from small-index nodes to large-index nodes, message propagation accumulates value in large-index nodes, which enhances the antisymmetry property.

Gossip propagation with learnable gates. To learn the edge direction that correctly emphasizes homophily or antisymmetry, we propose the gossip propagation model as shown in Figure 7. It multiplies a learnable gate P for the message sent from the node with the smaller index, and $1 - P$ for the reversed one. P is learned from the query embedding. For different queries, P ranges from 0 to 1 to balance the influence of *homophily* and *antisymmetry*. When $P \rightarrow 0.5$, messages from the smaller indexed node and the reversed one are weighed equally. So it simulates undirected message passing that stress *homophily* by taking the average of adjacent node values. When the gate value moves away from 0.5, the message from one end of the edge is strengthened. For example, when $P \rightarrow 1$, the node values only accumulate from nodes with smaller indices to nodes with larger ones. So that it simulates directed message passing that stress *antisymmetry* of the transitive partial order of node indices.

The messages of MPGNNs are multiplied with g_{ji} on both edge directions. With learnable gates, the model can balance the effects of homophily and antisymmetry for further performance improvement.

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \text{AGG}_{j \in N(i)} g_{ji} \cdot \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right)$$

$$g_{ji} = \begin{cases} P & v_j \leq v_i \\ 1 - P & v_j > v_i \end{cases} \quad (7)$$

Final count prediction. The neighborhood count with gossip propagation is a more accurate estimation of the canonical count. The summation of the neighborhood counts is the unbiased estimation of subgraph count on the whole target graph as Theorem 1 states.

5 EXPERIMENTS

Dataset	#graphs	Avg. #nodes	Avg. #edges
SYNTHETIC	1827	134.91	381.58
MUTAG	188	17.93	19.79
COX2	467	41.22	43.45
ENZYMES	600	32.63	62.14
IMDB-BINARY	1000	19.77	96.53
MSRC-21	563	77.52	198.32
FIRSTMM-DB	41	1.3K	3.0K
CITESEER	1	3.3K	4.5K
CORA	1	2.7K	5.4K

Table 1: Graph statistics of datasets used in experiments.

We compare the performance of DeSCo with state-of-the-art neural subgraph counting methods, as well as the approximate heuristic method. Our evaluation showcases the scalability and generalization capabilities of DeSCo across diverse and larger target datasets, contrasting with prior neural methods that mostly focused on smaller datasets. We also demonstrate the runtime advantage of DeSCo compared to recent exact and approximate heuristic methods. Extensive ablation studies further show the benefit of each component of DeSCo.

5.1 Experimental Setup

Datasets. While previous neural approaches primarily targeted smaller datasets with limited scalability and generalization, we go beyond those limitations by evaluating on larger real-world datasets from various domains such as chemistry (MUTAG [22], COX2 [66]), biology (ENZYMES [13]), social networks (IMDB-BINARY [88]), computer vision (MSRC-21 [54], FIRSTMM-DB [54]), and citation networks (CiteSeer [31], Cora [51]). To ensure a comprehensive evaluation, we also construct a synthetic dataset using mixed graph generators, which covers diverse graph characteristics. See Table 1 for the statistics of the datasets. For detailed information on target dataset and query characteristics, please refer to the Appendix D. **Generalization framework.** To enable subgraph counting on diverse target graphs, we initially train all neural methods with target-query pairs from the Synthetic dataset and standard queries of size 3 – 5. Neural methods are trained and assessed using the subgraph count objective. Notably, DeSCo only requires a single training phase across various datasets and tasks.

Baselines. We compare DeSCo with state-of-the-art subgraph counting GNNs, including LRP [20], DIAMNet [46] and DMPNN [47]. We ensure a fair comparison by adopting optimal configurations for both the baselines and DeSCo. For the approximate heuristic counting method, we choose the state-of-the-art MOTIVO [15], which employs color-based sampling with efficient c++ implementation. For exact counting methods, we consider VF2 [21] and IMSM [73]. Refer to Appendix D.4 and F for the specific baseline configurations. **Evaluation metric.** Our evaluation employs mean square error (MSE) and mean absolute error (MAE) of total subgraph count prediction. MSE is normalized by dividing the variance of the ground truth counts. These metrics allow for a comprehensive assessment of DeSCo’s performance across different query sizes and datasets.

5.2 Neural Counting

Subgraph counting. Table 2 summarizes the normalized MSE and MAE for predicting the subgraph count of twenty-nine standard query graphs on diverse target graph datasets. Utilizing canonical partition, neighborhood counting, and gossip propagation, DeSCo demonstrates 49.7× and 8.4× improvements of normalized mse and mae over the best neural baseline on average; 17.5× and 4.1× improvements of normalized mse and mae over the approximate heuristic method on average. These results underscore the reliability of DeSCo in real-world scenarios, even for smaller queries of size 3 – 5. Further analysis of the relative count error using the q-error metric is provided in Appendix G.1. Note that IMDB-BINARY contains graphs with high density, thus challenging for all neural methods. Despite lower accuracy than heuristic methods, DeSCo still achieves robust and significant improvements over neural methods, while maintaining the linear runtime efficiency.

Position distribution. DeSCo introduce the accurate pattern position prediction for the first time. The pattern position prediction realizes 3.8×10^{-3} normalized MSE as discussed in Appendix E.2.

5.3 Scalability

Setup. Obtaining ground truth for large queries and targets via exact counting is extremely expensive and can take months, so we only test scalable queries and targets with the following setting in Section 5.3. This allows us to demonstrate that with minimal pre-training, the model efficiently scales up and provides reliable predictions for larger queries and targets.

Large queries. We select two frequently appearing queries for each query size between 6 to 13 from ENZYMES. See Appendix D.2 for more details. All the models are pre-trained with standard queries. Except for DeSCo (zero-shot), all models are fine-tuned with larger queries on the synthetic dataset. DeSCo (zero-shot) showcases the generalization power of DeSCo for unseen queries. Figure 8 shows the distributions of the square error of each query-target pair, normalized with the variance of all ground truth counts. Numeric results can be found in Appendix G.2.

Large target. We also evaluate the models on large target graphs as shown in Table 3. The maximum ground truth count of standard queries reaches up to 3.8×10^6 and 3.3×10^7 on CiteSeer and Cora, respectively, making it a challenging task. DeSCo outperforms other neural methods as shown in Table 3. Notably, prediction result of LRP is infinite and thus not included in the table.

Dataset	MUTAG			COX2			ENZYMES			IMDB-BINARY			MSRC-21		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE															
MOTIVO	2.9E-1	6.7E-1	1.2E+0	1.6E-1	3.4E-1	5.9E-1	1.6E-1	1.9E-1	3.0E-1	2.7E-2	3.9E-2	5.0E-2	4.8E-2	7.2E-2	9.5E-2
LRP	1.5E-1	2.7E-1	3.5E-1	1.4E-1	2.9E-1	1.1E-1	8.5E-1	5.4E-1	6.2E-1	inf	inf	inf	2.4E+0	1.4E+0	1.1E+0
DIAMNet	4.1E-1	5.6E-1	4.7E-1	1.1E+0	7.8E-1	7.2E-1	1.4E+0	1.1E+0	1.0E+0	1.1E+0	1.0E+0	1.0E+0	2.7E+0	1.6E+0	1.3E+0
DMPNN	6.1E+2	6.6E+2	3.0E+2	2.6E+3	2.4E+3	3.0E+3	2.9E+3	1.4E+3	1.2E+3	2.1E+4	1.3E+2	1.4E+2	1.1E+4	1.3E+3	4.1E+2
DeSCo	2.2E-3	7.5E-4	6.0E-3	6.6E-4	6.3E-4	4.9E-3	5.4E-3	5.9E-2	5.3E-2	8.5E-3	2.1E-1	4.5E-1	2.5E-3	3.8E-3	8.7E-2
MAE															
MOTIVO	4.9E+0	5.1E+0	3.3E+0	8.3E+0	9.4E+0	7.3E+0	1.7E+1	2.3E+1	2.6E+1	4.7E+1	1.6E+2	6.1E+2	4.1E+1	9.5E+1	1.7E+2
LRP	3.8E+0	5.1E+0	4.5E+0	9.5E+0	4.0E+0	6.3E+0	4.3E+1	4.0E+1	3.7E+1	inf	inf	inf	3.2E+2	4.6E+2	5.9E+2
DIAMNet	8.3E+0	7.9E+0	4.2E+0	3.0E+1	1.7E+1	1.2E+1	5.4E+1	5.1E+1	4.0E+1	2.9E+2	8.3E+2	2.6E+3	3.4E+2	4.9E+2	6.3E+2
DMPNN	6.8E+2	6.9E+2	2.4E+2	3.6E+3	4.3E+3	3.8E+3	4.8E+3	5.8E+3	6.0E+3	1.7E+5	2.2E+5	2.8E+5	3.4E+4	4.6E+4	5.7E+4
DeSCo	5.0E-1	1.8E-1	2.9E-1	6.1E-1	4.4E-1	7.7E-1	3.6E+0	1.1E+1	9.9E+0	2.4E+1	3.0E+2	1.6E+3	1.0E+1	2.5E+1	1.3E+2

Table 2: Normalized MSE and MAE performance of approximate heuristic and neural methods on subgraph counting of twenty-nine standard queries.

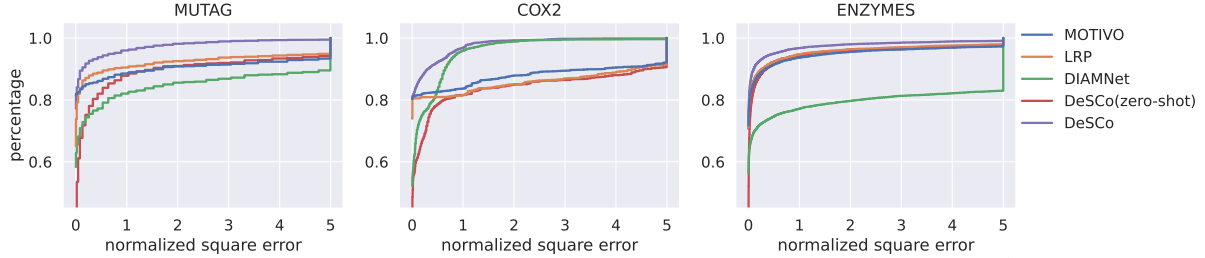


Figure 8: The accumulative distributions of normalized square error of large queries (size up to 13) on three target datasets. The x-axis is clipped at 5. Given any square error tolerance bound (x-axis), DeSCo has the highest percentage of predictions that meet the bound (y-axis). DeSCo(zero-shot) generalizes to unseen queries with competitive performance over specifically trained baselines.

Dataset	CiteSeer			Cora		
Query-Size	3	4	5	3	4	5
normalized MSE						
DIAMNet	2.0E+0	1.5E+0	1.2E+0	1.0E+10	3.2E+7	3.7E+4
DMPNN	9.5E+4	2.5E+2	6.8E+1	1.8E+5	1.1E+2	6.7E+1
DeSCo	3.5E-5	9.7E-2	1.6E-1	4.2E-3	2.1E-1	6.3E-2
MAE						
DIAMNet	1.1E+4	6.0E+4	3.6E+05	2.1E+9	1.6E+9	8.3E+8
DMPNN	6.1E+6	7.6E+6	8.7E+6	1.8E+7	2.4E+7	3.0E+7
DeSCo	6.0E+1	1.2E+4	1.1E+5	1.3E+3	7.3E+4	5.4E+5

Table 3: Normalized MSE and MAE performance of neural methods on large targets with standard queries.

5.4 Generalization Ability

Synthetic Dataset. We leverage the Synthetic dataset to demonstrate DeSCo’s strong generalization ability. We consider a vector of graph statistics, including the number of nodes, number of edges, average degree, clustering coefficient, shortest path length, diameter, and density. As shown in Figure 9 (a), real-world graphs have diverse statistics, especially those from different domains.

Fortunately, despite the wide variation in real-world graphs, they often exhibit similarities in terms of small local substructures. This observation is depicted in Figure 9 (b), where we observe

Test-Set	MUTAG			MSRC-21			FIRSTMM-DB		
Query-Size	3	4	5	3	4	5	3	4	5
Existing	6.5E-3	3.4E-3	8.7E-2	1.1E+1	1.9E+0	1.1E+0	1.1E-1	1.1E-1	1.6E-1
Synthetic	2.3E-3	8.4E-4	6.5E-3	2.5E-3	3.8E-3	8.7E-2	2.1E-3	3.6E-2	5.4E-2

Table 4: Normalized MSE performance with different training datasets. When pre-training on existing datasets, MSRC-21 uses MUTAG; CiteSeer uses Cora; FIRSTMM-DB uses CiteSeer.

similar characteristics in the canonical neighborhoods of graphs from different datasets. Furthermore, Figure 9 (c) demonstrates that our synthetic dataset, consisting of only 1827 graphs, successfully covers most real-world graphs. This finding reinforces the fact that DeSCo can be effectively trained on the synthetic dataset and exhibits strong generalization capabilities across a diverse set of real-world graphs.

Generalization. To evaluate the generalization performance, we employ the proposed framework and pre-train DeSCo on the Synthetic dataset, followed by direct testing on various real-world graph datasets. For comparison, we adhere to the regular scheme of training DeSCo on existing datasets. Table 4 presents the results, clearly indicating that DeSCo, pre-trained on the synthetic dataset, achieves remarkable accuracy and generalizability across different test datasets. In contrast, pre-training on existing datasets only yields limited generalization ability within the same domain.

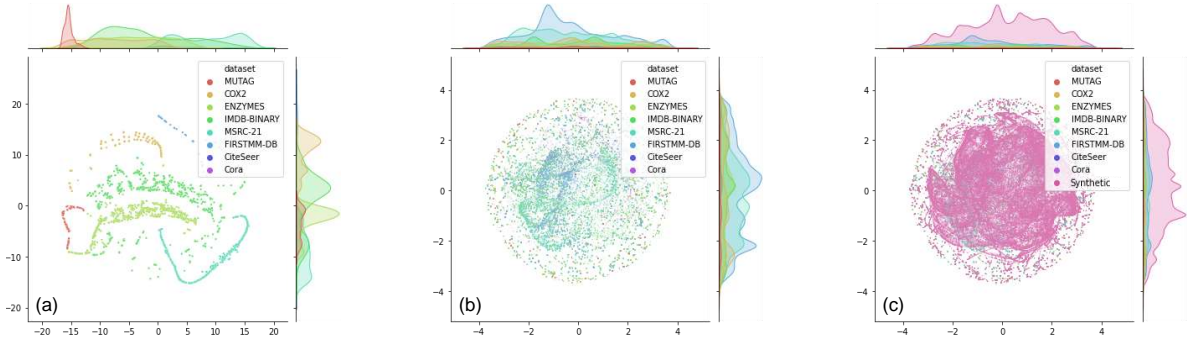


Figure 9: Visualization of statistics of diverse graph datasets. The embedding is obtained by projecting the vectors of graph statistics via t-SNE. (a) Each point represents a graph. (b) Each point represents a canonical neighborhood. (c) Canonical neighborhoods of the synthetic dataset cover most canonical neighborhoods of real-world graphs in terms of data distribution.

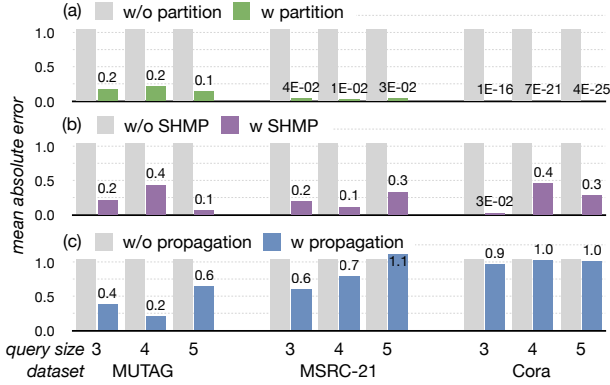


Figure 10: MAE performance with and without canonical partition, SHMP and gossip propagation.

5.5 Ablation Study

We explore the effectiveness of each component of DeSCo through the ablation study by removing each component. Figure 10 shows the MAE results on three datasets. The geometric mean of normalized MSE on eight real world datasets are shown in Figure 1. The numeric results are shown in Appendix E.

Ablation of canonical partition. We remove the canonical partition of DeSCo and instead train it with the objective of subgraph count on the entire target, mirroring the approach of other neural baselines. The divide-and-conquer approach of canonical partition substantially reduces errors, allowing DeSCo to surpass costly transformations used by other SOTA neural baselines (Figure 1).

Ablation of subgraph-based heterogeneous message passing. We use our proposed SHMP to improve the performance of GraphSAGE by transforming its standard message passing to heterogeneous message passing. We use triangle as the subgraph to categorize heterogeneous edges as shown in Figure 6. Figure 10(b) shows the accuracy improvement brought by SHMP to GraphSAGE. Table 5 shows the superiority over expressive GNNs, including GIN and ID-GNN.

Ablation of gossip propagation. The normalized MSE of direct summation of neighborhood counts and the summation after gossip propagation are compared to show the effectiveness of gossip propagation. With gossip propagation, the normalized MSE and MAE further reduces 1.8 \times and 1.4 \times , respectively.

Dataset	MUTAG			MSRC-21			Cora		
Query-Size	3	4	5	3	4	5	3	4	5
GCN	inf	inf	inf	inf	inf	inf	inf	inf	inf
SAGE	4.0E-1	9.6E-2	5.6E-1	1.9E-1	6.3E-1	3.4E-1	9.4E+0	5.9E-1	1.1E+0
GIN	8.4E-2	7.3E-2	1.6E-1	2.4E+0	1.6E+0	1.3E+0	1.9E+0	1.3E+0	1.1E+0
ID-GNN	3.3E-2	2.9E-2	1.5E-2	3.0E+0	1.7E+0	1.3E+0	2.1E+0	1.3E+0	1.1E+0
SHMP-SAGE	1.8E-2	1.6E-2	1.7E-2	7.5E-3	6.9E-3	6.7E-2	5.3E-3	2.2E-1	7.0E-2

Table 5: Normalized MSE performance with different GNN models for neighborhood counting.

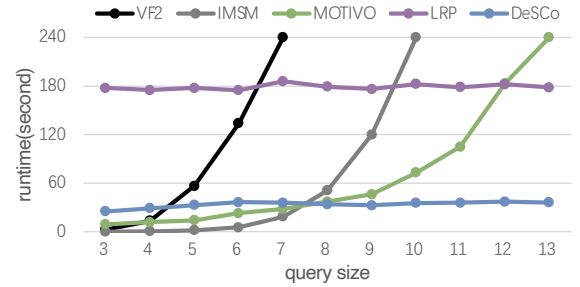


Figure 11: The runtime comparison between exact, heuristic approximate, neural methods and DeSCo. All tested on the ENZYMES dataset.

5.6 Runtime Comparison

Figure 11 shows the runtime of each method with four minutes' time-bound. For the exact methods, VF2 and IMSM, the runtime grows exponentially because of the #P hard nature of the subgraph counting problem. For the approximate heuristic method MOTIVO, the exponential growth is primarily attributed to the coloring phase before sampling. In contrast, the neural methods, LRP and DeSCo, demonstrate polynomial scalability. Unlike LRP, DeSCo does not need the heavy permutation of node features, so it further achieves 5.3 \times speedup. More runtime analysis is discussed in Appendix F.

6 CONCLUSION

We propose DeSCo, a neural network based pipeline for generalizable and scalable subgraph counting. With canonical partition, subgraph-based heterogeneous message passing, and gossip propagation, DeSCo accurately predicts counts for both large queries and targets. It demonstrates magnitudes of improvements in mean square error and runtime efficiency. It additionally provides the important position distribution of patterns that previous works cannot.

REFERENCES

- [1] Balázs Adamcsek, Gergely Palla, Illés J. Farkas, Imre Derényi, and Tamás Vicsek. 2006. CFinder: locating cliques and overlapping modules in biological networks. *Bioinformatics* 22, 8 (2006), 1021–1023.
- [2] 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.
- [3] Leman Akoglu and Christos Faloutsos. 2013. Anomaly, event, and fraud detection in large network datasets. In *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 773–774.
- [4] Réka Albert and Albert-László Barabási. 2000. Topology of evolving networks: local events and universality. *Physical review letters* 85, 24 (2000), 5234.
- [5] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. 2013. *Graph partitioning and graph clustering*. Vol. 588. American Mathematical Society Providence, RI.
- [6] Gary D. Bader and Christopher W. V. Hogue. 2003. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics* 4, 1 (2003), 2–2.
- [7] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.
- [8] Jordi Bascompte and Carlos J. Melián. 2005. Simple trophic modules for complex food webs. *Ecology* 86, 11 (2005), 2868–2873.
- [9] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 1–4.
- [10] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.
- [11] Bibek Bhattacharai, Hang Liu, and H Howie Huang. 2019. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*. 1447–1462.
- [12] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics* 14, 7 (2013), 1–13.
- [13] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.
- [14] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif counting beyond five nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 4 (2018), 1–25.
- [15] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1651–1663.
- [16] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2021. Faster motif counting via succinct color coding and adaptive sampling. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 6 (2021), 1–27.
- [17] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. 2013. House of Graphs: a database of interesting graphs. *Discrete Applied Mathematics* 161, 1–2 (2013), 311–314.
- [18] Raphaël Charbey and Christophe Prieur. 2019. Stars, holes, or paths across your Facebook friends: A graphlet-based characterization of many networks. *Network Science* 7, 4 (2019), 476–497.
- [19] Jingji Chen and Xuehai Qian. 2020. Dwarvesgraph: A high-performance graph mining system with pattern decomposition. *arXiv preprint arXiv:2008.09682* (2020).
- [20] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can graph neural networks count substructures? *ArXiv abs/2002.04025* (2020).
- [21] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26, 10 (2004), 1367–1372.
- [22] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34, 2 (1991), 786–797.
- [23] Sofie Demeyer, Tom Michael, Jan Fostier, Pieter Audenaert, Mario Pickavet, and Piet Demeester. 2013. The index-based subgraph matching algorithm (ISMA): fast subgraph enumeration in large networks using optimized search trees. *PloS one* 8, 4 (2013), e61183.
- [24] Evan Donato, Ming Ouyang, and Cristian Peguero-Isalguez. 2018. Triangle Counting with A Multi-Core Computer. *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018), 1–7.
- [25] Paul Erdős, Alfréd Rényi, et al. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960), 17–60.
- [26] Katherine Faust. 2010. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks* 32, 3 (2010), 221–233.
- [27] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.
- [28] Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. 2015. Induced subgraph isomorphism: Are some patterns substantially easier than others? *Theoretical Computer Science* 605 (2015), 119–128.
- [29] Tianyu Fu, Ziqian Wan, Guohao Dai, Yu Wang, and Huazhong Yang. 2020. LessMine: Reducing Sample Space and Data Access for Dense Pattern Mining. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.
- [30] Chao Gao and John Lafferty. 2017. Testing for global network structure using small subgraph statistics. *arXiv preprint arXiv:1710.00862* (2017).
- [31] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*. 89–98.
- [32] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
- [33] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
- [34] 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*. 1429–1446.
- [35] 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.
- [36] Paul W Holland and Samuel Leinhardt. 1976. Local structure in social networks. *Sociological methodology* 7 (1976), 1–45.
- [37] Petter Holme and Beom Jun Kim. 2002. Growing scale-free networks with tunable clustering. *Physical review E* 65, 2 (2002), 026107.
- [38] Alon Itai and Michael Rodeh. 1977. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*. 1–10.
- [39] Anand Padmanabha Iyer, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, Vladimir Braverman, and Ion Stoica. 2018. {ASAP}: Fast, approximate graph pattern mining at scale. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 745–761.
- [40] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th international conference on world wide web*. 495–505.
- [41] Yuval Kalish and Garry Robins. 2006. Psychological predispositions and network structure: The relationship between individual predispositions, structural holes and network closure. *Social networks* 28, 1 (2006), 56–84.
- [42] 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.
- [43] AA Leman and Boris Weisfeiler. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tekhnicheskaya Informatsiya* 2, 9 (1968), 12–16.
- [44] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densityification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.
- [45] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiao-Li Li. 2016. Network motif discovery: A GPU approach. *IEEE transactions on knowledge and data engineering* 29, 3 (2016), 513–528.
- [46] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, and Xin Jiang. 2020. Neural Subgraph Isomorphism Counting. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020).
- [47] Xin Liu and Yangqiu Song. 2022. Graph convolutional networks with dual message passing for subgraph isomorphism counting and matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 7594–7602.
- [48] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. *Advances in neural information processing systems* 32 (2019).
- [49] Daniel Mawhirter, Sam Reinehr, Connor Holmes, Tongping Liu, and Bo Wu. 2019. Graphzero: Breaking symmetry for efficient graph mining. *arXiv preprint arXiv:1911.12877* (2019).
- [50] Daniel Mawhirter and Bo Wu. 2019. Automine: harmonizing high-level abstraction and high performance for graph mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 509–523.
- [51] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
- [52] 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.
- [53] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *ArXiv abs/1810.02244* (2019).
- [54] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: efficient graph kernels from propagated information.

- Machine Learning* 102 (2016), 209–245.
- [55] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. 2020. k-hop Graph Neural Networks. *Neural networks : the official journal of the International Neural Network Society* 130 (2020), 195–205.
- [56] Hoang Nt and Takanori Maehara. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550* (2019).
- [57] Mark Ortmann and Ulrik Brandes. 2017. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science* 2, 1 (2017), 1–17.
- [58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
- [59] Tiago P. Peixoto. 2014. The graph-tool python library. *figshare* (2014). <https://doi.org/10.6084/m9.figshare.1164194>
- [60] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*. 1431–1440.
- [61] Christina Prell and John Skvoretz. 2008. Looking at social capital through triad structures. *Connections* 28, 2 (2008), 4–16.
- [62] Ronald C Read and Robin J Wilson. 1998. *An atlas of graphs*. Vol. 21. Clarendon Press Oxford.
- [63] Bernardete Ribeiro, Ning Chen, and Alexander Kovacec. 2017. Shaping graph pattern mining for financial risk. *Neurocomputing* (2017).
- [64] 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.
- [65] Pedro Ribeiro and Fernando Silva. 2010. Efficient subgraph frequency estimation with g-tries. In *International Workshop on Algorithms in Bioinformatics*. Springer, 238–249.
- [66] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. <https://networkrepository.com>
- [67] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding network motifs using MCMC sampling. In *Complex Networks VI*. Springer, 13–24.
- [68] Tianhui Shi, Mingshu Zhai, Yi Xu, and Jidong Zhai. 2020. Graphpi: High performance graph pattern matching through effective redundancy elimination. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–14.
- [69] Neil James Alexander Sloane. 2014. *A handbook of integer sequences*. Academic Press.
- [70] Ricard V Solé and Sergi Valverde. 2008. Spontaneous emergence of modularity in cellular networks. *Journal of The Royal Society Interface* 5, 18 (2008), 129–133.
- [71] Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. 2019. AqSolDB, a curated reference set of aqueous solubility and 2D descriptors for a diverse set of compounds. *Scientific data* 6, 1 (2019), 143.
- [72] Olaf Sporns, Rolf Kötter, and Karl J Friston. 2004. Motifs in brain networks. *PLoS biology* 2, 11 (2004), e369.
- [73] Shixuan Sun and Qiong Luo. 2020. In-memory subgraph matching: An in-depth study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1083–1098.
- [74] Ichigaku Takigawa and Hiroshi Mamitsuka. 2013. Graph mining: procedure, application to drug discovery and recent advances. *Drug discovery today* 18, 1-2 (2013), 50–57.
- [75] Charalampos E Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. 2017. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*. 1451–1460.
- [76] Shahadat Uddin, Liaquat Hossain, et al. 2013. Dyad and triad census analysis of crisis communication network. *Social Networking* 2, 01 (2013), 32.
- [77] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* (1979).
- [78] Sergi Valverde and Ricard V Solé. 2005. Network motifs in computational graphs: A case study in software architecture. *Physical Review E* 72, 2 (2005), 026107.
- [79] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).
- [80] 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.
- [81] Stanley Wasserman, Katherine Faust, et al. 1994. *Social network analysis: Methods and applications*. (1994).
- [82] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *nature* 393, 6684 (1998), 440–442.
- [83] Melanie Weber. 2019. Curvature and Representation Learning: Identifying Embedding Spaces for Relational Data.
- [84] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.
- [85] Elisabeth Wong, Brittany Baur, Saad Quader, and Chun-Hsi Huang. 2012. Biological network motif detection: principles and practice. *Briefings in bioinformatics* 13, 2 (2012), 202–215.
- [86] Peng Wu, Junfeng Wang, and Bin Tian. 2018. Software homology detection with software motifs based on function-call graph. *IEEE Access* 6 (2018), 19007–19017.
- [87] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
- [88] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1365–1374.
- [89] Chen Yang, Min Lyu, Yongkun Li, Qianqian Zhao, and Yinlong Xu. 2018. SSRW: a scalable algorithm for estimating graphlet statistics based on random walk. In *International Conference on Database Systems for Advanced Applications*. Springer, 272–288.
- [90] Guan-Can Yang, Gang Li, Chun-Ya Li, Yun-Hua Zhao, Jing Zhang, Tong Liu, Dar-Zen Chen, and Mu-Hsuan Huang. 2015. Using the comprehensive patent citation network (CPC) to evaluate patent value. *Scientometrics* 105, 3 (2015), 1319–1346.
- [91] Hao Yin, Austin R Benson, and Jure Leskovec. 2019. The local closure coefficient: A new perspective on network clustering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 303–311.
- [92] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/e77dbaf6759253c7c6d0efc5690369c7-Paper.pdf
- [93] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. *arXiv preprint arXiv:2101.10320* (2021).
- [94] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. *Proceedings of the 2021 International Conference on Management of Data* (2021).
- [95] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedom Lipka, Nesreen K Ahmed, and Danaï Koutra. 2021. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11168–11176.

ETHICAL CONSIDERATIONS

In the realm of graph analysis, DeSCo stands as a fundamental tool rather than a specific application-driven solution. While the direct potential for DeSCo to induce negative societal impacts is minimal, it remains prudent to acknowledge and address potential adverse outcomes.

Accuracy. Similar to other non-exact counting methods, DeSCo cannot ensure absolute prediction correctness. Despite thorough testing on extensive real-world datasets, which has showcased significant error reductions and exceptional generalization capabilities, the potential for inaccurate predictions, especially for outlier graphs, remains a possibility. Therefore, it’s advisable to exercise caution and validate basic graph statistics, such as maximum degree, before applying the DeSCo method.

Privacy. DeSCo introduces a breakthrough in accurately counting large subgraphs, previously unattainable. Moreover, it reveals the positional distribution of these counts. As subgraph counting finds applications in recommendation systems, social network analysis, and other domains, there’s potential for corporations and governments to glean intelligence that was once inaccessible. This advancement could inadvertently compromise user privacy if not subjected to proper oversight. To mitigate this, it’s essential to consider enforcing relevant regulations should corresponding technologies be developed.

A CANONICAL PARTITION

A.1 Proof of Lemma 4.1

PROOF. Following the notions from Section 3, given a query graph G_q and a target graph G_t , the node-induced count is defined as the number of G_t 's node-induced subgraph, pattern, G_p that is isomorphic to G_q . We denote the set of all G_p as \mathbb{M} .

$$\mathbb{M} = \{G_p \subseteq G_t | G_p \cong G_q\} \quad (8)$$

$$C(G_q, G_t) = |\mathbb{M}| \quad (9)$$

Assuming that G_q has k nodes. Then, under the node-induced definition, given G_t , we can use the k -node set $V_p = \{v | v \in G_p\}$ of G_p to represent the pattern.

We can decompose the set of all patterns \mathbb{M} into many subsets \mathbb{M}_c based on the maximum node index of each $G_p \in \mathbb{M}$.

$$\mathbb{M}_c = \{G_p \subseteq G_t | G_p \cong G_q, \max V_p = c\} \quad (10)$$

This maximum-index decomposition is exclusive and complete: every G_p has a single corresponding maximum node index. So we have the following properties:

$$\forall c \neq j, \mathbb{M}_c \cap \mathbb{M}_j = \emptyset \quad (11)$$

$$\mathbb{M} = \bigcup_{c=0}^{|V|-1} \mathbb{M}_c \quad (12)$$

Thus, the node-induced count in Equation 9 can be rewritten using the inclusion-exclusion principle.

$$\begin{aligned} C(G_q, G_t) &= \left| \bigcup_{c=0}^{|V|-1} \mathbb{M}_c \right| \\ &= \sum_{c=0}^{|V|-1} |\mathbb{M}_c| + \sum_{k=1}^{|V|-1} (-1)^k \left(\sum_{0 \leq i_0 \leq \dots \leq i_k < |V|} |\mathbb{M}_{i_0} \cap \dots \cap \mathbb{M}_{i_k}| \right) \\ &= \sum_{c=0}^{|V|-1} |\mathbb{M}_c| \end{aligned} \quad (13)$$

According to the definition of canonical count in Equation 2, $C_c(G_q, G_t, v_c) = |\mathbb{M}_c|$. Thus, Lemma 4.1 is proven with Equation 13. \square

A.2 Proof of Theorem 1

PROOF. By the definition of \mathbb{M}_c in Equation 10, we have a corollary.

COROLLARY 1.1. Denote v_c 's index as c , \mathcal{D} as the shortest path length between two nodes. Any graph in \mathbb{M}_c has node v_c and has the same graph-level property with G_q , e.g., diameter.

$$\forall G_p \in \mathbb{M}_c, v_c \in V_p, \max_{v_i, v_j \in V_p} \mathcal{D}(G_p, v_i, v_j) = \max_{v_i, v_j \in V_q} \mathcal{D}(G_q, v_i, v_j) \quad (14)$$

The distance between v_c and any nodes of G_p in \mathbb{M}_c is bounded by $\max_{v_i, v_j \in V_q} \mathcal{D}(G_q, v_i, v_j)$ as shown in corollary 1.1. So we can further know that graphs in \mathbb{M}_c are node-induced subgraphs of v_c 's d -hop ego-graph.

$$\begin{aligned} \forall G_p \in \mathbb{M}_c, \exists G_{d-ego} \subseteq G_t, V_{d-ego} &= \{v_i \in V_t | \mathcal{D}(G_t, v_i, v_c) \leq d\} \\ \text{s. t. } G_p &\cong G_{d-ego} \end{aligned} \quad (15)$$

Given Equation 10, it is also clear that all graphs in \mathbb{M}_c have smaller node indices than c .

$$\forall G_p \in \mathbb{M}_c, \exists G_{small} \subseteq G_t, V_{small} = \{v_i \in V_t | I_i \leq I_c\} \text{ s. t. } G_p \cong G_{small} \quad (16)$$

With Equation 15 and 16, we know that all the graphs in \mathbb{M}_c are subgraphs of $\mathcal{P}(G_t, v_c, d)$ defined in Equation 4. Thus, with respect to Equation 8, we can redefine \mathbb{M}_c as follows.

$$\mathbb{M}_c = \{G_p \subseteq \mathcal{P}(G_t, v_c, d) | G_p \cong G_q, \max V_p = c\} \quad (17)$$

Combining Equation 13 with Equation 17, Theorem 1 is proven. \square

A.3 Implementation of Canonical Partition

Algorithm 1 Index-restricted breadth-first search

```

 $V_c \leftarrow \{v_c\}, V_{front} \leftarrow \{v_c\}$ 
while  $depth < d$  do
     $V_{add} \leftarrow \{v | v \in \bigcup_{v_i \in V_{front}} \{v_j | (v_i, v_j) \in E_t\}, v \leq v_c\}$ 
     $V_{front} \leftarrow V_{add} \setminus V_c$ 
     $V_c \leftarrow V_c \cup V_{front}$ 
end while
 $G_c \leftarrow (V_c, E_c) \text{ s. t. } G_c \subseteq G_t$ 

```

The canonical partition is implemented using an index-restricted breadth-first search (BFS). Compared with regular BFS, it restricts the frontier nodes to have smaller indices than that of the canonical node. The time complexity of canonical partition equals the BFS on each neighborhood $G_n = (V_n, E_n)$, which is $\sum O(V_n + E_n) = O(V_t \times (\bar{V}_n + \bar{E}_n))$.

A.4 Complexity Benefit of Canonical Partition

We discuss the computational benefit of the canonical partition method in this section.

Search space reduction. Canonical partition uses the divide-and-conquer scheme to bring about drastic search space reduction. We denote the complexity of searching and counting all subgraphs on size V_t target graph as $S(V_t)$. The Canonical partition divides the original problem into subproblems with the total search space of $\sum_{i \in V_t} S(V_{n_i})$, where V_{n_i} stands for the size of canonical neighborhoods. Thanks to the sparse nature of real-world graphs, V_{n_i} s are generally small, even for huge target graphs. So with canonical partition, the search space is drastically reduced.

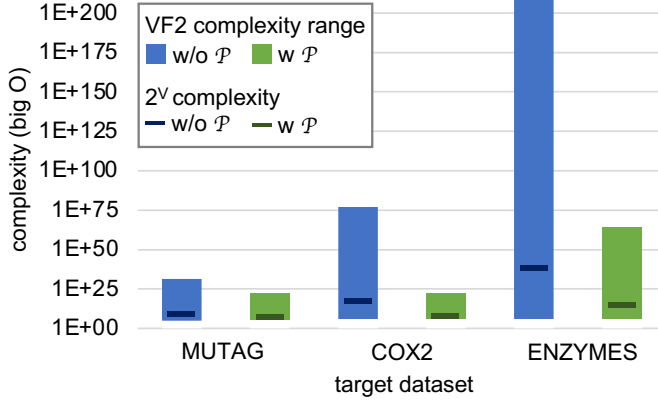


Figure 12: The complexity of subgraph counting with and without canonical partition on different target datasets. The complexity for the VF2 exact subgraph counting method is $O(V^2)$ to $O(V! \times V)$. The $O(2^V)$ complexity estimates the empirically observed average complexity.

We conduct experiments on real-world graphs to show how canonical partition fundamentally reduces the search space. Figure 12 shows the computational complexity with different assumptions in the form of S . VF2 [21] claims that the asymptotic complexity for the problem ranges from $O(V^2)$ to $O(V! \times V)$ in best and worst cases. Under such assumptions of S , the average worst-case complexity is reduced by a factor of $1/10^{70}$ with canonical partition, while the average best-case complexity stays in the same magnitude. Empirically, we observe exponential runtime growth of the subgraph counting problem. Thus, under the assumption that $S(V) = 2^V$, the average complexity is also reduced drastically by a factor of $1/10^{11}$ with canonical partition.

Redundant match elimination. Canonical partition, along with the canonical count definition, eliminates the redundant automorphic match of the query graph. Previous works [50, 68] have shown that the automorphism of the query graph can cause a large amount of redundant count. For example, the triangle query graph G_q has three symmetric nodes. We denote the triangle pattern as $G_p \subseteq G_t$ and the bijection $\mathbb{R}^3 \mapsto \mathbb{R}^3$ as $f : (v_{p_0}, v_{p_1}, v_{p_2}) \mapsto (v_{q_0}, v_{q_1}, v_{q_2})$. For the same pattern, there exist six bijections $\{f : (v_{p_0}, v_{p_1}, v_{p_2}) \mapsto (v_{q_i}, v_{q_j}, v_{q_k}) | (i, j, k) \in \text{Perm}(1, 2, 3)\}$ where $\text{Perm}(x, y, z)$ denotes all $3!$ permutations of (x, y, z) .

Canonical partition eliminates such redundant bijections by adding asymmetry, the canonical node. As discussed in Equation 2, by attributing the count to only one canonical node, the bijection f_c can be rewritten as a $\mathbb{R}^3 \mapsto \mathbb{R}$ function, $f_c : (v_{p_0}, v_{p_1}, v_{p_2}) \mapsto \max_I(v_{q_0}, v_{q_1}, v_{q_2})$. It means that each query corresponds to only one bijection instead of six, thus preventing double counting and reducing the computational complexity.

Reduction for the variation of counts. Canonical partition also reduces the variation of counts, which makes the regression task easier for the neural network as discussed in Section 1. The detailed statistics of the range of counts (maximum count minus minimum count) are shown in Appendix D.2. The canonical partition reduces the range of counts to $1/3$ on average in Figure 16.

B EXPRESSIVE POWER OF SHMP

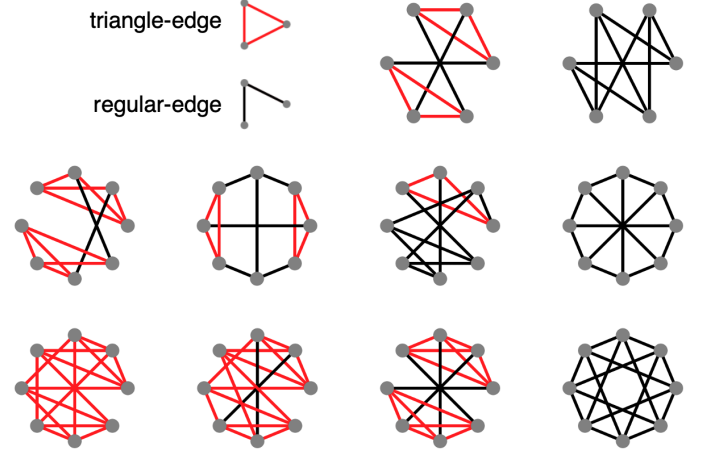


Figure 13: Examples of 1-SHMP distinguishable graphs. Any graph pair of each row cannot be distinguished by the 1-WL test. While with one layer of triangle-based SHMP, the histogram of the triangle-edge can distinguish all these graph pairs.

B.1 Theoretical Comparison with Regular Message Passing

Previous work [87] has shown that the expressive power of existing message passing GNNs is upper-bounded by the 1-WL test, and such bound can be achieved with the Graph Isomorphism Network (GIN). We prove the expressive power of SHMP with the following Lemma.

LEMMA B.1. *The SHMP version of GIN has stronger expressive power than the 1-WL test.*

By setting $\forall \phi_h^k = \phi^k$ and $\text{Agg}' = \text{Agg}$, SHMP from Equation 6 becomes an instance of GIN, which proves that SHMP-GIN is at least as expressive as GIN or the 1-WL test. The examples in Figure 13 and Table 6 further prove that one layer of triangle-based SHMP-GIN can distinguish certain graphs that the 1-WL test cannot. Thus, SHMP-GIN has stronger expressive power than the 1-WL test, exceeding the upper bound of regular message passing neural networks.

B.2 Experiments on Regular Graphs

To further illustrate the expressive power of SHMP, we show the number of graph pairs that are WL indistinguishable but SHMP distinguishable in Table 6. We collect all the connected, d -regular graphs of sizes six to twelve from the House of Graphs [17]. Among these 157 graphs, 654 pairs of graphs are indistinguishable by the 1-WL test, even with infinite iterations. In comparison, only 208 pairs are indistinguishable by the triangle-based SHMP with a single layer. So 68% of typical fail cases of the 1-WL test are easily solved with SHMP. Some examples are shown in Figure 13.

Graph Size	6	7	8	9	10	11	12
Number of Graphs	5	4	15	10	30	5	88
Number of Graph Pairs	10	6	105	45	435	10	3828
WL Indistinguishable	1	1	19	13	64	1	555
SHMP Indistinguishable	0	1	4	4	26	0	173
Error Reduction	100.0%	0.0%	78.9%	69.2%	59.4%	100.0%	68.8%

Table 6: The number of indistinguishable d -regular graph pairs for the WL-test and SHMP.

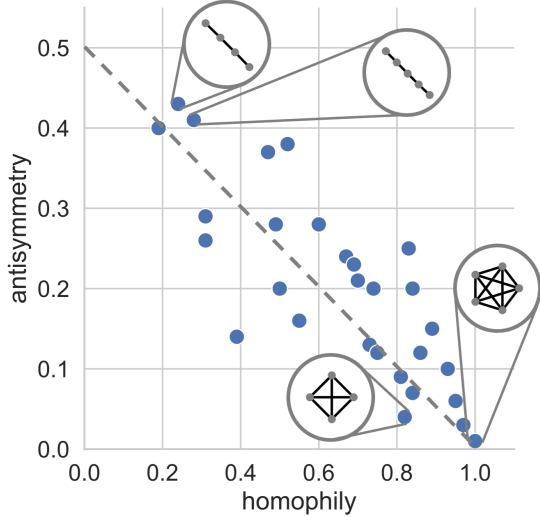


Figure 14: *Homophily* and *antisymmetry* of different queries, measured by the homophily ratio and the index-count correlation, respectively. The two inductive biases are negatively correlated.

B.3 Discussion on Substructure Enhanced GNNs

Previous substructure enhanced GNNs [53, 55] focus on the idea of high-order abstractions of the graph. However, the direct instantiation of all high-order substructures poses significant runtime overhead, which is unfriendly for the large-scale subgraph counting problem. For example, [53] has to add $\binom{n}{k}$ nodes to represent the corresponding k -order substructure of an n -node target graph. This results in massive memory overhead and heavy message passing computation. Though both of them use the three-node substructure information, experiments show that the five-layer DeSCo is $3.51\times$ faster than the five-layer 1-2-3-GNN [53] when embedding the same COX2 dataset. In contrast, DeSCo’s subgraph-based heterogeneous message passing (SHMP) focuses on the idea of distinguishing different local graph structures. By categorizing the messages on the original graph, DeSCo efficiently uses the same amount of message passing computation as traditional MPGNNs, while providing better expressive power.

C HOMOPHILY AND ANTISYMMETRY ANALYSIS OF GOSSIP PROPAGATION

Explanation. 1) *Homophily*. Since the neighborhoods of adjacent nodes share much common graph structure, they tend to have

similar canonical counts as shown in Figure 2. It is called the *homophily* of canonical counts. 2) *Antisymmetry*. As mentioned after Definition 4.2, for nodes with similar neighborhood structures, the one with a larger node index has a larger canonical count, resulting in *antisymmetry* of canonical counts. See the example target graph with neighborhood counts in Figure 3. The nodes (0, 1, 2) and (3, 5, 6) are both triangles, yet (3, 5, 6) have larger node indices, thus larger canonical counts. Figure 14 further show that *homophily* and *antisymmetry* have a negative correlation for different queries, which inspires us to learn such negative correlation to improve counting accuracy.

Example and observation. The *homophily* and the *antisymmetry* are two important inductive biases for the canonical count. The target graph in Figure 3 serves as a vivid example. The numbers in the green square indicate the canonical count value of each node. On the one hand, note that the adjacent nodes 3, 5, and 6 have the same count value of 2. Adjacent nodes 0, 1, and 2 also have the same value, 0. This homophily inductive bias suggests that taking an average of the adjacent node values can reduce the prediction error of individual nodes. On the other hand, though node 1 and node 5 have similar neighborhood graph structures, node 5, with a larger node index, has a higher canonical count value. It corresponds to the definition of canonical count as discussed in Section 4.1. This antisymmetry inductive bias suggests that the embedding phase for two structurally similar nodes with different node indices should also be different.

Quantization. We quantify the *homophily* and the *antisymmetry* inductive biases. For homophily, we treat the canonical count as the node label and use the homophily ratio from [95] to quantify how similar the count is between adjacent nodes. The homophily ratio ranges from 0 to 1. The higher the homophily ratio is, the more similar the labels will be between adjacent nodes. For antisymmetry, we use the Pearson correlation coefficient r [9] between the node index and its canonical count as the quantification metric. We quantify the different *homophily* and *antisymmetry* for the standard queries on the ENZYMES target graph.

Key insight. As shown in Figure 14, the key insight is that *homophily* and the *antisymmetry* generally have negative correlation $r = -0.82$. So the emphasis on one should suppress the other. Based on such observation, we design the gossip propagation model with learnable gates to imitate the mutually exclusive relation between the two inductive biases for different queries. As shown in Figure 7, the proposed learnable gate balances the influence of *homophily* and *antisymmetry* by controlling the direction of message passing. The gate value is trained to adapt for different queries to imitate different extents of *homophily* and *antisymmetry*.

D EXPERIMENTAL SETUP

D.1 Synthetic Dataset

The DeSCo can be pre-trained once and be directly applied to any targets. So we generate a synthetic datasets. Synthetic dataset has 1827 graphs. It is designed to generally cover various graph datasets. We use the exact counting method to generate the ground truth counts of all twenty-nine *standard queries* (queries of sizes 3, 4, 5) of this synthetic graphs.

The synthetic dataset generates each graph with a generator from the generator pool. The pool consists of six different graph generators: the Erdős-Rényi (ER) model [25], the Watts-Strogatz (WS) model [82], the Extended Barabási-Albert (Ext-BA) model [4], the Power Law (PL) cluster model [37], Barabási-Albert (BA) model [7] and the random graph generator (gnm-random-graph) from networkx [32].

The Synthetic dataset generation process first generates a set of expected #node, #edge pairs as the jobs. The jobs are then randomly assigned to the six generators. Each generator tries to generate the expected graph. The generator sets its parameters so that, from the perspective of probability, the expected #node, #edge confirms the assigned job. For the #node, #edge pairs, the Synthetic dataset uniformly generates 1380 jobs with nodes ranging from 10 to 59 and the average degree ranging from 1 to 12. It then uniformly generates 447 jobs with nodes ranging from 60 to 800 and the average degree ranging from 1 to 3. For the ER model, parameter $p = 2m/(n(n-1))$ where n and m denotes the number of nodes and edges. For the WS model, the parameter $k = 2m/n$. We set $p = 0.1$ for the model. For the Ext-BA model, the parameter $p = (m - \lfloor m/n \rfloor n)/n$. We set $q = 0.1$ for the model. For the power model, parameter $k = \lfloor (n - \sqrt{n^2 - 4m})/2 \rfloor$, $p = (m - (n - k)k)/((k - 1)(n - k))$.

D.2 Query Graphs

Figure 16 shows all twenty-nine *standard queries* discussed in Section 5.1. They form the complete set of all non-isomorphic, connected, undirected graphs with three to five nodes. Figure 17 shows all sixteen *large query graphs* discussed in Section 5.3. They are frequent subgraphs with six to thirteen nodes in the ENZYMES dataset.

The figures also show the range of the ground truth counts of these queries on different target graphs from the ENZYMES dataset. The range of canonical counts on the corresponding neighborhoods is also shown. Note how *canonical partition* reduces the range of counts for the regression task of GNNs.

D.3 Target Graphs

To demonstrate DeSCo’s generalization power, we use real-world datasets from various domains for evaluation. All the datasets are treated as undirected graphs without node or edge features in alignment with the setting of the approximate heuristic method [15]. Figure 15 shows the graph statistics of the canonical neighborhoods from the real-world datasets and the Synthetic dataset. The Synthetic dataset successfully covers most of the canonical neighborhoods for many graph statistics, which confirms the conclusion in Figure 9.

D.4 Hyper-parameter Configurations

DeSCo configurations. For DeSCo’s canonical partition stage, we set $d = 4$ for all the tasks according to Theorem 1. For DeSCo’s neighborhood counting stage, it contains two GNNs to encode target and query graphs into embedding vectors, and a regression model to predict canonical count based on the vectors. For the GNN encoders, we use the triangle-based message passing variant of GraphSAGE as shown in Table 5. The SHMP GNN has 8 layers with a feature size of 64. The canonical node of the neighborhood is marked with a special node type. The adjacent matrix A is used to find the triangle and define the heterogeneous edge type with Equation 18.

$$E_{triangle} = \{(i, j) | (A \odot A^2)_{ij} > 0\} \quad (18)$$

For the Multilayer perceptron (MLP) of neighborhood counting, we use two fully-connected linear layers with 256 hidden feature size and LeakyReLU activation.

For the gossip propagation stage, we use a two-layer GNN with 64 hidden feature size and a learnable gate as described in Equation 7. The learnable gate is a two-layer, 64-hidden-size MLP that takes the query embedding vector from the neighborhood counting stage and outputs the gate values for each GNN layer. The neighborhood counting prediction is expanded to 64 dimensions with a Linear layer and concatenated with the query embedding as the input for the two-layer GNN.

Neural baseline configurations. We follow the configurations of the official implementations of neural baselines and adapt them to our settings. They both contain two GNN encoders and a regression model like DeSCo’s neighborhood counting model.

For LRP, we follow the official configurations for the ZINC dataset to use a deep LRP-7-1 graph embedding layer with 8 layers and hidden dimension 8. The regression model is the same as DeSCo.

For DIAMNet, we follow the official configurations for the MUTAG dataset to use GIN with feature size 128 as GNN encoders. The number of GNN layers is expanded from 3 to 5. The regression model is DIAMNet with 3 recurrent steps, 4 external memories, and 4 attention heads.

For DMPNN, we straightly use the official implementation of DMPNN with the official configurations for the MUTAG dataset. It use a 3-layer DMPNN.

Training details. For LRP and DIMANet, We use $C_c \leftarrow \log_2(C_c + 1)$ normalization for the ground truth canonical count C_c to ease the high count variation problem. When evaluating the MSE of predictions, $\hat{C}_c \leftarrow 2^{\hat{C}_c} - 1$ is used to undo the normalization. We use the SmoothL1Loss with $\beta = 1.0$ from PyTorch [58] as the loss function to perform the regression task of neighborhood counting and gossip propagation.

$$\mathcal{L}(\hat{C}_c, C_c) = \begin{cases} 0.5(\hat{C}_c - C_c)^2 & |\hat{C}_c - C_c| < 1 \\ |\hat{C}_c - C_c| - 0.5 & \text{otherwise} \end{cases} \quad (19)$$

We use the Adam optimizer for neighborhood counting and gossip propagation and set the learning rate to 0.001.

For DMPNN, we adopt the official implementation with its prescribed configurations for the MUTAG dataset. While training with the Synthetic dataset, the ground truth information of 8 queries

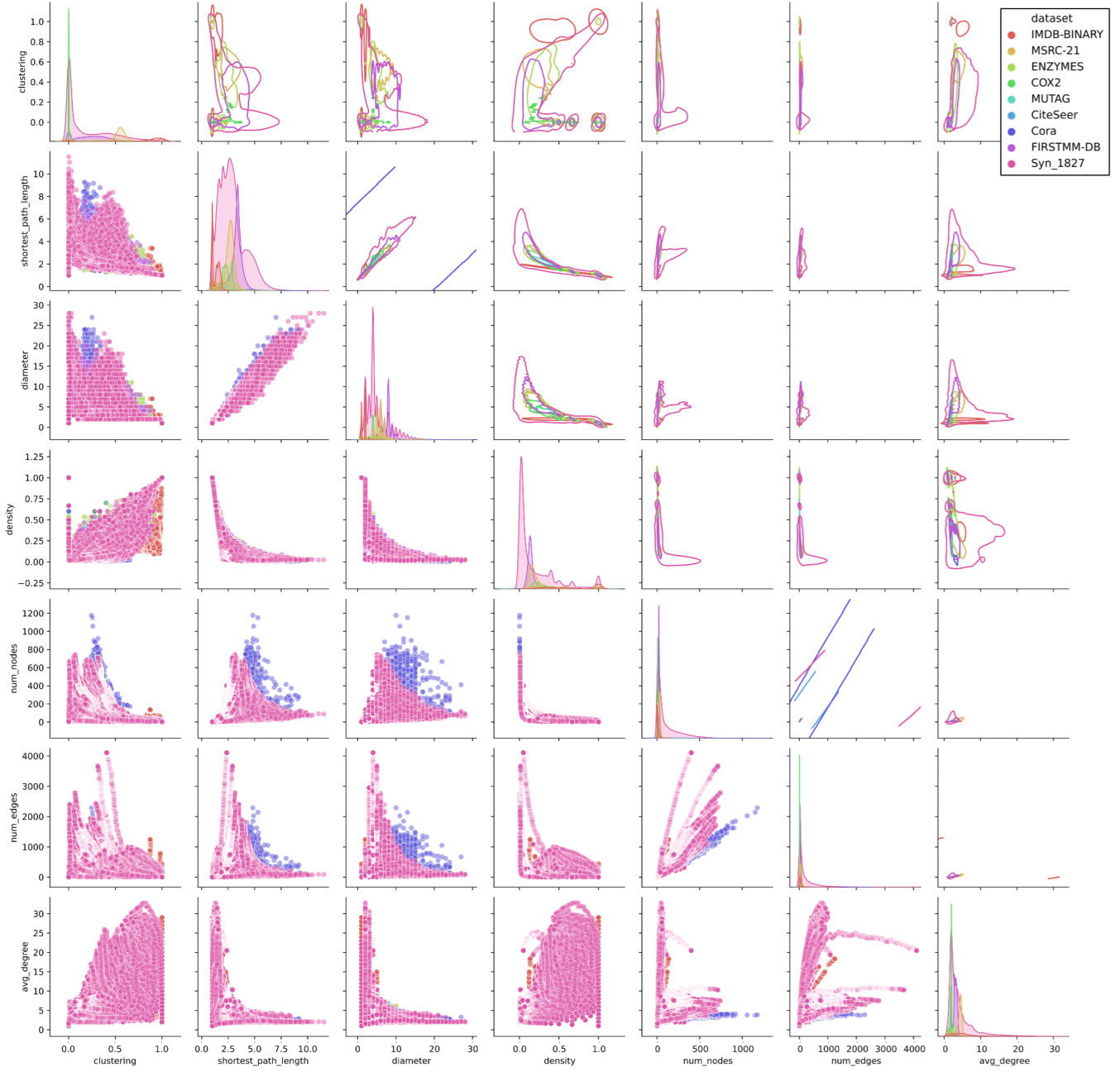


Figure 15: The canonical neighborhoods’ statistics of real-world datasets and the synthetic dataset. The synthetic dataset covers most of the real-world datasets, providing a strong foundation for DeSCo’s generalization ability.

of size 5 exceeds the maximum size that DMPNN support due to its scalability limit. Consequently, we omit these queries from the training set. This omission might marginally affect the accuracy of size 5 query predictions, but its impact on other query sizes remains minimal.

We align the computational resources when training different neural methods. DeSCo, DIAMNet, and DMPNN have similar training efficiency, so DeSCo’s neighborhood counting model, DIAMNet

and DMPNN are both trained for 300 epochs. After training the neighborhood counting model, DeSCo’s gossip propagation model is trained for 50 epochs with little resource consumption. In contrast, LRP is much slower. Even given twice training time, it can only be trained for 50 epochs.

Approximate heuristic configurations. For the MOTIVO baseline, we follow the official setting and use 10^7 samples for each






























G_q	range G_t	range G_c	G_q	range G_t	range G_c	G_q	range G_t	range G_c
	339	33		1773	288		37	15
	61	12		195	48		37	19
	282	48		391	71		19	12
	778	107		499	121		62	25
	244	42		386	86		23	13
	67	15		44	13		24	14
	67	17		115	28		9	5
	16	5		156	51		7	3
	203	80		52	16		2	1
	1766	301		86	21			

Figure 16: The *standard query graphs*, the range of the subgraph counts on target graphs G_t , and the range of the canonical counts on neighborhoods G_c . The statistics come from the ENZYMES dataset.

dataset. If the dataset has many graphs, the samples are evenly distributed on each target graph.

E ADDITIONAL NUMERIC RESULTS

All of the average results are calculated by geometric mean.

E.1 Ablation study

We perform an ablation study across 8 real-world datasets to demonstrate the effectiveness of our canonical partition, SHMP, and gossip propagation components. Numerical results are presented in Table 7, Table 8, and Table 9. The ablation study shows that all three components are essential for DeSCo’s performance.

E.2 Count Distribution Prediction

To the best of our knowledge, DeSCo is the first approximate method that predicts the subgraph count distribution over the whole target graph. We use the canonical count of each node as the ground truth for the distribution prediction accuracy analysis.

The canonical count represents the number of *patterns* in each node’s neighborhood while avoiding missing or double counting as discussed in Section 4.1. Following the setup in Section 5.1, we use all the size 3 – 5 *standard query graphs* to test the distribution performance of DeSCo on different target graphs. The normalized MSE is the mean square error of the canonical count prediction of each (query, target graph node) pair divided by the variance of the (query, target graph node) pair’s true canonical count. The MAE is the mean absolute error of the canonical count prediction of each (query, target graph node) pair.

Experiments show DeSCo achieves a low 3.8×10^{-3} normalized MSE for the count distribution prediction task. See Table 11 for the detailed results. A visualization of DeSCo’s distribution prediction on the CiteSeer dataset is also shown in Figure 2. Note how DeSCo accurately predicts the distribution while providing meaningful insight on the graph.

















G_q	range G_t	range G_c	G_q	range G_t	range G_c	G_q	range G_t	range G_c
	3573	688		1631	453		6	5
	615	150		804	318		16	16
	5223	1035		2071	732		3	3
	1543	448		526	330		9	8
	4049	930		41	17			
	1725	595		928	296			

Figure 17: The *large query graphs*, the range of the subgraph counts on target graphs G_t , and the range of the canonical counts on neighborhoods G_c . The statistics come from the ENZYMES dataset.

Dataset	MUTAG			COX2			ENZYMES			IMDB-BINARY		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
w/o \mathcal{P}	6.0E-1	3.1E-1	3.4E-1	1.3E+0	6.8E-1	6.8E-1	5.3E-1	3.7E-1	4.6E-1	9.5E+07	1.0E+10	3.5E+14
w \mathcal{P}	1.8E-2	1.6E-2	1.7E-2	1.1E-2	1.2E-2	9.3E-3	1.1E-2	3.7E-2	4.7E-2	7.9E-03	2.1E-01	4.5E-01
MAE												
w/o \mathcal{P}	8.3E+0	4.8E+0	3.6E+00	3.0E+1	1.6E+1	9.9E+0	3.4E+1	3.5E+1	3.2E+1	2.3E+05	1.4E+07	1.8E+10
w \mathcal{P}	1.4E-1	9.8E-1	4.8E-1	2.7E+0	2.3E+0	1.2E+0	5.5E+0	9.3E+0	9.5E+0	2.5E+01	3.0E+02	1.6E+03
Dataset	MSRC-21			CiteSeer			Cora			FIRSTMM-DB		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
w/o \mathcal{P}	8.4E+00	1.3E+02	6.7E+02	inf	inf	inf	3.3E+29	5.3E+39	1.0E+48	inf	inf	inf
w \mathcal{P}	7.5E-03	6.9E-03	6.7E-02	6.9E-05	1.1E-01	1.7E-01	5.3E-03	2.2E-01	7.0E-02	1.7E-03	2.3E-02	5.1E-02
MAE												
w/o \mathcal{P}	4.7E+02	2.3E+03	4.3E+03	inf	inf	inf	9.4E+18	1.1E+25	1.6E+30	inf	inf	inf
w \mathcal{P}	1.8E+01	3.3E+01	1.2E+02	7.3E+01	1.3E+04	1.1E+05	1.4E+03	7.5E+04	5.6E+05	1.8E+02	6.7E+02	1.0E+03

Table 7: Normalized MSE performance with or without canonical partition. Since gossip propagation relies on the output of neighborhoods, it’s also removed for both for a fair comparison.

F RUNTIME COMPARISON

F.1 Experiment setup

We configure DeSCo and baselines as follows. For the exact method VF2 [21], we use the Python implementation from the graph processing framework [32] and use Python’s concurrent standard library to enable multiprocessing on four CPU cores. For the exact method IMSM [73], we use the official c++ implementation with four CPU cores. We use the IMSM-recommended method configurations: GQL [35] as the filtering method, RI [12] as the ordering

method, and LFTJ [11, 34] as the enumeration method. The failing set pruning optimization is also enabled. For the approximate heuristic method MOTIVO [15], we use the official c++ implementation with four CPU cores. For the neural method DeSCo, we use the Python implementation with one CPU core and one GPU core.

We use Intel Xeon Gold 6226R CPU with 2.90GHz frequency and NVIDIA GeForce RTX 3090 GPU for runtime tests. All the methods are set to count the induced subgraphs in the ENZYMES dataset. Note that IMSM can only perform non-induced subgraph counting. So VF2, MOTIVO, and DeSCo are set to perform induced

Dataset	MUTAG			COX2			ENZYMES			IMDB-BINARY		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
GCN	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
SAGE	4.0E-01	9.6E-02	5.6E-01	3.1E-01	5.4E-02	3.8E-01	5.1E-01	3.2E-01	3.0E-01	2.9E+01	1.5E+01	4.6E+00
GIN	8.4E-02	7.3E-02	1.6E-01	3.6E-02	6.6E-02	2.3E-01	1.9E-01	7.1E-02	1.1E-01	1.1E+00	1.0E+00	1.0E+00
ID-GNN	3.3E-02	2.9E-02	1.5E-02	1.5E-02	7.0E-03	2.9E-02	1.9E-02	2.6E-02	8.1E-02	1.1E+00	1.0E+00	1.0E+00
SAGE+SHMP	1.8E-02	1.6E-02	1.7E-02	1.1E-02	1.2E-02	1.0E-02	1.1E-02	3.7E-02	4.7E-02	7.9E-03	2.1E-01	4.5E-01
MAE												
GCN	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
SAGE	6.8E+00	2.3E+00	7.4E+00	1.4E+01	4.3E+00	1.8E+01	4.1E+01	3.4E+01	2.9E+01	5.9E+02	2.4E+03	6.1E+03
GIN	3.5E+00	2.3E+00	1.7E+00	5.8E+00	6.7E+00	6.4E+00	2.5E+01	1.8E+01	1.8E+01	3.0E+02	8.3E+02	2.6E+03
ID-GNN	1.9E+00	1.0E+00	5.3E-01	3.2E+00	1.7E+00	1.8E+00	6.2E+00	8.9E+00	1.3E+01	3.0E+02	8.3E+02	2.6E+03
SAGE+SHMP	1.4E+00	9.8E-01	4.8E-01	2.7E+00	2.3E+00	1.2E+00	5.5E+00	9.3E+00	9.5E+00	2.5E+01	3.0E+02	1.6E+03
Dataset	MUTAG			COX2			ENZYMES			IMDB-BINARY		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
GCN	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
SAGE	1.9E-01	6.3E-01	3.4E-01	1.5E+00	2.7E-01	9.7E-01	9.4E+00	5.9E-01	1.1E+00	3.2E-02	1.7E-01	2.2E-01
GIN	2.4E+00	1.6E+00	1.3E+00	1.9E+00	1.5E+00	1.2E+00	1.9E+00	1.3E+00	1.1E+00	1.1E+00	1.2E+00	1.1E+00
ID-GNN	3.0E+00	1.7E+00	1.3E+00	2.2E+00	1.5E+00	1.2E+00	2.1E+00	1.3E+00	1.1E+00	1.5E+00	1.2E+00	1.1E+00
SAGE+SHMP	7.5E-03	6.9E-03	6.7E-02	6.9E-05	1.1E-01	1.7E-01	5.3E-03	2.2E-01	7.0E-02	1.7E-03	2.3E-02	5.1E-02
MAE												
GCN	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
SAGE	9.6E+01	3.0E+02	3.7E+02	9.7E+03	2.7E+04	3.1E+05	5.5E+04	1.7E+05	2.1E+06	8.3E+02	2.3E+03	2.6E+03
GIN	3.1E+02	4.9E+02	6.4E+02	1.1E+04	6.0E+04	3.7E+05	2.3E+04	2.2E+05	2.2E+06	3.8E+03	4.8E+03	4.9E+03
ID-GNN	3.6E+02	5.0E+02	6.4E+02	1.2E+04	6.0E+04	3.7E+05	2.5E+04	2.2E+05	2.2E+06	4.6E+03	5.0E+03	4.9E+03
SAGE+SHMP	1.8E+01	3.3E+01	1.2E+02	7.3E+01	1.3E+04	1.1E+05	1.4E+03	7.5E+04	5.6E+05	1.8E+02	6.7E+02	1.0E+03

Table 8: Normalized MSE and MAE performance with different GNN models for neighborhood counting.

Dataset	MUTAG			COX2			ENZYMES			IMDB-BINARY		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
w/o propagation	1.8E-02	1.6E-02	1.7E-02	1.1E-02	1.2E-02	1.0E-02	1.1E-02	3.7E-02	4.7E-02	7.9E-03	2.1E-01	4.5E-01
w propagation	2.3E-03	8.4E-04	6.5E-03	6.9E-04	5.3E-04	5.4E-03	5.3E-03	5.7E-02	5.3E-02	8.7E-03	2.1E-01	4.5E-01
MAE												
w/o propagation	1.4E+00	9.8E-01	4.8E-01	2.7E+00	2.3E+00	1.2E+00	5.5E+00	9.3E+00	9.5E+00	2.5E+01	3.0E+02	1.6E+03
w propagation	5.1E-01	1.9E-01	3.0E-01	6.2E-01	4.0E-01	8.1E-01	3.5E+00	1.1E+01	9.8E+00	2.3E+01	3.0E+02	1.6E+03
Dataset	MSRC-21			CiteSeer			Cora			FIRSTMM-DB		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE												
w/o propagation	7.5E-03	6.9E-03	6.7E-02	6.9E-05	1.1E-01	1.7E-01	5.3E-03	2.2E-01	7.0E-02	1.7E-03	2.3E-02	5.1E-02
w propagation	2.6E-03	3.9E-03	8.5E-02	3.5E-05	9.7E-02	1.6E-01	4.2E-03	2.1E-01	6.3E-02	2.1E-03	3.4E-02	5.4E-02
MAE												
w/o propagation	1.8E+01	3.3E+01	1.2E+02	7.3E+01	1.3E+04	1.1E+05	1.4E+03	7.5E+04	5.6E+05	1.8E+02	6.7E+02	1.0E+03
w propagation	1.0E+01	2.5E+01	1.3E+02	6.0E+01	1.2E+04	1.1E+05	1.3E+03	7.3E+04	5.4E+05	1.5E+02	7.2E+02	9.6E+02

Table 9: The normalized MSE and MAE performance with and without gossip propagation.

subgraph counting tasks, while IMSM performs non-induced tasks for runtime comparison. For query sizes no larger than five nodes, the *standard queries* from Section 5.1 are used. For query sizes larger than five, the same thirty queries of each size are selected for VF2 and DeSCo. We cannot assign specific queries for MOTIVO, so it

is set to output the count of any thirty queries of each size. In the experiments, the data loading and graph format conversion time is ignored for all methods. We further extend the time budget for MOTIVO to 60 minutes. The results show that DeSCo achieves

Dataset	CiteSeer			Cora			FIRSTMM-DB		
Query-Size	3	4	5	3	4	5	3	4	5
normalized MSE									
LRP	inf	inf	inf	inf	inf	inf	1.5E+00	1.2E+00	1.1E+00
DIAMNet	2.0E+00	1.5E+00	1.2E+00	1.0E+10	3.2E+07	3.7E+04	1.7E+14	5.5E+13	9.7E+12
DMPNN	9.5E+04	2.5E+02	6.8E+01	1.8E+05	1.1E+02	6.7E+01	5.6E+02	2.5E+02	1.6E+02
DeSCo	3.5E-05	9.7E-02	1.6E-01	4.2E-03	2.1E-01	6.3E-02	2.1E-03	3.6E-02	5.4E-02
MAE									
LRP	inf	inf	inf	inf	inf	inf	4.5E+03	4.9E+03	4.9E+03
DIAMNet	1.1E+04	6.0E+04	3.6E+05	2.1E+09	1.6E+09	8.3E+08	1.7E+10	1.3E+10	6.6E+09
DMPNN	6.1E+06	7.6E+06	8.7E+06	1.8E+07	2.4E+07	3.0E+07	2.2E+05	2.9E+05	3.3E+05
DeSCo	6.0E+01	1.2E+04	1.1E+05	1.3E+03	7.3E+04	5.4E+05	1.5E+02	7.3E+02	9.6E+02

Table 10: Normalized MSE and MAE performance of neural methods on large targets with standard queries.

Dataset	MUTAG			COX2			ENZYMES		
Query-Size	3	4	5	3	4	5	3	4	5
Norm. MSE	7.51E-2	2.36E-1	1.71E+0	4.94E-4	5.63E-4	1.44E-2	4.74E-5	5.69E-5	5.66E-4
MAE	2.97E-4	1.19E-3	1.90E-2	3.90E-4	5.19E-4	1.71E-2	7.60E-2	1.51E-1	3.15E-1

Table 11: DeSCo’s count distribution prediction error under normalized MSE and MAE. Use the canonical count of each target graph node as the ground truth.

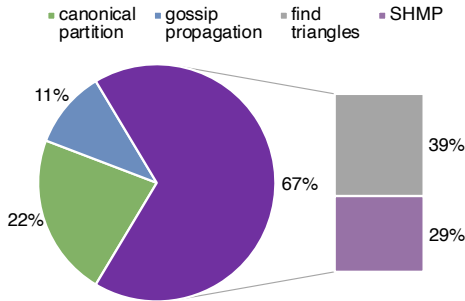


Figure 18: Runtime breakdown of DeSCo.

15×, 53×, and 120× speedup over MOTIVO for query size 13 to 15, respectively.

As Figure 18 shows, currently DeSCo’s triangle finding in *neighborhood counting* takes the majority of the runtime, which can be easily substituted with other efficient implementations, e.g., [24], to further speed up DeSCo.

F.2 Asymptotic complexity

For the proposed DeSCo’s three-step pipeline, assuming the average canonical neighborhood G_n of the target graph G_t has V_n nodes and E_n edges. The time complexity for canonical partition is the index-restricted breadth-first search starting from all the target vertices as shown in Appendix A.3, which is $O(V_t \times (\tilde{V}_n + \tilde{E}_n))$. The time complexity for neighborhood counting consists of triangle counting and heterogeneous message passing on G_q and G_t . The complexity of triangle counting is $O(E^{3/2})$ on the target and query

graph [38]. The heterogeneous message passing has the complexity of regular GNNs [48] on the V_t neighborhoods and the queries, which is $O(V_t \times (\tilde{V}_n + \tilde{E}_n)) + O(V_q + E_q)$. For gossip propagation, the time complexity also equals a regular GNN, which is $O(E_t + V_t)$.

In conclusion, the overall time complexity of DeSCo is $O(E_t^{3/2} + V_t \times (\tilde{V}_n + \tilde{E}_n)) + O(E_q^{3/2} + V_q)$. In real-world graphs, the common contraction of neighborhoods [83] makes \tilde{V}_n and \tilde{E}_n relatively small. So the major asymptotic complexity comes from the triangle counting on the target graph, which only has polynomial time complexity.

In contrast, for the heuristic approximate method MOTIVO, the build-up phase alone has time complexity $O(a^{V_q} \times E_t)$ for some $a > 0$. So it suffers from exponential runtime growth. For exact method VF2, the time complexity is $O(V^2)$ to $O(V! \times V)$ where $V = \max\{V_t, V_q\}$. In practice, we generally observe exponential runtime growth. Experiments of Figure 11 confirm the above analysis.

F.3 Discussion on GPU-based exact methods

Like CPU-based exact methods, GPU-based exact methods also suffer from high asymptotic complexity, thus not scalable. We wish to provide a precise comparison with existing GPU-based exact methods. Unfortunately, existing method [45] does not open-source their code. Thus, we directly compare the reported results and find that our method can easily beat it. For example, when both are set to find size-4 to size-8 queries on the 10^4 -scale target graphs (CE and ENZYMES), [45] takes 1×10^4 seconds (Figure 4(a) in [45]), while DeSCo only takes 1.7×10^2 seconds. Though the query graphs are not exactly the same, DeSCo is much faster in general.

Test Dataset	IMDB-BINARY			MSRC-21			CiteSeer			Cora			FIRSTMM-DB			MUTAG		
Query-Size	3	4	5	3	4	5	3	4	5	3	4	5	3	4	5	3	4	5
normalized MSE																		
Pre-train on Existing	overflow			1.1E+01	1.9E+00	1.1E+00	1.7E-01	5.1E-01	6.8E-01	4.4E-01	1.1E+00	8.0E-01	1.1E-01	1.1E-01	1.6E-01	6.51E-03	3.40E-03	8.70E-02
Pre-train on Synthetic	8.5E-03	2.1E-01	4.5E-01	2.5E-03	3.8E-03	8.7E-02	3.5E-05	9.7E-02	1.6E-01	4.2E-03	2.1E-01	6.3E-02	2.1E-03	3.6E-02	5.4E-02	2.3E-03	8.4E-04	6.5E-03
MAE																		
Pre-train on Existing	overflow			2.9E+02	4.5E+02	6.1E+02	3.3E+03	3.4E+04	2.5E+05	1.1E+04	1.7E+05	1.6E+06	1.2E+03	1.7E+03	2.2E+03	7.85E-01	3.68E-01	1.40E+00
Pre-train on Synthetic	2.4E+01	3.0E+02	1.6E+03	1.0E+01	2.5E+01	1.3E+02	6.0E+01	1.2E+04	1.1E+05	1.3E+03	7.3E+04	5.4E+05	1.5E+02	7.3E+02	9.6E+02	5.1E-01	1.9E-01	3.0E-01

Table 12: Normalized MSE and MAE performance with different training datasets. When pre-training on existing datasets, IMDB-BINARY and MSRC-21 use MUTAG; CiteSeer uses Cora; Cora and FIRSTMM-DB use CiteSeer; MUTAG uses ENZYMES.

G ADDITIONAL RESULTS ANALYSIS FOR LARGE QUERIES

To give an even more in-depth understanding of the performance for large queries, we additionally provide the results with more evaluation metrics.

G.1 Q-error Analysis

Definition. Given the ground truth subgraph count C of query G_q in target G_t , as well as the estimated count \hat{C} . We use the definition of q-error from previous work [94].

$$e_q(G_q, G_t) = \max \left\{ \frac{C}{\hat{C}}, \frac{\hat{C}}{C} \right\}, e_q \in [1, +\infty) \quad (20)$$

The q-error quantifies the factor that the estimation differs from the true count. The more it is close to 1, the better estimation. In [94], there is also an alternative form of q-error used in figures to show the systematic bias of predictions.

$$e_q(G_q, G_t) = \frac{\hat{C}}{C}, e_q \in (0, +\infty) \quad (21)$$

We follow the previous settings and use Equation 21 in our visualization.

Experimental results. We reassess the performance of DeSCo with large queries. The results are shown in Figure 19. The data that $C = 0$ is ignored for mathematic correctness. The box of MOTIVO on MUTAG is too close to zero to be shown in the figure. DeSCo’s q-error is the closest to 1 with minimal spread. It shows how DeSCo excels in systematic error and consistency compared with the baselines.

Dataset	MUTAG	COX2	ENZYMES
MOTIVO	1.2E+01	3.2E+00	1.4E+00
LRP	7.6E-01	1.1E+00	6.7E-01
DIAMNet	2.3E+00	2.4E-01	8.9E+01
DeSCo	1.5E-01	1.2E-01	4.0E-01

Table 13: Normalized MSE of approximate heuristic and neural methods on subgraph counting of sixteen large queries.

Limitations of q-error. Despite the advantage of demonstrating relative error, the q-error metric also has obvious limitations, thus not being chosen as the major evaluation metric. In [94], the authors assume $C \geq 1$ and $\hat{C} \geq 1$. However, this assumption may not hold,

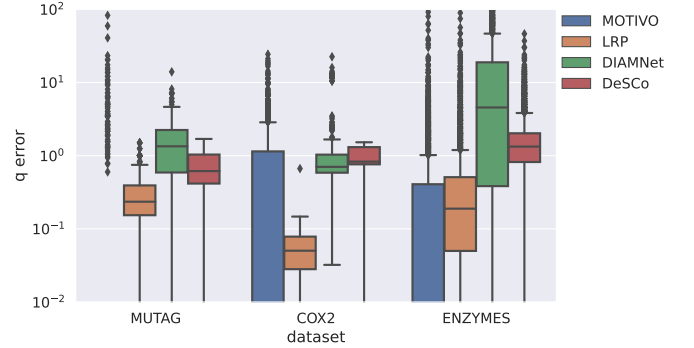


Figure 19: The q-error box plot of large query-target pairs. The q-error (y-axis) is clipped at 10^{-2} and 10^2 . For q-error, the closer to 1, the better.

given that the query graph may not (or is predicted to) exist in the target graph, especially for larger queries. The zero or near-zero denominators greatly influence the average q-error. It causes the overestimation of the subgraph existence problem instead of the subgraph counting problem.

G.2 MSE Analysis

Definition. We follow the same setting in Figure 8 and show the normalized MSE for predicting the subgraph count of large queries. Note that in a few cases, the tested large queries of a certain size may not exist in the target graph. For example, the two size-thirteen queries in Figure 17 do not exist in the CiteSeer dataset. To prevent divide-by-zero in normalization, the MSE is normalized by the variance of ground truth counts of all large queries, instead of being normalized for each query size.

Experimental results. The experimental results are shown in Table 13. DeSCo demonstrates the lowest MSE on all tested target graphs.

Received 10 August 2023