Beyond Neighbors: Distance-Generalized Graphlets for Enhanced Graph Characterization

Yeongho Kim
KAIST
Seoul, Republic of Korea
yeongho@kaist.ac.kr

Yuyeong Kim
KAIST
Seoul, Republic of Korea
youyoung@kaist.ac.kr

Geon Lee
KAIST
Seoul, Republic of Korea
geonlee0325@kaist.ac.kr

Kijung Shin
KAIST
Seoul, Republic of Korea
kijungs@kaist.ac.kr

Abstract

Graphs are widely used to model complex systems across various domains, including social networks and biological systems. A key task in graph analysis is identifying recurring structural patterns, known as graphlets, which capture connectivity among a fixed-size subset of nodes. While graphlets have been extensively applied in tasks such as measuring graph similarity and identifying communities, conventional graphlets focus only on direct connections between nodes. This limitation overlooks potential insights from more distant relationships within the graph structure.

In this paper, we introduce (d, s)-graphlets, a generalization of size-s graphlets that incorporates indirect connections between nodes up to distance d. This new formulation provides a more fine-grained and comprehensive understanding of local graph structures. To efficiently count (d, s)-graphlets in a graph, we present EDGE, an exact counting algorithm that employs optimized combinatorial techniques to significantly reduce computational complexity compared to naive enumeration. Our empirical analysis across diverse real-world datasets demonstrates that (d, s)-graphlets provide superior graph characterization, outperforming conventional graphlets in a graph clustering task. Moreover, our case studies show that (d, s)-graphlets uncover non-trivial insights that would remain undiscovered when using conventional graphlets.

CCS Concepts

• **Information systems** → **Data mining**.

Keywords

Graphlet, Distance Generalization, Network Analysis

ACM Reference Format:

Yeongho Kim, Yuyeong Kim, Geon Lee, and Kijung Shin. 2025. Beyond Neighbors: Distance-Generalized Graphlets for Enhanced Graph Characterization. In *Proceedings of the ACM Web Conference 2025 (WWW '25), April 28-May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3696410.3714558

1 INTRODUCTION

Graphs are widely used to model complex systems across various domains, from social networks to biological systems. A key task in understanding and predicting the behaviors of these systems is identifying recurring structural patterns, which can provide insights into their underlying dynamics.

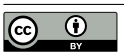
Among the various approaches, *graphlets* [47, 48] describe connectivity patterns among a small set of nodes (typically 3, 4, or 5 nodes). Graphlets capture local structures within a graph, and real-world graphs can often be distinguished by their domain, or from random graphs, based on the occurrence patterns of the graphlets. In practice, the occurrences of each graphlet within a given graph are counted [51], and these counts are then used to measure graph similarity [54], detect anomalies [6, 26], classify nodes [17, 31], or identify communities [8, 35, 70].

While graphlets are defined to capture connectivity patterns based only on direct connections in a general context of graph analysis, both traditional and recent studies have highlighted the potential of exploring relationships beyond direct connections. The significance of relationships between nodes that are not directly connected (i.e., at a distance of 2 or larger) has long been recognized in social science to enhance the contextual interpretation of nodes [37]. More recently, incorporating non-neighboring nodes has been shown to offer key benefits across multiple domains, including improved feature representation in machine learning tasks, with applications in biology [4], recommendation systems [16], and general graph machine learning [23, 62, 67, 69].

Motivated by these insights, in this paper, we introduce **(d, s)-graphlets**, a novel generalization of size-s graphlets that accounts for indirect connections between nodes up to distance d. We first define d-edges, which generalize edges by representing relationships between non-neighboring nodes at a distance of d. Using these higher-order connections, we define (d, s)-graphlets to describe local connectivity patterns, incorporating all 1-edges to d-edges while distinguishing connections based on their distances. This extension allows for a more fine-grained and comprehensive analysis of local graph structures, revealing patterns that would otherwise remain undiscovered with conventional graphlets. An example is shown in Figure 1, where (d, s)-graphlets effectively capture and distinguish local structural patterns, while conventional graphlets fail to differentiate or identify them.

Our comprehensive analysis using 13 real-world datasets from 5 different domains reveals that (d, s)-graphlets are highly effective at capturing local structural patterns. Specifically, the counts of each (d, s)-graphlet (spec., counts relative to those of null models) show better domain-based differentiation between graphs, compared to conventional graphlets. This enhanced characterization highlights the importance of modeling relationships beyond immediate neighbors for a more accurate analysis of local graph structures.

As a means to count the occurrences of each (d, s)-graphlet in a graph, we develop **EDGE** (**E**xact Counting of **D**istance-**G**eneralized



WWW '25, Sydney, NSW, Australia

ACM ISBN 979-8-4007-1274-6/25/04
https://doi.org/10.1145/3696410.3714558

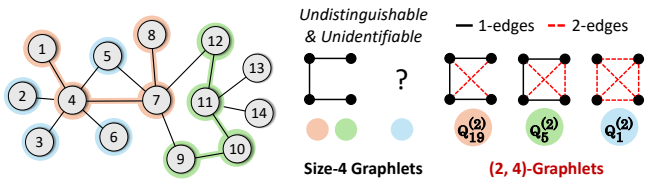


Figure 1: A sample graph and three sets of 4-node subgraphs, where black solid lines indicate direct connections. In (2-4) graphlets, a red dotted line between two nodes indicates that their distance is 2, and the absence of any line means that their distance is greater than 2. Graphlets, which only account for direct connections between nodes, (1) cannot distinguish between two subgraphs {1, 4, 7, 8} and {9, 10, 11, 12}, and (2) cannot describe subgraphs with disconnected nodes, such as {2, 3, 5, 6}. Our proposed (d, s)-graphlets address these limitations by considering relationships that extend beyond direct connections (distance ≥ 2), allowing for more fine-grained and comprehensive local structure analysis.

Graphlets), an algorithm for exactly counting instances of each $(2, 3)$-, $(3, 3)$-, and $(2, 4)$-graphlets. To avoid exhaustive enumeration, EDGE categorizes (d, s)-graphlets into non-deducible, semi-deducible and deducible (d, s)-graphlets based on structural properties. It selectively enumerates instances of non/semi-deducible (d, s)-graphlets, and using their counts, rapidly computes the count of deducible (d, s)-graphlets through combinatorial methods without enumeration. Moreover, EDGE employs a specialized directed acyclic graph that models relationships between nodes up to distance d, further enhancing its speed and scalability.

To summarize, our contributions are:

- **Concept.** We present a novel extension of graphlets that captures relationships between non-neighboring nodes (Section 4).
- **Algorithm.** We develop an efficient algorithm for exactly counting the occurrences of each (d, s)-graphlet. EDGE is up to 14.86× faster than a naive counting method (Section 5).
- **Discoveries.** We show that (d, s)-graphlets exhibit strong characterization power in distinguishing real-world graphs (Section 6).

Reproducibility. Our code and datasets are available at **https://github.com/thisis05/EDGE**.

2 RELATED WORK

In this section, we review previous work relevant to our study.

Local structural patterns and graphlets. Mining local structural patterns from graphs is a common approach for understanding the underlying dynamics of complex systems [19, 28, 65]. A key challenge is identifying structural properties that distinguish real-world graphs from random ones, as these distinctions provide valuable insights into the behavior and organization of such systems. Among various analytic tools for graph analysis, graphlets [47, 48] have been effective in characterizing network structures. As fundamental building blocks of graphs, the counts of graphlets serve as characteristic measures used to assess graph similarity [54], detect anomalies [6, 26], classify nodes [17, 31], and identify communities [8, 35, 70]. Recently, graphlets have also been leveraged to enhance the graph machine learning techniques [15, 18, 31]. Graphlets have been extended in various directions by incorporating node or edge labels [25, 53], node attributes [55], edge weights [61], hyperedges [29, 30], and multi-layer structures [10, 50]. These existing concepts commonly focus only on direct node connections.

Graphlet counting algorithm. Various methods have been proposed to count graphlets in a graph. Early approaches enumerate all connected subgraphs with a small number of nodes [39, 41, 63, 64]. To improve scalability, later methods take an analytical approach, and specifically they deduce the count of some graphlets based on the counts of others [1, 2, 27, 38, 46]. For example, PGD [1, 2] and ESCAPE [46] decompose graphlets into smaller primitives and use their counts to derive the count of the larger ones using combinatorial arguments, significantly improving the scalability of graphlet counting and the size of the graphlets that can be counted.

Distance generalization in general graph analysis. Many prior studies have emphasized the potential of exploring relationships between nodes that are not directly connected by edges [14, 16, 37, 49, 62]. Incorporating relationships between nodes without direct connections (i.e., those at a distance greater than 1) has been shown to enhance the performance on various tasks in domains including natural language processing [9, 21, 49], biology [4], recommendation systems [16], and general graph machine learning [23, 45, 62, 67, 69]. One of the earliest such generalizations is d-clique [5, 37], where every pair of nodes in the clique is within a distance of d. Similarly, a d-club [5, 42] is defined as a maximal subset of nodes in which the induced subgraph has a diameter of at most d. More recently, k-cores have been generalized to (k, d)-cores [14, 20, 36, 56], which ensure that each node has at least k other nodes within a distance of d. These generalizations have revealed interesting patterns that emerge beyond direct relationships.

Table 1: Frequently-used notations.

Notation	Definition
$G = (V, E)$	a graph with nodes V and edges E
$\delta(u, v)$	distance between nodes u and v
$E^{(d)}$	set of d-edges (the distance between nodes is d)
$E^{(\leq d)}$	set of $\{1, 2, \cdots, d\}$-edges (i.e., $\{E^{(1)}, \cdots, E^{(d)}\}$)
$N_u^{(d)}$	set of d-neighbors of node u
$\vec{N}_u^{(d)}$	set of out-going d-neighbors of node u
$G^{(d)} = (V, E^{(\leq d)})$	d-graph of the graph G
$\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$	d-DAG (directed acyclic graph) of the graph G
$C(g; G)$ (or $C(g)$)	count of (d, s)-graphlet g in a graph G

3 NOTATIONS & BASIC CONCEPTS

In this section, we discuss the notations and basic concepts that will be used to describe our concepts (Section 4) and algorithms (Section 5). Refer to Table 1 for the frequently-used notations.

Graphs and distances. A graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. Throughout this work, "graph" refers to an undirected graph unless stated otherwise. The distance $\delta(u, v)$ between two nodes $u, v \in V$ is defined as the length of the shortest path connecting them. Specifically, if u and v are directly connected by an edge (i.e., $\{u, v\} \in E$), the distance between them is 1. If no path exists between two nodes, their distance is considered infinite.

Induced subgraphs. Given a set of nodes $S \subseteq V$, the induced subgraph on S is the subgraph $G_S = (S, E_S)$, where E_S is the set of all edges between nodes in S that are present in the original graph G, i.e., $E_S = \{\{u, v\} \in E : u, v \in S\}$.

Graphlets. A *graphlet* [47, 48] is a subgraph that represents a specific connectivity pattern among a small number of nodes (typically, 3, 4, or 5 nodes). Formally, it is defined as an equivalence class of induced subgraphs under graph isomorphism.[1] The size of a graphlet refers to the number of nodes it contains.

Null models. To accurately characterize real-world graphs, we compare them with random graphs generated by a null model. We employ the configuration model [43] as the null model, which preserves the node degree distributions of the input graph.

4 PROPOSED CONCEPTS

In this section, we propose (d, s)-graphlets, which are tools for analyzing the local structural characteristics of graphs. We first introduce several basic concepts and then formally define (d, s)-graphlets based on them. See Table 1 for frequently-used notations.

4.1 Preliminary Concepts

We begin by defining some basic concepts that are essential for defining (d, s)-graphlets in Section 4.2.

d-edges. Given a graph $G = (V, E)$, we define a d-edge as a pair of nodes whose distance is d. Any pair of nodes $\{u, v\}$ that forms an edge in the graph (i.e., $\{u, v\} \in E$) is referred to as a 1-edge. Node pairs with a distance of $d \geq 2$ form d-edges, which represent *indirect* connections between the nodes. We denote the set of d-edges in G by $E^{(d)} := \{\{u, v\} \in \binom{V}{2} : \delta(u, v) = d\}$, where $\delta(u, v)$ denotes the distance between two nodes u and v. Importantly, the sets $E^{(d)}$'s are pairwise disjoint (i.e., $E^{(d)} \cap E^{(d')} = \emptyset$ for all $d \neq d'$) since $\delta(u, v)$ is uniquely defined for each node pair $\{u, v\}$. We denote the union of all 1-edges through d-edges as $E^{(\leq d)} = \{E^{(1)}, E^{(2)}, \cdots, E^{(d)}\}$, and define the d-extended graph as $G^{(d)} = (V, E^{(\leq d)})$ which includes both direct and indirect connections up to distance d. Finally, a node v is called a d-neighbor of u if $\{u, v\}$ is a d-edge, and we denote the set of d-neighbors of u as $N_u^{(d)}$.

d-induced subgraphs. Given a set of nodes $S \subseteq V$, the d-induced subgraph on S is the subgraph $G_S^{(d)} = (S, E_S^{(\leq d)})$, where $E_S^{(\leq d)}$ consists of all 1-edges, 2-edges, up to d-edges between nodes in S from the graph G. That is, $E_S^{(\leq d)} = \{E_S^{(1)}, E_S^{(2)}, \cdots, E_S^{(d)}\}$, where the edge set $E_S^{(d')}$, which is defined as $\{\{u, v\} \in E^{(d')} : u, v \in S\}$, represents the set of all d'-edges between nodes in S for each $d' \in \{1, 2, \cdots, d\}$ (note that the distance between nodes in $G_S^{(d)}$ is measured in the entire graph G, not in the subgraph $G_S^{(d)}$). Notably, conventional induced subgraphs are 1-induced subgraphs, which only consider direct connections (i.e., 1-edges) between nodes in S. In contrast, d-induced subgraphs generalize this concept by capturing higher-order connectivity patterns beyond direct connections.

d-isomorphism. Given two sets of nodes, S and S', and their d-induced subgraphs $G_S^{(d)} = (S, E_S^{(\leq d)})$ and $G_{S'}^{(d)} = (S', E_{S'}^{(\leq d)})$, they are considered d-isomorphic if there exists a bijection $\phi : S \to S'$ such that for every pair of nodes $\{u, v\} \in \binom{S}{2}$, the following holds:

$$\{u, v\} \in E_S^{(d')} \Leftrightarrow \{\phi(u), \phi(v)\} \in E_{S'}^{(d')}, \quad \forall d' \in \{1, 2, \cdots, d\}.$$

This indicates that the mapping ϕ preserves the structure of all edges up to distance d between the nodes in S and S', meaning the subgraphs are structurally identical with respect to d-edges.

4.2 (d, s)-Graphlets

We now define (d, s)-graphlets, which generalize graphlets by incorporating *relationships beyond direct connections* to describe the connectivity patterns of s nodes.

Definition. A (d, s)-graphlet is a d-isomorphism class of size-s d-induced subgraphs. That is, the d-induced subgraphs $G_S^{(d)}$ and $G_{S'}^{(d)}$ of two sets, S and S', each containing s nodes (i.e., $|S| = |S'| = s$), belong to the same (d, s)-graphlet if they are d-isomorphic. In essence, a (d, s)-graphlet represents an equivalence class of d-induced subgraphs where the local structure, including both direct and indirect connections up to distance d, is identical.

Examples. In Figure 2, we present examples of (d, s)-graphlets. We let $\mathrm{T}^{(d)}$ denote the set of all size-3 (d, s)-graphlets and $\mathrm{Q}^{(d)}$ denote that of size-4 (d, s)-graphlets. There exist 6 $(2, 3)$-graphlets ($\mathrm{T}^{(2)} = \{\mathrm{T}_1^{(2)}, \cdots, \mathrm{T}_6^{(2)}\}$), 13 $(3, 3)$-graphlets ($\mathrm{T}^{(3)} = \{\mathrm{T}_1^{(3)}, \cdots, \mathrm{T}_{13}^{(3)}\}$), and 36 $(2, 4)$-graphlets ($\mathrm{Q}^{(2)} = \{\mathrm{Q}_1^{(2)}, \cdots, \mathrm{Q}_{36}^{(2)}\}$).

Comparison with graphlets. When considering only direct edges (i.e., 1-edges), there are only two types of size-3 graphlets (a triangle and a wedge) and six types of size-4 graphlets (e.g., as a 4-clique or 4-cycle). However, as shown in Figure 2, incorporating indirect connections (e.g., 2-edges and 3-edges) allows for finer distinctions among patterns of 3 or 4 nodes. While increasing the number of nodes in conventional graphlets may provide more insights into graph structure, it also exponentially increases the number of graphlet types and thus requires significantly more complex and computationally expensive counting algorithms. In Section 6, we demonstrate empirically that size-3 and size-4 (d, s)-graphlets more effectively characterize graphs compared to larger graphlets.

4.3 Characteristic Profiles

To summarize the (d, s)-graphlet characteristics of a graph, we use a measure called *characteristic profile* (CP), which is conventionally used in graphlet studies [29, 40, 59, 60, 66]. First, we count the occurrences (i.e., number of instances) of each (d, s)-graphlet, which we simply refer to as the count of each (d, s)-graphlet from now on. Let the occurrence count of the (d, s)-graphlet g in graph G be denoted as $C(g; G)$ (or $C(g)$ for brevity). Then, the significance of a (d, s)-graphlet g is defined as:

$$\mu_g = \frac{C(g; G) - C(g; G_{\text{rand}})}{C(g; G) + C(g; G_{\text{rand}}) + \epsilon},$$

where G_{rand} is a randomized graph of G generated by a null model (see Section 3), and ϵ is a small constant (e.g., $\epsilon = 1$). Based on the significance, the CP of g is computed as the normalized significance:

$$\mathrm{CP}_g = \frac{\mu_g}{\sqrt{\sum_{g' \in \tilde{g}} \mu_{g'}^2}},$$

[1] Two induced subgraphs $G_S = (S, E_S)$ and $G_{S'} = (S', E_{S'})$ are isomorphic if and only if there exists a bijection $\phi : S \to S'$ such that for every pair of nodes $\{u, v\} \in \binom{S}{2}$, the connectivity relationship is preserved, i.e., $\{u, v\} \in E_S \Leftrightarrow \{\phi(u), \phi(v)\} \in E_{S'}$. This implies that the connectivity patterns are identical between the subgraphs on S and S' under the mapping ϕ.

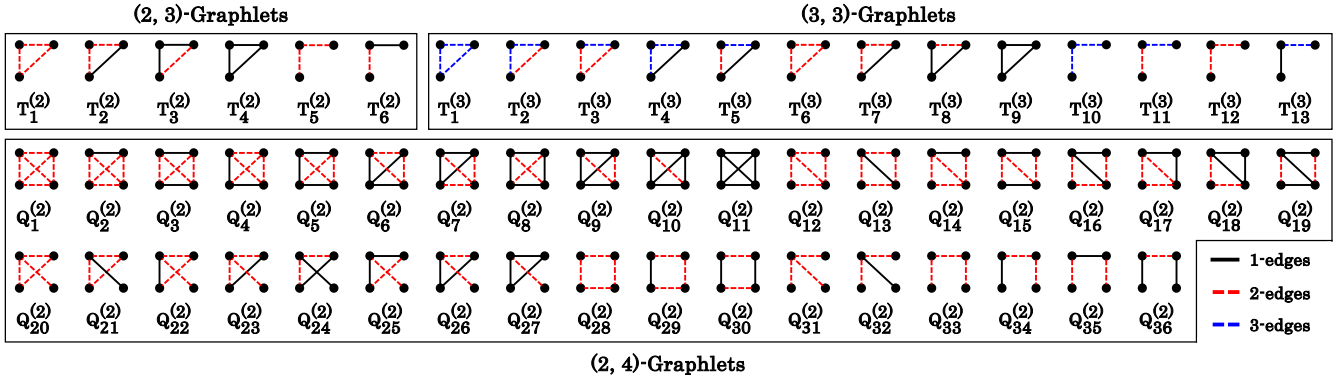


Figure 2: All the $(2,3)$-graphlets ($T_1^{(2)}, \cdots, T_6^{(2)}$), $(3,3)$-graphlets ($T_1^{(3)}, \cdots, T_{13}^{(3)}$), and $(2,4)$-graphlets ($Q_1^{(2)}, \cdots, Q_{36}^{(2)}$). Solid edges represent direct connections (i.e., 1-edges), while dotted edges represent indirect connections (i.e., 2-edges and 3-edges). The 1-edges, 2-edges, and 3-edges are colored in black, red, and blue, respectively.

where $\tilde{g}$ is the set of all considered (d,s)-graphlets (e.g., $T^{(2)}$, $T^{(3)}$, or $Q^{(2)}$). The CP is represented as a vector by concatenating the CP values of each (d,s)-graphlet, which contains the local structural information of the graph.

5 PROPOSED ALGORITHMS

In this section, we present EDGE, our algorithm for the exact counting of (d,s)-graphlets in a given graph. While (d,s)-graphlets are generally defined for arbitrary values of d and s, we focus on three specific configurations: $(d,s) = \{(2,3),(3,3),(2,4)\}$. In Section 6, we empirically demonstrate that these configurations are effective and general enough, compared to conventional graphlets with similar sizes, to uncover non-trivial structural patterns within the graph. We first introduce our method for counting size-3 (d,s)-graphlets (i.e., $(2,3)$- and $(3,3)$-graphlets), followed by our method for counting size-4 (d,s)-graphlets (i.e., $(2,4)$-graphlets).

Remarks. The problem of counting (d,s)-graphlets (particularly for $d \geq 2$), is computationally more challenging than counting graphlets (i.e., for $d = 1$). Specifically, (d,s)-graphlets are defined based on relationships between nodes up to distance d, requiring the exploration of $E^{(2)}, \cdots, E^{(d)}$, where the number of edges grows exponentially as $O(\Delta^d)$, with Δ indicating the maximum node degree. For example, as shown in Table 4, the number of 3-edges is at most 15× greater than that of the original edges (i.e., 1-edges). Moreover, the type of each connection (i.e., d in d-edges) should be accounted for when determining the (d,s)-graphlet of each instance. These unique challenges incur significant bottlenecks for exhaustive enumeration, and thus efficient and specialized algorithms for counting (d,s)-graphlets are demanded.

5.1 Graph Construction

For efficient (d,s)-graphlet counting, EDGE constructs a directed acyclic graph (DAG) that consists of direct connections (i.e., 1-edges) and indirect connections (i.e., d-edges with $d > 1$), as a common preprocessing step.

d-Graph construction. EDGE first constructs indirect connections between non-adjacent nodes. Specifically, it builds additional edge sets $E^{(2)}, \cdots, E^{(d)}$, resulting in a d-graph $G^{(d)} = (V, E^{(\leq d)})$. This process is performed using a breadth-first search (BFS) traversal, as described in Appendix A.1 (refer to Algorithm 3). The time complexity of this step is given in Lemma 1, and the proof is provided in Appendix A.1.

Lemma 1 (Complexity of d-Edge Construction). *The time complexity of constructing the d-graph $G^{(d)}$ for a graph $G = (V, E)$ is $O(|V|\Delta^d)$, where Δ is the maximum degree of the graph.*

d-DAG construction. Once the (undirected) d-graph $G^{(d)}$ is constructed, EDGE builds a d-degree-ordered directed acyclic graph (DAG) of G, referred to as a d-DAG. Specifically, it creates a directed edge (u,v) if $u \prec^{(d)} v$, where $\prec^{(d)}$ represents the degree ordering based on the d-edges, implying $|N_u^{(d)}| \leq |N_v^{(d)}|$. The resulting d-DAG is denoted as $\vec{G} = (V, \vec{E}^{(\leq d)})$, where $\vec{E}^{(\leq d)} = \{\vec{E}^{(1)}, \cdots, \vec{E}^{(d)}\}$, and $\vec{E}^{(d')} = \{(u,v) \in V \times V : \{u,v\} \in E^{(d')} \wedge u \prec^{(d)} v\}$ for each $d' \in \{1, \cdots, d\}$. For a node u, $\vec{N}_u^{(d)}$ denotes the out-going neighbors of u at distance d, i.e., $\vec{N}_u^{(d)} = \{v : (u,v) \in \vec{E}^{(d)}\}$. Importantly, the number of out-going neighbors is typically smaller than the number of neighbors in undirected graphs (i.e., $|\vec{N}_u^{(d)}| \ll |N_u^{(d)}|$), which significantly contributes to improving the scalability of EDGE, as empirically demonstrated in Section 6. For more details, refer to Appendix A.2.

5.2 Size-3 (d,s)-Graphlet Counting

We now describe how EDGE counts size-3 (d,s)-graphlets (i.e., $s = 3$), specifically focusing on $(2,3)$-graphlets ($T^{(2)}$) and $(3,3)$-graphlets ($T^{(3)}$). As described in Algorithm 1, whose sub-algorithms are detailed in Appendix A.3, we categorize size-3 (d,s)-graphlets ($T^{(d)}$) into two groups: *non-deducible* and *deducible* ones as follows:

- **Non-deducible size-3 (d,s)-graphlets** ($\bar{T}^{(d)}$) require explicit enumeration, as their counts cannot be directly inferred. These include the following types of triangles:
  - $\bar{T}^{(2)} = \{T_1^{(2)}, T_2^{(2)}, T_4^{(2)}\}$
  - $\bar{T}^{(3)} = \{T_1^{(3)}, T_2^{(3)}, T_3^{(3)}, T_4^{(3)}, T_6^{(3)}, T_7^{(3)}, T_9^{(3)}\}$
- **Deducible size-3 (d,s)-graphlets** ($\hat{T}^{(d)}$) are those whose counts can be inferred from the graph structure (e.g., node degrees) and the counts of non-deducible (d,s)-graphlets:

Algorithm 1: Counting Size-3 (d,s)-Graphlets. Refer to Appendix A.3 for the definitions of sub-algorithms.

Input: (1) d-Graph $G^{(d)} = (V, E^{(\leq d)})$ of graph G
(2) d-DAG $\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$ of graph G
(3) Maximum considered distance d
Output: The count of each size-3 (d,s)-graphlet $\mathrm{T}_i^{(d)}$: $C(\mathrm{T}_i^{(d)})\ \ \forall i \in \{1, \cdots, |\mathrm{T}^{(d)}|\}$

// Initialization
1 $C(\mathrm{T}_i^{(d)}) \leftarrow 0\ \ \forall i \in \{1, \cdots, |\mathrm{T}^{(d)}|\}$
// Count non-deducible (d,s)-graphlets $\overline{\mathrm{T}}^{(d)}$
2 **for** ***each*** $u \in V$
3 $P_u \leftarrow$ Effective_Neighbor_Pairs$(u, d, \vec{G}^{(d)})$
4 **for** ***each*** $(v, w) \in P_u$
5 $\mathrm{T}_*^{(d)} \leftarrow$ Get_Triangle$\left((u, v, w), d, \vec{G}^{(d)}\right)$
6 $C(\mathrm{T}_*^{(d)}) \leftarrow C(\mathrm{T}_*^{(d)}) + 1$
// Count deducible (d,s)-graphlets $\widehat{\mathrm{T}}^{(d)}$
7 **for** ***each*** $\mathrm{T}_j^{(d)} \in \widehat{\mathrm{T}}^{(d)}$
8 $C(\mathrm{T}_j^{(d)}) \leftarrow$ Comb_Three$\left(\mathrm{T}_j^{(d)}, \{C(\mathrm{T}_i^{(d)})\}_{i=1}^{|\mathrm{T}^{(d)}|}, G^{(d)}\right)$
9 **return** $C(\mathrm{T}_i^{(d)})\ \ \forall i \in \{1, \cdots, |\mathrm{T}^{(d)}|\}$

- $\widehat{\mathrm{T}}^{(2)} = \{\mathrm{T}_3^{(2)}, \mathrm{T}_5^{(2)}, \mathrm{T}_6^{(2)}\}$
- $\widehat{\mathrm{T}}^{(3)} = \{\mathrm{T}_5^{(3)}, \mathrm{T}_8^{(3)}, \mathrm{T}_{10}^{(3)}, \mathrm{T}_{11}^{(3)}, \mathrm{T}_{12}^{(3)}, \mathrm{T}_{13}^{(3)}\}$

As we describe in detail below, we first selectively enumerate each instance of non-deducible (d,s)-graphlets (lines 2 - 6). Afterward, the counts of deducible (d,s)-graphlets can be rapidly computed using specialized combinatorial methods without enumeration (lines 7 - 8). This two-stage approach significantly contributes to the speed of EDGE, as empirically demonstrated in Section 6.

Counting non-deducible (d,s)-graphlets. To count each non-deducible (d,s)-graphlet, EDGE iterates over each node u. It samples a subset of its neighboring pairs to ensure that only instances of non-deducible (d,s)-graphlets are enumerated (line 3). For each *effective* neighboring pair (v,w), it identifies the (d,s)-graphlet of the triangle (u,v,w) based on the distances between the constituent nodes (line 5). The count of the corresponding (d,s)-graphlet is then incremented (line 6).

Counting deducible (d,s)-graphlets. Once EDGE counts the non-deducible (d,s)-graphlets, it efficiently computes the counts of deducible (d,s)-graphlets using combinatorial counting (line 8). For each deducible (d,s)-graphlet, EDGE leverages predefined equations specific to each (d,s)-graphlet, based on (1) the exact count of the non-deducible (d,s)-graphlets and (2) structural information (e.g., node degree), if needed. This deductive approach avoids the need for explicit enumeration for deducible (d,s)-graphlets. For example, $C(\mathrm{T}_5^{(2)})$ can be computed by using the following equation:

$$C(\mathrm{T}_5^{(2)}) = \sum_{u \in V} \binom{|N_u^{(2)}|}{2} - 3C(\mathrm{T}_1^{(2)}) - C(\mathrm{T}_2^{(2)}).$$

The first term counts all cases where the center node of $\mathrm{T}_5^{(2)}$ has neighbors connected by 2-edges on both sides. Since these neighbors may also be connected by an (1- or 2-) edge, the counts of the non-deducible $(2,3)$-graphlets $\mathrm{T}_1^{(2)}$ and $\mathrm{T}_2^{(2)}$ are subtracted. As $\mathrm{T}_1^{(2)}$ can appear at any of the three nodes in a triangle, its count is multiplied by 3 when subtracting. All specific equations for Comb_Three are provided in Appendix A.3 (Algorithm 7).

Complexity analysis. We analyze the time complexity of EDGE for counting size-3 (d,s)-graphlets (Algorithm 1) in Theorem 1.

Theorem 1 (Complexity of Algorithm 1). *The time complexity of EDGE for counting size-3 (d,s)-graphlets is $O(|V|d^4\vec{\Delta}^{2d}\log\vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree of nodes, i.e., $\vec{\Delta} = \max_{u\in V}|\vec{N}_u^{(1)}|$.*

Proof. Refer to Appendix A.3.

Remarks. The time complexity of Algorithm 1 is primarily dominated by the counting of non-deducible (d,s)-graphlets. EDGE achieves this complexity in two ways: (1) It employs d-DAGs, where each node has fewer neighbors compared to d-graphs, reducing redundancy in enumeration, (2) EDGE selectively enumerates only non-deducible (d,s)-graphlets and rapidly counts deducible (d,s)-graphlets afterward. As demonstrated in Section 6, these optimizations lead to a significant speedup of EDGE.

5.3 Size-4 (d,s)-Graphlet Counting

We now describe how EDGE counts size-4 (d,s)-graphlets (i.e., $s = 4$), focusing on $(2,4)$-graphlets ($\mathrm{Q}^{(2)}$), as outlined in Algorithm 2, whose sub-algorithms are detailed in Appendix A.4. For $(2,4)$-graphlets, we categorize the 36 possible configurations (i.e., $\mathrm{Q}_1^{(2)}, \cdots, \mathrm{Q}_{36}^{(2)}$) into *non-deducible*, *semi-deducible*, and *deducible*.

- **Non-deducible $(2,4)$-graphlets** ($\overline{\mathrm{Q}}^{(2)}$), which are all *cliques*, require explicit enumeration to obtain their exact counts:
  - $\overline{\mathrm{Q}}^{(2)} = \{\mathrm{Q}_1^{(2)}, \mathrm{Q}_2^{(2)}, \cdots, \mathrm{Q}_{11}^{(2)}\}$
- **Semi-deducible $(2,4)$-graphlets** ($\widetilde{\mathrm{Q}}^{(2)}$), which are all *cycles*, partially require enumeration, followed by adjustment:
  - $\widetilde{\mathrm{Q}}^{(2)} = \{\mathrm{Q}_{28}^{(2)}, \mathrm{Q}_{29}^{(2)}, \mathrm{Q}_{30}^{(2)}\}$
- **Deducible $(2,4)$-graphlets** ($\widehat{\mathrm{Q}}^{(2)}$) are those whose counts can be rapidly obtained using the counts of non-deducible $(2,4)$-graphlets and the graph structure:
  - $\widehat{\mathrm{Q}}^{(2)} = \{\mathrm{Q}_{12}^{(2)}, \cdots, \mathrm{Q}_{27}^{(2)}, \mathrm{Q}_{31}^{(2)}, \cdots, \mathrm{Q}_{36}^{(2)}\}$

We first count the non-deducible (d,s)-graphlets by enumerating their occurrences on the 2-DAG (lines 3 - 10) and use their counts to compute the counts of the deducible ones through combinatorial methods (lines 17 - 18). For semi-deducible ones, we initially compute the number of their non-induced instances (i.e., instances that induce semi-deducible (d,s)-graphlets) (lines 11 - 16) and adjust their counts accordingly after the enumeration (lines 19 - 20).

Counting non-deducible (d,s)-graphlets. Instead of exhaustively enumerating all size-4 instances on the 2-DAG to obtain the exact count of $(2,4)$-graphlets, we count them by decomposing their structure. Notably, all non-deducible (d,s)-graphlets form cliques, which can be decomposed into two triangles sharing an (1- or 2-) edge, and the remaining two nodes are also connected by an (1- or 2-) edge (see an example in Figure 3). For every edge (u,v) on the 2-DAG, using their common neighbors (line 4), we determine the set of triangle pairs where the remaining nodes are connected (line 5). For each pair of triangles, we first identify the $(2,3)$-graphlet of each triangle (lines 7 - 8) and then determine the $(2,4)$-graphlet

Algorithm 2: $(2,4)$-Graphlets Counting. Refer to Appendix A.4 for the definitions of sub-algorithms.

Input: (1) 2-graph $G^{(2)} = (V, E^{(\leq 2)})$ of graph G
(2) 2-DAG $\vec{G}^{(2)} = (V, \vec{E}^{(\leq 2)})$ of graph G
Output: The count of each size-4 (d,s)-Graphlets $\mathrm{Q}_i^{(2)}$: $C(\mathrm{Q}_i^{(2)})$ $\forall i \in \{1, \cdots, |\mathrm{Q}^{(2)}|\}$

// Initialization
1 $C(\mathrm{Q}_i^{(2)}) \leftarrow 0 \quad \forall i \in \{1, \cdots, |\mathrm{Q}^{(2)}|\}$
2 **for** *each* $u \in V$
// Count non-deducible $\overline{\mathrm{Q}}^{(2)}$
3 **for** *each* $v \in \vec{N}_u^{(1)} \cup \vec{N}_u^{(2)}$
4 $N_{u,v} \leftarrow \left(\vec{N}_u^{(1)} \cup \vec{N}_u^{(2)}\right) \cap \left(\vec{N}_v^{(1)} \cup \vec{N}_v^{(2)}\right)$
5 $\mathcal{T}_{u,v} \leftarrow$ Triangle_Pairs$((u,v), N_{u,v}, \vec{G}^{(2)})$
6 **for** *each* $((u,v,w),(u,v,w')) \in \mathcal{T}_{u,v}$
7 $\mathrm{T}_{\circ}^{(2)} \leftarrow$ Get_Triangle$\left((u,v,w), \vec{G}^{(2)}\right)$
8 $\mathrm{T}_{\bullet}^{(2)} \leftarrow$ Get_Triangle$\left((u,v,w'), \vec{G}^{(2)}\right)$
9 $\mathrm{Q}_{*}^{(2)} \leftarrow$ Get_Clique$\left(\mathrm{T}_{\circ}^{(2)}, \mathrm{T}_{\bullet}^{(2)}, u, v, w, w'\right)$
10 $C(\mathrm{Q}_{*}^{(2)}) \leftarrow C(\mathrm{Q}_{*}^{(2)}) + 1$
// Count semi-deducible $\widetilde{\mathrm{Q}}^{(2)}$
11 **for** *each* $(v,v') \in (\vec{N}_u^{(1)} \cup \vec{N}_u^{(2)}) \times \vec{N}_u^{(2)}$
12 **for** *each* $w \in \{w' \in N_v^{(2)} \cap \left(N_{v'}^{(1)} \cup N_{v'}^{(2)}\right) : u \prec^{(d)} w'\}$
13 $\mathrm{T}_{\triangle}^{(2)} \leftarrow$ Get_Non-Induced_Wedge$\left((u,v,v'), \vec{G}^{(2)}\right)$
14 $\mathrm{T}_{\blacktriangle}^{(2)} \leftarrow$ Get_Non-Induced_Wedge$\left((w,v,v'), \vec{G}^{(2)}\right)$
15 $\mathrm{Q}_{*}^{(2)} \leftarrow$ Get_Non-Induced_Cycle$\left(\mathrm{T}_{\triangle}^{(2)}, \mathrm{T}_{\blacktriangle}^{(2)}\right)$
16 $C(\mathrm{Q}_{*}^{(2)}) \leftarrow C(\mathrm{Q}_{*}^{(2)}) + 1$
// Count deducible (d,s)-graphlets $\widehat{\mathrm{Q}}^{(2)}$
17 **for** *each* $\mathrm{Q}_j^{(2)} \in \widehat{\mathrm{Q}}^{(2)}$
18 $C(\mathrm{Q}_j^{(2)}) \leftarrow$ Comb_Four$\left(\mathrm{Q}_j^{(2)}, \{C(\mathrm{Q}_i^{(2)})\}_{i=1}^{|\mathrm{Q}^{(2)}|}, G^{(2)}\right)$
// Adjust counts of semi-deducible (d,s)-graphlets $\widetilde{\mathrm{Q}}^{(2)}$
19 **for** *each* $\mathrm{Q}_j^{(2)} \in \widetilde{\mathrm{Q}}^{(2)}$
20 $C(\mathrm{Q}_j^{(2)}) \leftarrow$ Comb_Four$\left(\mathrm{Q}_j^{(2)}, \{C(\mathrm{Q}_i^{(2)})\}_{i=1}^{|\mathrm{Q}^{(2)}|}\right)$
21 **return** $C(\overline{\mathrm{Q}}_i^{(2)}) \quad \forall i \in \{1, \cdots, |\overline{\mathrm{Q}}^{(2)}|\}$

based on the combination of the two $(2,3)$-graphlets (line 9). The count of the identified $(2,4)$-graphlet is then incremented (line 10).

Counting semi-deducible (d,s)-graphlets. For semi-deducible $(2,4)$-graphlets, which are cycles, we adopt a two-step approach. Note that a cycle is composed of two wedges that share two end nodes (see an example in Figure 3). Based on this structure, given two (dis)connected nodes (v,v'), we first enumerate pairs of (non-)induced wedges (which can be either wedges or triangles) on the 2-graph using their common neighbors. Next, we identify the $(2,3)$-graphlet of each non-induced wedge (lines 13 - 14). Based on their combination, we determine the $(2,4)$-graphlet of the (non-)induced cycle (which can be cycles, chordalcycles, or cliques) (line 15) and increment the count (line 16). Once the enumeration finishes, we adjust the counts by subtracting the counts of $(2,4)$-graphlets that are not cycles, ensuring the correct count of semi-deducible (d,s)-graphlets (lines 19 - 20).

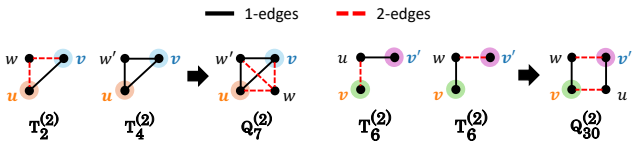


Figure 3: Examples of: (left) two triangles forming a clique, when w and w' are connected by a 2-edge, and (right) two wedges forming a cycle, when u and w are disconnected, as well as v and v'.

Counting deducible (d,s)-graphlets. The counts of deducible $(2,4)$-graphlets can be rapidly computed from the counts of the non-deducible $(2,4)$-graphlets (lines 17 - 18). The equations of Comb_Four are provided in Algorithm 13 of Appendix A.4.

Complexity analysis. We analyze the time complexity of EDGE for counting $(2,4)$-graphlets (Algorithm 2) in Theorem 2.

Theorem 2 (Complexity of Algorithm 2). *The time complexity of EDGE for counting $(2,4)$-graphlets is $(|V|\vec{\Delta}^4\Delta^2 \log \vec{\Delta})$, where Δ and $\vec{\Delta}$ denote the maximum undirected degree and the maximum out-degree, respectively, i.e., $\Delta = \max_{u \in V} |N_u^{(1)}|$ and $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

Proof. Refer to Appendix A.4.

6 EXPERIMENTS

We share our empirical analysis using (d,s)-graphlets and its counting algorithm, EDGE. We aim to answer the following questions:

- **Q1. Graph characterization:** How effective are (d,s)-graphlets in distinguishing and clustering graphs across different domains?
- **Q2. Real-world discoveries:** What insights do (d,s)-graphlets provide that cannot be uncovered by conventional graphlets?
- **Q3. Speed and scalability:** How fast and scalable is EDGE? Do d-DAG and deduced counting contribute to its efficiency?

6.1 Experimental Settings

We report the settings where the experiments were performed.

Datasets. We used 13 real-world graphs from five different domains: collaboration [3, 13, 24, 32], web [11, 12], social-Facebook [57, 58], tags [7], and road [34]. We present basic statistics of the datasets with further details in Appendix B.1.

Implementations. We implemented EDGE in C++. EDGE supports multi-threading, and we set the number of threads to 6. To count size-4 and size-5 graphlets, we used open-source C++ implementations of recent exact counting methods, PGD [44] and ESCAPE [22]. We ran PGD with 6 threads, while ESCAPE does not support multi-threading.

Machines. We performed all experiments on a machine with an Intel i9-10900K CPU and 64GB memory.

6.2 Q1. Graph Characterization

We analyzed the characterization power of (d,s)-graphlets. Specifically, we computed the characteristic profiles (CPs; see Section 4.3) for each graph using counts of $(2,3)$, $(2,4)$, and $(3,3)$-graphlets. As shown in Figure 4, graphs within the same domain exhibit highly

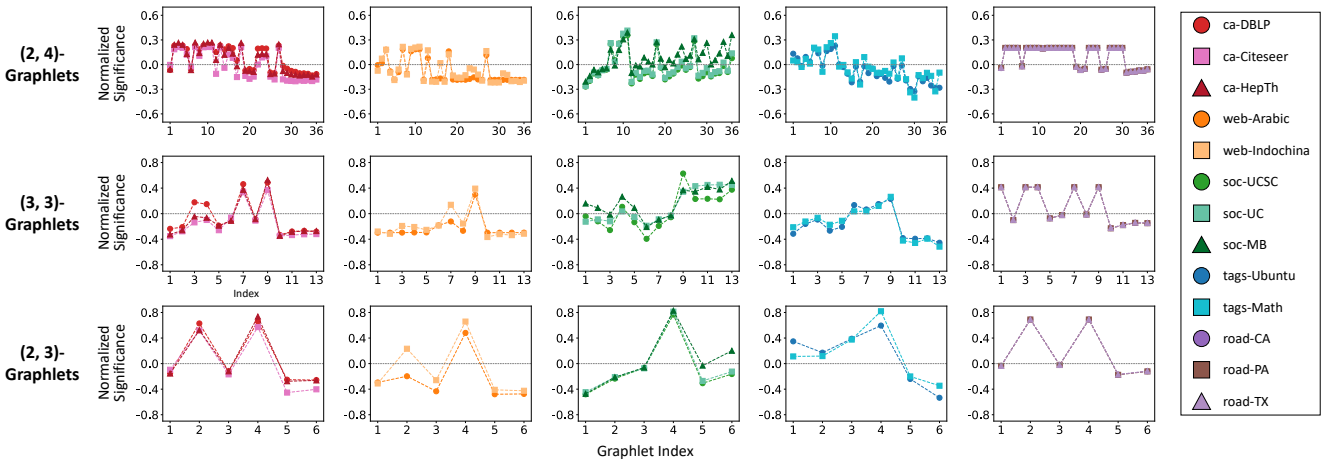


Figure 4: Graphs from the same domain exhibit similar CPs derived from the counts of $(2,4)$-, $(3,3)$-, and $(2,3)$-graphlets.

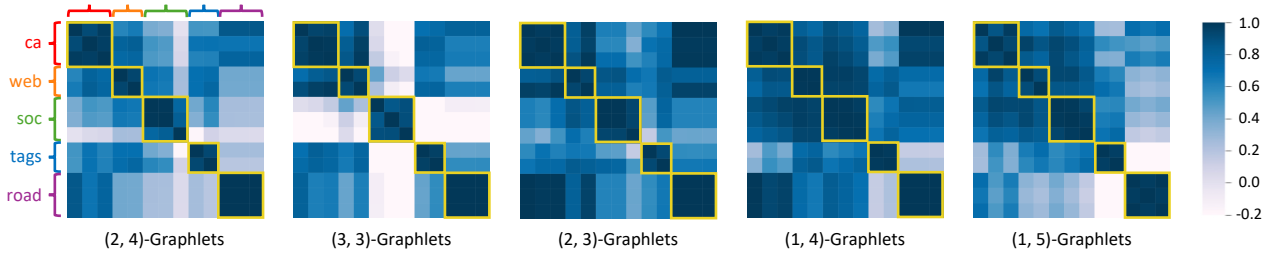


Figure 5: The domains of the graphs are effectively distinguished by CPs derived from the counts of (d,s)-graphlets. Specifically, $(2,4)$-, $(3,3)$-, and $(2,3)$-graphlets, which account for generalized distances, provide a clearer distinction of graphs across domains compared to conventional graphlets that consider only direct connections (e.g., $(1,4)$- and $(1,5)$-graphlets, which represent size-4 and size-5 graphlets, respectively). For numerical comparisons, refer to Table 2.

Table 2: (d,s)-graphlets exhibit larger correlation gaps between graphs within the same domain and across domains, as well as superior clustering performance compared to the original graphlets.

	(d,s)	Correlation Gap (Within - Across)	Clustering F1	NMI	SH
Original	$(1,3)$	0.000	0.467	0.455	0.266
	$(1,4)$	0.252	0.670	0.772	0.539
	$(1,5)$	0.440	0.920	0.908	0.680
Ours	$(2,3)$	0.253	0.667	0.856	0.585
	$(3,3)$	**0.667**	**1.000**	**1.000**	**0.797**
	$(2,4)$	0.473	**1.000**	**1.000**	0.795

similar CPs for all (d,s)-graphlets, while CPs are clearly distinguished across different domains.

We computed the correlations between CPs of different graphs, as shown in Figure 5. Notably, $(2,4)$-, $(3,3)$-, and $(2,3)$-graphlets provide clearer distinctions between graphs across domains, compared to the conventional graphlets (i.e., $(1,4)$- and $(1,5)$-graphlets). Specifically, as shown in Table 2, the correlation gap (i.e., the difference between average correlations within and across domains) is largest for $(3,3)$-graphlets, followed by $(2,4)$-graphlets, even though they use fewer nodes per graphlet than $(1,5)$-graphlets. These large gaps demonstrate the effectiveness of incorporating multi-hop distances for graphlets in graph characterization.

We further evaluated the clustering of graphs using the CPs as input features, specifically applying spectral clustering. As shown in Table 2, (d,s)-graphlets lead to higher clustering performance in terms of F1 score, NMI, and Silhouette score. This further validates the effectiveness of (d,s)-graphlets in graph characterization.

Extra results. (d,s)-graphlets can be effective for various machine learning tasks, as demonstrated for link prediction and graph classification in Appendix B.4.

6.3 Q2. Real-World Discoveries

We conduct case studies on real-world graphs that support the expressiveness and significance of (d,s)-graphlets in graph characterization.

Expressiveness. In Figure 6, we demonstrate that incorporating higher-order distances between nodes allows for finer differentiation of local structures. As shown on the left side of the figure, the conventional 3-path size-4 graphlet is divided into two cases, $Q_5^{(2)}$ and $Q_{19}^{(2)}$, when considering distances up to 2; and the 2D scatter plot shows that the proportions of these finer local structures lead to clearer graph characterization. For example, graphs in the collaboration domain, indistinguishable when considering only the 3-path graphlet, become distinguishable due to a lower

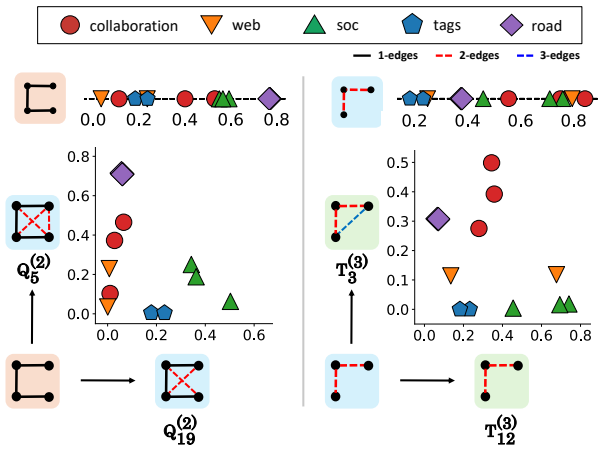


Figure 6: Indirect connections allow for finer differentiation of local structures. (Top) The ratio of a graphlet relative to the total number of instances. (Bottom) The graphlet can be decomposed into finer structures when considering indirect connections beyond those in the graphlet. With the finer structures, the domains of graphs are better distinguished.

proportion of $Q_{19}^{(2)}$. Similarly, as shown on the right side of the figure, a $(2,3)$-graphlet $T_5^{(2)}$ is generalized to two $(3,3)$-graphlets, $T_3^{(2)}$ and $T_{12}^{(2)}$, when considering distances up to 3. This further improves domain distinction and thus graph characterization.

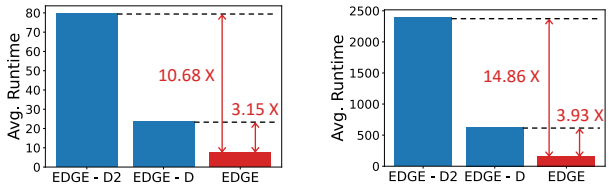
Significance. To assess the significance of (d,s)-graphlets in graph characterization, we use a scoring function that measures each (d,s)-graphlet's contribution to distinguishing graphs by domain, as in [30]. Based on these scores, we ranked the (d,s)-graphlets and retrieved the top 5 from each graph. Notably, 59 out of the 65 retrieved (d,s)-graphlets were those that cannot be described by conventional graphlets, demonstrating the effectiveness of (d,s)-graphlets in capturing more comprehensive structural patterns. For more details, refer to Appendix B.2.

6.4 Q3. Speed and Scalability

We evaluate the speed and scalability of EDGE by comparing it with its variants and existing counting methods.

Effects of EDGE's components. We first analyze the effects of EDGE's design components: (1) deducible counting for deducible (d,s)-graphlets (i.e., $\widetilde{T}^{(d)}$ and $\widetilde{Q}^{(d)}$) and (2) using d-DAG for counting non- and semi-deducible (d,s)-graphlets. We evaluate two variants: EDGE-D2, which removes both (1) and (2), and EDGE-D, which removes only (1). As shown in Figure 7, EDGE is significantly faster than these variants, achieving up to 14.86× and 3.93× speed-ups over EDGE-D2 and EDGE-D, respectively. These results demonstrate the effectiveness of EDGE's design choices in avoiding unnecessary enumeration and using d-DAGs to reduce redundancy. For more details, see Appendix B.3.

Comparison to graphlet counting methods. We compare the counting times of PGD and ESCAPE for $(1,4)$-graphlets with EDGE for $(2,4)$-graphlets. As shown in Figure 8, EDGE has a higher total runtime than PGD and ESCAPE due to the additional edges connecting indirect nodes. However, when comparing running time



(a) $(2,3)$**-graphlet** **(b)** $(3,3)$**-graphlet**

Figure 7: EDGE is faster than its variants: (1) EDGE-D2, which lacks deducible counting and d-DAG, and (2) EDGE-D, which lacks d-DAG. This demonstrates the effectiveness of EDGE's design choices for fast (d,s)-graphlet counting.

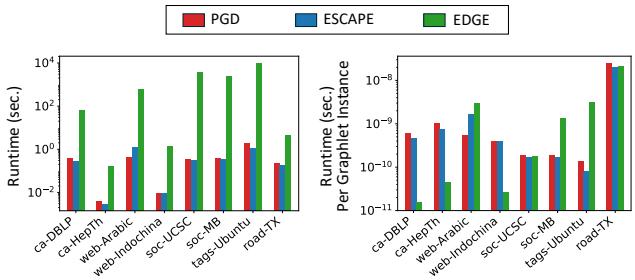


Figure 8: Comparison between EDGE (for counting $(2,4)$-graphlets) and PGD & ESCAPE (for counting $(1,4)$-graphlets). (Left) In total runtime, EDGE is slower than PGD and ESCAPE due to the extra indirect connections. (Right) In runtime per instance, EDGE is competitive and even faster in some cases.

per graphlet, EDGE is competitive, and in some cases, even faster than the baselines. For more details, see Appendix B.3.

7 CONCLUSIONS

We present (d,s)-graphlets, distance-generalized graphlets for enhanced graph characterization. Our contributions are as follows:

- **Concept:** We introduce (d,s)-graphlets, distance-generalized graphlets that enable a more detailed analysis of local structures.
- **Algorithm:** We introduce EDGE, an optimized counting algorithm for (d,s)-graphlets that reduces unnecessary enumeration through specialized combinatorial techniques.
- **Experiments:** Our experiments across 13 real-world datasets demonstrate the effectiveness of (d,s)-graphlets in graph characterization and the efficiency of EDGE.

Our code and datasets are available at **https://github.com/thisis05/EDGE** for reproducibility. Future research directions include accelerating the counting of (d,s)-graphlets through approximation.

Acknowledgements This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00457882, AI Research Hub Project, 50%) (No. 2022-0-00871 / RS-2022-II220871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration, 40%) (RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST), 10%).

References

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *ICDM*.

[2] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, Nick G Duffield, and Theodore L Willke. 2017. Graphlet decomposition: Framework, algorithms, and applications. *Knowledge and Information Systems* 50 (2017), 689–722.

[3] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. 2013. *Graph partitioning and graph clustering*. Vol. 588. American Mathematical Society Providence, RI.

[4] Sasitharan Balasubramaniam et al. 2013. Multi-hop conjugation based bacteria nanonetworks. *IEEE Transactions on nanobioscience* 12, 1 (2013), 47–59.

[5] Balabhaskar Balasundaram, Sergiy Butenko, and Svyatoslav Trukhanov. 2005. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization* 10 (2005), 23–39.

[6] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data* 4, 3 (2010), 1–28.

[7] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[8] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[9] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*.

[10] Hanjo D Boekhout, Walter A Kosters, and Frank W Takes. 2019. Efficiently counting complex multilayer temporal motifs in large-scale networks. *Computational Social Networks* 6, 1 (2019), 8.

[11] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. UbiCrawler: A Scalable Fully Distributed Web Crawler. *Software: Practice & Experience* 34, 8 (2004), 711–726.

[12] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*.

[13] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *WWW*.

[14] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *SIGMOD*.

[15] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 1 (2022), 657–668.

[16] Hao Chen, Zhong Huang, Yue Xu, Zengde Deng, Feiran Huang, Peng He, and Zhoujun Li. 2022. Neighbor enhanced graph convolutional networks for node classification and recommendation. *Knowledge-based Systems* 246 (2022), 108594.

[17] Lina Chen, Xiaoli Qu, Mushui Cao, Yanyan Zhou, Wan Li, Binhua Liang, Weiguo Li, Weiming He, Chenchen Feng, Xu Jia, et al. 2013. Identification of breast cancer patients based on human signaling network motifs. *Scientific reports* 3, 1 (2013), 3368.

[18] Xuexin Chen, Ruichu Cai, Yuan Fang, Min Wu, Zijian Li, and Zhifeng Hao. 2023. Motif graph neural network. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[19] Hong Cheng, Xifeng Yan, and Jiawei Han. 2014. Mining graph patterns. *Frequent pattern mining* (2014), 307–338.

[20] Qiangqiang Dai, Rong-Hua Li, Lu Qin, Guoren Wang, Weihua Yang, Zhiwei Zhang, and Ye Yuan. 2021. Scaling up distance-generalized core decomposition. In *CIKM*.

[21] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. 2019. Cognitive graph for multi-hop reading comprehension at scale. *arXiv preprint arXiv:1905.05460* (2019).

[22] ESCAPE. 2016. Efficient Subgraph Counting Algorithmic Package (ESCAPE) library. https://bitbucket.org/seshadhri/escape.

[23] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *NeurIPS*.

[24] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *ALENEX*.

[25] Shawn Gu, John Johnson, Fazle E Faisal, and Tijana Milenković. 2018. From homogeneous to heterogeneous network alignment via colored graphlets. *Scientific reports* 8, 1 (2018), 12524.

[26] Christopher R Harshaw, Robert A Bridges, Michael D Iannacone, Joel W Reed, and John R Goodall. 2016. Graphprints: Towards a graph analytic method for network anomaly detection. In *CISRC*.

[27] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.

[28] Hong Huang, Jie Tang, Sen Wu, Lu Liu, and Xiaoming Fu. 2014. Mining triadic closure patterns in social networks. In *WWW*.

[29] Geon Lee, Jihoon Ko, and Kijung Shin. 2020. Hypergraph motifs: concepts, algorithms, and discoveries. *PVLDB* 13, 12 (2020), 2256–2269.

[30] Geon Lee, Seokbum Yoon, Jihoon Ko, Hyunju Kim, and Kijung Shin. 2024. Hypergraph motifs and their extensions beyond binary. *The VLDB Journal* 33, 3 (2024), 625–665.

[31] John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. 2019. Graph convolutional networks with motif-based attention. In *CIKM*.

[32] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data* 1, 1 (2007), 2–es.

[33] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[34] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[35] Pei-Zhen Li, Ling Huang, Chang-Dong Wang, and Jian-Huang Lai. 2019. EdMot: An edge enhancement approach for motif-aware community detection. In *KDD*.

[36] Qing Liu, Xuliang Zhu, Xin Huang, and Jianliang Xu. 2021. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *PVLDB* 14, 9 (2021), 1531–1543.

[37] R Duncan Luce. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15, 2 (1950), 169–190.

[38] Dror Marcus and Yuval Shavitt. 2010. Efficient counting of network motifs. In *ICDCS Workshops*.

[39] Brendan D McKay. 2007. Nauty user's guide (version 2.4). *Computer Science Dept., Australian National University* (2007), 225–239.

[40] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. 2004. Superfamilies of evolved and designed networks. *Science* 303, 5663 (2004), 1538–1542.

[41] 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.

[42] Robert J Mokken et al. 1979. Cliques, clubs and clans. *Quality & Quantity* 13, 2 (1979), 161–173.

[43] Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM Rev.* 45, 2 (2003), 167–256.

[44] PGD. 2015. Parallel Parameterized Graphlet Decomposition (PGD) library. https://github.com/nkahmed/PGD.

[45] Yinhua Piao, Sangseon Lee, Dohoon Lee, and Sun Kim. 2022. Sparse structure learning via graph neural networks for inductive document classification. In *AAAI*.

[46] Ali Pinar, Comandur Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *WWW*.

[47] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[48] Natasa Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.

[49] Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. 2019. Dynamically fused graph network for multi-hop reasoning. In *ACL*.

[50] Yuanfang Ren, Aisharjya Sarkar, Ahmet Ay, Alin Dobra, and Tamer Kahveci. 2019. Finding conserved patterns in multilayer networks. In *BCB*.

[51] 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. *Comput. Surveys* 54, 2 (2021), 1–36.

[52] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *AAAI*.

[53] Ryan A Rossi, Nesreen K Ahmed, Aldo Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyee Koh. 2020. Heterogeneous graphlets. *ACM Transactions on Knowledge Discovery from Data* 15, 1 (2020), 1–43.

[54] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS*.

[55] Shinji Tajima, Ren Sugihara, Ryota Kitahara, and Masayuki Karasuyama. 2024. Learning Attributed Graphlets: Predictive Graph Mining by Graphlets with Trainable Attribute. In *KDD*.

[56] Nikolaj Tatti. 2023. Fast computation of distance-generalized cores using sampling. *Knowledge and Information Systems* 65, 6 (2023), 2429–2453.

[57] Amanda L Traud, Eric D Kelsic, Peter J Mucha, and Mason A Porter. 2011. Comparing Community Structure to Characteristics in Online Collegiate Social Networks. *SIAM Rev.* 53, 3 (2011), 526–543.

[58] Amanda L Traud, Peter J Mucha, and Mason A Porter. 2012. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications* 391, 16 (2012), 4165–4180.

[59] Kun Tu, Jian Li, Don Towsley, Dave Braines, and Liam D Turner. 2019. gl2vec: Learning feature representation using graphlets for directed networks. In *ASONAM*.

[60] Katherine Van Koevering, Austin Benson, and Jon Kleinberg. 2021. Random graphs with prescribed k-core sequences: A new null model for network analysis. In *WWW*.
[61] Virginia Vassilevska and Ryan Williams. 2009. Finding, minimizing, and counting weighted subgraphs. In *STOC*.
[62] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. 2020. Multi-hop attention graph neural network. *arXiv preprint arXiv:2009.14332* (2020).
[63] Sebastian Wernicke. 2005. A faster algorithm for detecting network motifs. In *WABI*.
[64] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.
[65] Qinghua Wu and Jin-Kao Hao. 2015. A review on algorithms for maximum clique problems. *European Journal of Operational Research* 242, 3 (2015), 693–709.
[66] Wen-Jie Xie, Ming-Xia Li, Zhi-Qiang Jiang, and Wei-Xing Zhou. 2014. Triadic motifs in the dependence networks of virtual societies. *Scientific Reports* 4, 1 (2014), 5244.
[67] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.
[68] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in neural information processing systems* 33 (2020), 5812–5823.
[69] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. 2018. ANRL: attributed network representation learning via deep neural networks.. In *IJCAI*.
[70] Junyou Zhu, Chunyu Wang, Chao Gao, Fan Zhang, Zhen Wang, and Xuelong Li. 2021. Community detection in graph: An embedding method. *IEEE Transactions on Network Science and Engineering* 9, 2 (2021), 689–702.

A ALGORITHMIC DETAILS

In this section, we provide a more detailed explanation of the counting algorithms introduced in Section 5.

A.1 d-Graph Construction

To count the instances of (d, s)-graphlets, we construct $E^{(\leq d)} = \{E^{(1)}, E^{(2)}, \cdots, E^{(d)}\}$ as a preprocessing step. Algorithm 3 describes the process of constructing d-edges (i.e., $E^{(d)}$) from the input graph. For each node $u \in V$, we utilize the BFS function to identify all nodes that are exactly d-hops away from u (line 3). Then, these nodes are added as edges in $E^{(d)}$ (line 6) while avoiding duplicates. The time complexity of this preprocessing step is given in Lemma 1, and we provide its proof as follows:

Proof of Lemma 1. For each node $u \in V$, the algorithm performs a BFS up to d-hops, and thus the total number of nodes explored up to d-hops is $O(\Delta + \Delta^2 + \cdots + \Delta^d) = O(\Delta^d)$. Thus, the time complexity for iterating over all nodes $u \in V$ is $O(|V|\Delta^d)$.

A.2 Effects of d-DAG

In this section, we assess the effectiveness of using d-DAGs instead of (undirected) d-graphs. As shown in Table 3, both the average and maximum degrees of nodes with respect to 1-edges, 2-edges, and 3-edges are significantly smaller in d-DAGs compared to d-graphs, i.e., $d_{avg} \gg \vec{d}_{avg}$ and $\Delta \gg \vec{\Delta}$. This reduction dramatically decreases the number of enumerations required for counting non-deducible (d, s)-graphlets. We empirically demonstrate the effectiveness of employing d-DAGs in Section 6.

A.3 Details of Algorithm 1

In this subsection, we provide the details of sub-algorithms used in Algorithm 1. Then, we provide the proof of Theorem 1 regarding the complexity of Algorithm 1.

Retrieve_Distance (Algorithm 4). Given a pair of nodes (u, v) and the d-DAG $\vec{G}^{(d)}$, which considers up to d-edges, this function computes the distance between u and v. Equivalently, it determines in which of $\vec{E}^{(\leq d)} = \{\vec{E}^{(1)}, \cdots, \vec{E}^{(d)}\}$ the edge (u, v) is contained. If (u, v) is not included in any of the $\vec{E}^{(1)}, \cdots, \vec{E}^{(d)}$, the distance is considered to be ∞. Since the d-DAG is d-degree-ordered, a directed edge $u \to v$ (or $v \to u$) exists if $u \prec^{(d)} v$ (or $v \prec^{(d)} u$). For each distance $d' \in \{1, \cdots, d\}$, it checks whether v is an out-going neighbor of u (i.e., $v \in \vec{N}_u^{(d')}$) if $u \prec^{(d)} v$, and vice versa. If the neighbor is found, the function returns the corresponding distance d'. The time complexity of Algorithm 4 is provided in Lemma 2.

Algorithm 3: d-Edge Construction (Preprocess)

Input: (1) Input graph $G = (V, E = E^{(1)})$
(2) Maximum distance considered d
Output: Set of d-edges $E^{(d)}$

1 $E^{(d)} \leftarrow \emptyset$
2 **for** ***each*** $u \in V$
// Get all nodes at exactly d-hops from u
3 $S^{(d)} \leftarrow \text{BFS}(u, d, G)$
4 **for** ***each*** $v \in S^{(d)}$
5 **if** $u \prec v$
6 $E^{(d)} \leftarrow E^{(d)} \cup \{(u, v)\}$
7 **return** $E^{(d)}$

Table 3: Degree statistics for various datasets. Each value represents the degree characteristics of 1-edge, 2-edge, and 3-edge types. Δ denotes the maximum degree, $\vec{\Delta}$ denotes the out-going maximum degree, and d_{avg} and $\vec{d}_{avg}$ denote the average degree and the out-going average degree, respectively.

Datasets	Edge type	d_{avg}	Δ	$\vec{d}_{avg}$	$\vec{\Delta}$
ca-DBLP	1-edge	3	343	2	113
	2-edge	39	5.15K	20	378
	3-edge	482	42.9K	241	3.10K
ca-Citeseer	1-edge	3	1.37K	2	206
	2-edge	32	5.45K	16	1.36K
	3-edge	221	24.4K	111	1.91K
web-Arabic	1-edge	10	1.10K	5	164
	2-edge	18	2.66K	9	1.10K
	3-edge	91	15.8K	46	1.13K
web-Indochina	1-edge	4	199	2	83
	2-edge	37	2.01K	19	194
	3-edge	325	3.92K	163	1.47K
soc-UCSC	1-edge	24	454	12	82
	2-edge	799	5.64K	400	1.46K
	3-edge	2.76K	6.91K	1.38K	4.51K
soc-UC	1-edge	22	660	11	99
	2-edge	690	4.90K	345	1.35K
	3-edge	1.97K	5.12K	987	3.19K

Lemma 2 (Complexity of Algorithm 4). *The time complexity of retrieving the distance between two nodes is $O(d^2 \log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

Proof. Without loss of generality, let $u \prec^{(d)} v$. For each distance $d' \in \{1, \cdots, d\}$, we check v is the out-going d'-neighbor of u, i.e., $v \in \vec{N}_u^{(d')}$. Assuming that $\vec{N}_u^{(d')}$ is implemented as a

Algorithm 4: RETRIEVE_DISTANCE

Input: (1) A pair of nodes u and v
(2) d-DAG $\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$ of graph G
(3) Maximum considered distance d
Output: $\delta(u, v)$ (distance between u and v)

for ***each*** $d' \in \{1, \cdots, d\}$
  if $u \prec^{(d)} v$
    if $v \in \vec{N}_u^{(d')}$ ▷ Binary Search
      return d'
  else
    if $u \in \vec{N}_v^{(d')}$ ▷ Binary Search
      return d'
return ∞

Algorithm 5: EFFECTIVE_NEIGHBOR_PAIRS

Input: (1) Node u
(2) Maximum considered distance d
(3) d-DAG $\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$ of graph G
Output: Effective node pairs set P_u

$P_u \leftarrow \emptyset$
for ***each*** $(d_i, d_j) \in \{(d_i', d_j') : 1 \leq d_i' \leq d_j' \leq d\}$
  for ***each*** $(v, w) \in \{(v', w') \in \vec{N}_u^{(d_i)} \times \vec{N}_u^{(d_j)} : v' \prec^{(d)} w'\}$
    if $w \in \vec{N}_v^{(d_k)}\ \exists d_k \in \{1, \cdots, d\} \setminus \{d_i + d_j, |d_i - d_j|\}$
      $P_u \leftarrow P_u \cup \{(v, w)\}$
return P_u

sorted list, we employ a binary search with a time complexity of $O(\log |\vec{N}_u^{(d')}|)$. In the worst case, we search through all distances $d' \in \{1, \cdots, d\}$, yielding a total time complexity of $O(\log |\vec{N}_u^{(1)}| + \cdots + \log |\vec{N}_u^{(d)}|)$. Let $\vec{\Delta}$ be the maximum out-degree, i.e., $\vec{\Delta} = \max_{v \in V} |\vec{N}_v^{(1)}|$. The time complexity of Algorithm 4 is thus derived as $O(\log \vec{\Delta} + \cdots \log \vec{\Delta}^d) = O(d^2 \log \vec{\Delta})$.

EFFECTIVE_NEIGHBOR_PAIRS (Algorithm 5). This function determines the *effective* neighboring pairs from all possible pairs of neighbors. Specifically, given a node u and the maximum considered distance d, the set of all neighbors of node u up to distance d is $\bigcup_{d' \in \{1, \cdots, d\}} \vec{N}_u^{(d')}$. Among all pairs of neighbors (i.e., $\binom{\bigcup_{d' \in \{1, \cdots, d\}} \vec{N}_u^{(d')}}{2}$), we only consider those that are connected and thus form a triangle with u. In addition, we exclude the neighbor pairs whose triangle forms deducible (d, s)-graphlets, as these can be efficiently counted without enumeration. This reduction in the set of neighbor pairs significantly and speeds up EDGE. The time complexity of Algorithm 5 is provided in Lemma 3.

LEMMA 3 (COMPLEXITY OF ALGORITHM 5). *The time complexity of retrieving effective neighboring pairs of a node is $O(d^4 \vec{\Delta}^{2d} \log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

PROOF. For a given node u and the maximum considered distance d, we consider u's out-going neighbors at all $O(d^2)$ combinations of distances (d_i, d_j). For each pair of distances, we examine all pairs (v, w) (where $v \prec^{(d)} w$) of d_i-neighbors and d_j-neighbors, which results is $O(|\vec{N}_u^{d_i}| \cdot |\vec{N}_u^{d_j}|) = O(\vec{\Delta}^{2d})$ pairs, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\max_{u \in V} |\vec{N}_u^{(1)}|$. Next, we check whether the two neighbors are connected, which can be done by performing a binary search for w from $\vec{N}_v^{(d_k)}\ \forall d_k \in \{1, \cdots, d_k\}$ which takes $O(\log |\vec{N}_v^{(1)}| + \cdots + \log |\vec{N}_v^{(d)}|) = (d^2 \log \vec{\Delta})$ time. Thus, the overall time complexity is $O(d^4 \vec{\Delta}^{2d} \log \vec{\Delta})$.

GET_TRIANGLE (Algorithm 6). Given three nodes (u, v, w) which consists a triangle, and the maximum distance considered d, this function returns the corresponding $(d, 3)$-graphlet of the triangle. First, it retrieves the distances for all pairs of edges. Then, based on these three distances, it identifies the $(d, 3)$-graphlet. The time complexity of Algorithm 6 is provided in Lemma 4.

LEMMA 4 (COMPLEXITY OF ALGORITHM 6). *The time complexity of identifying $(d, 3)$-graphlet of a triangle is $O(d^2 \log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

PROOF. The time complexity for retrieving the distances of three pairs of nodes is $O(d^2 \log \vec{\Delta})$ (from Lemma 2). Once the distances are obtained, the corresponding $(d, 3)$-graphlet can be identified in $O(1)$ time. Thus, the overall time complexity is $O(d^2 \log \vec{\Delta})$.

COMB_THREE (Algorithm 7). This function is for computing the counts of deducible $(d, 3)$-graphlets ($\widehat{\mathrm{T}}^{(d)}$). Based on the counts of non-deducible $(d, 3)$-graphlets ($\overline{\mathrm{T}}^{(d)}$), it computes the counts of the target deducible $(d, 3)$-graphlets. The time complexity of Algorithm 7 is provided in Lemma 5.

LEMMA 5 (COMPLEXITY OF ALGORITHM 7). *The time complexity of computing the count of the given deducible $(d, 3)$-graphlet is $O(|V|)$.*

PROOF. To compute the count of the target deducible $(d, 3)$-graphlet, degree-based computations are required, e.g., $\sum_{u \in V} \binom{|N_u^{(1)}|}{2}$ to compute $C(\mathrm{T}_3^{(2)})$. This takes $O(|V|)$ time.

Proof of Theorem 1. Below, we provide the proof for Theorem 1.

PROOF. The time complexity of Algorithm 1 is determined by two main operations: (1) counting non-deducible $(d, 3)$-graphlets and (2) counting deducible $(d, 3)$-graphlets.

- To count **non-deducible** $(d, 3)$-graphlets, we iterate over each node $u \in V$ and obtain its effective neighbor pairs using EFFECTIVE_NEIGHBOR_PAIRS, which takes $O(d^4 \vec{\Delta}^{2d} \log \Delta)$ time (refer to Lemma 3). For each effective neighbor pair (v, w), we identifies the $(d, 3)$-graphlet of the triangle (u, v, w) using GET_TRIANGLE, which takes $O(d^2 \log \vec{\Delta})$ time (Lemma 4). In practice, we (1) check the connectivity between v and w takes $O(d^2 \log \vec{\Delta})$ time, and (2) subsequently identify the $(d, 3)$-graphlet if it is connected which takes $O(|V| d^2 \log \vec{\Delta})$ time as well. Thus, the total time complexity of counting non-deducible $(d, 3)$-graphlets is $O(|V| d^4 \vec{\Delta}^{2d} \log \vec{\Delta})$.
- To count **deducible** $(d, 3)$-graphlets, we use COMB_THREE which takes $O(|V|)$ time (Lemma 5).

Since counting non-deducible (d, s)-graphlets dominates the entire complexity, the total time complexity of Algorithm 1 is $O(|V| d^4 \vec{\Delta}^{2d} \log \vec{\Delta})$.

Algorithm 6: GET_TRIANGLE

Input: (1) Three nodes consisting a triangle u, v, w
(2) Maximum distance considered d
(3) d-DAG $\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$ of graph G
Output: The corresponding (d, s)-graphlet $\mathrm{T}_*^{(d)} \in \mathrm{T}^{(d)}$

```
// Retrieve pairwise distances (Algorithm 4)
d_i ← RETRIEVE_DISTANCE((u, v), d, G⃗^(d))
d_j ← RETRIEVE_DISTANCE((u, w), d, G⃗^(d))
d_k ← RETRIEVE_DISTANCE((v, w), d, G⃗^(d))
// (2,3)-graphlets (d = 2)
if d = 2
    if (d_i, d_j, d_k) ∈ {(2, 2, 2)}
        T*^(2) ← T_1^(2)
    else if (d_i, d_j, d_k) ∈ {(1, 2, 2), (2, 1, 2), (2, 2, 1)}
        T*^(2) ← T_2^(2)
    else if (d_i, d_j, d_k) ∈ {(2, 1, 1), (1, 2, 1), (1, 1, 2)}
        T*^(2) ← T_3^(2)
    else if (d_i, d_j, d_k) ∈ {(1, 1, 1)}
        T*^(2) ← T_4^(2)
// (3,3)-graphlets (d = 3)
if d = 3
    if (d_i, d_j, d_k) ∈ {(3, 3, 3)}
        T*^(3) ← T_1^(3)
    else if (d_i, d_j, d_k) ∈ {(2, 3, 3), (3, 2, 3), (3, 3, 2)}
        T*^(3) ← T_2^(3)
    else if (d_i, d_j, d_k) ∈ {(2, 2, 3), (2, 3, 2), (3, 2, 2)}
        T*^(3) ← T_3^(3)
    else if (d_i, d_j, d_k) ∈ {(1, 3, 3), (3, 1, 3), (3, 3, 1)}
        T*^(3) ← T_4^(3)
    else if (d_i, d_j, d_k) ∈ {(2, 2, 2)}
        T*^(3) ← T_6^(3)
    else if (d_i, d_j, d_k) ∈ {(1, 2, 2), (2, 1, 2), (2, 2, 1)}
        T*^(3) ← T_7^(3)
    else if (d_i, d_j, d_k) ∈ {(1, 1, 1)}
        T*^(3) ← T_9^(3)
return T*^(d)
```

A.4 Details of Algorithm 2

In this subsection, we provide the details of sub-algorithms used in Algorithm 2. Then, we provide the proof of Theorem 2 regarding the complexity of Algorithm 2.

TRIANGLE_PAIRS (Algorithm 8). This function identifies the set of *effective* triangle pairs formed by the edge (u, v), where the remaining nodes are connected. Specifically, we consider a pair of triangles (u, v, w) and (u, v, w') effective if w and w' for a 1-edge or 2-edge. The time complexity of Algorithm 8 is provided in Lemma 6.

LEMMA 6 (COMPLEXITY OF ALGORITHM 8). *The time complexity of retrieving effective triangle pairs is $O(\vec{\Delta}^4 \log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

PROOF. It retrieves the distance for every common neighboring pair (w, w') of (u, v), which takes $O(|N_{u,v}|^2) = O(\vec{\Delta}^{2d})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$. For each pair, retrieving the distance between w and w' takes $O(d^2 \log \vec{\Delta})$ time (Lemma 2). Thus, the total time complexity is $O(\vec{\Delta}^{2d} d^2 \log \vec{\Delta}) = O(\vec{\Delta}^4 \log \vec{\Delta})$ since we assume $d = 2$.

GET_CLIQUE (Algorithm 9). Given two triangles ($(2, 3)$-graphlets) $\mathrm{T}_\circ^{(2)}, \mathrm{T}_\bullet^{(2)}$ and four nodes (u, v, w, w') that form a clique, this function returns the corresponding $(2, 4)$-graphlet of the clique. It first retrieves the distances of the additional necessary edge pairs. Then, based on these four distances and the $(2, 3)$-graphlets of the two triangles, it immediately identifies the $(2, 4)$-graphlet of the clique. The time complexity of Algorithm 9 is provided in Lemma 7.

LEMMA 7 (COMPLEXITY OF ALGORITHM 9). *The time complexity of identifying $(2, 4)$-graphlet of a clique is $O(\log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

PROOF. The time complexity for retrieving the distances of pairs of nodes in the clique is $O(d^2 \log \vec{\Delta})$ (from Lemma 2). Once the distances are obtained, the corresponding $(2, 4)$-graphlet can be identified in $O(1)$ time. Thus, the overall time complexity is $O(\log \vec{\Delta})$ assuming that we use $d = 2$.

GET_NON-INDUCED_WEDGE (Algorithm 11). This function identifies the type of non-induced wedges (which can either be a wedge or a triangle) for a given set of three nodes (u, v, w). Here, we assume that (u, w) is disconnected, and focus on identifying the wedge (i.e., $\mathrm{T}_5^{(2)}$ and $\mathrm{T}_6^{(2)}$) formed by the triple of nodes. To this end, we retrieve the distances between (u, v) and (v, w) and then rapidly identify the corresponding wedge based on these distances. The time complexity of Algorithm 11 is provided in Lemma 8.

LEMMA 8 (COMPLEXITY OF ALGORITHM 11). *The time complexity of identifying $\mathrm{T}_5^{(2)}$ and $\mathrm{T}_6^{(2)}$ of the given triple of nodes (u, v, w), assuming that u and w are disconnected, is $O(\log \vec{\Delta})$, where $\vec{\Delta}$ is the maximum out-degree, i.e., $\vec{\Delta} = \max_{u \in V} |\vec{N}_u^{(1)}|$.*

PROOF. The time complexity for retrieving the distances of pairs of nodes in the clique is $O(d^2 \log \vec{\Delta})$ (from Lemma 2). Once the distances are obtained, the corresponding wedge ($\mathrm{T}_5^{(2)}$ and $\mathrm{T}_6^{(2)}$) can be identified in $O(1)$ time. Thus, the overall time complexity is $O(\log \vec{\Delta})$ assuming that we use $d = 2$.

GET_NON-INDUCED_CYCLE (Algorithm 12). A non-induced cycle can be obtained based on the predefined conditions of the given pair of wedges. Thus, the time complexity of Algorithm 12 is $O(1)$.

COMB_FOUR (Algorithm 13). This function computes the counts of deducible $(2, 4)$-graphlets ($\widehat{\mathrm{Q}}^{(2)}$) and adjusts the counts of semi-deducible $(2, 4)$-graphlets ($\widetilde{\mathrm{Q}}^{(2)}$. Specifically, it leverages node degrees or edge counts to quickly compute these values. In the worst case, enumeration over the $E^{(2)}$. The time complexity of Algorithm 13 is provided in Lemma 9.

LEMMA 9 (COMPLEXITY OF ALGORITHM 13). *The time complexity of computing the counts of deducible $(2, 4)$-graphlets and adjusting the counts of semi-deducible $(2, 4)$-graphlet is $O(|V|\Delta^2)$.*

PROOF. In the worst case, it requires enumeration over $E^{(2)}$, and thus the time complexity is $O(|E|^{(2)}) = O(|V|\Delta^2)$.

Proof of Theorem 2. Below, we provide the proof for Theorem 2.

PROOF. The time complexity of Algorithm 2 is determined by three

Algorithm 7: COMB_THREE

Input: (1) Target deducible (d,s)-graphlet $\mathrm{T}_j^{(d)} \in \widehat{\mathrm{T}}^{(d)}$
(2) Intermediate counts of (d,s)-graphlets $\{C(\mathrm{T}_i^{(d)})\}_{i=1}^{|\mathrm{T}^{(d)}|}$
(3) d-graph $G^{(d)} = (V, E^{(\leq d)})$ of graph G
Output: The count of the target (d,s)-graphlet $\mathrm{T}_j^{(d)}$

// $(2,3)$-graphlets ($d=2$)
if $G^{(d)} = G^{(2)}$
  // Apply the appropriate equation to $\mathrm{T}_j^{(2)}$. The equations should be applied in the below following order.
  $C(\mathrm{T}_3^{(2)}) \leftarrow \sum_{u\in V} \binom{|N_u^{(1)}|}{2} - 3C(\mathrm{T}_4^{(2)})$
  $C(\mathrm{T}_5^{(2)}) \leftarrow \sum_{u\in V} \binom{|N_u^{(2)}|}{2} - 3C(\mathrm{T}_1^{(2)}) - C(\mathrm{T}_2^{(2)})$
  $C(\mathrm{T}_6^{(2)}) \leftarrow \sum_{u\in V} (|N_u^{(1)}|\,|N_u^{(2)}|) - 2C(\mathrm{T}_2^{(2)}) - 2C(\mathrm{T}_3^{(2)})$

// $(3,3)$-graphlets ($d=3$)
if $G^{(d)} = G^{(3)}$
  // Apply the appropriate equation to $\mathrm{T}_j^{(3)}$. The equations should be applied in the below following order.
  $C(\mathrm{T}_8^{(3)}) \leftarrow \sum_{u\in V} \binom{|N_u^{(1)}|}{2} - 3C(\mathrm{T}_9^{(3)})$
  $C(\mathrm{T}_5^{(3)}) \leftarrow \sum_{u\in V} (|N_u^{(1)}|\,|N_u^{(2)}|) - 2C(\mathrm{T}_7^{(3)}) - 2C(\mathrm{T}_8^{(3)})$
  $C(\mathrm{T}_{10}^{(3)}) \leftarrow \sum_{u\in V} \binom{|N_u^{(3)}|}{2} - 3C(\mathrm{T}_1^{(3)}) - C(\mathrm{T}_2^{(3)}) - C(\mathrm{T}_4^{(3)})$
  $C(\mathrm{T}_{11}^{(3)}) \leftarrow \sum_{u\in V} (|N_u^{(2)}|\,|N_u^{(3)}|) - 2C(\mathrm{T}_2^{(3)}) - 2C(\mathrm{T}_3^{(3)}) - 2C(\mathrm{T}_5^{(3)})$
  $C(\mathrm{T}_{12}^{(3)}) \leftarrow \sum_{u\in V} \binom{|N_u^{(2)}|}{2} - C(\mathrm{T}_3^{(3)}) - 3C(\mathrm{T}_6^{(3)}) - C(\mathrm{T}_7^{(3)})$
  $C(\mathrm{T}_{13}^{(3)}) \leftarrow \sum_{u\in V} (|N_u^{(1)}|\,|N_u^{(3)}|) - 2C(\mathrm{T}_4^{(3)}) - C(\mathrm{T}_5^{(3)})$

Algorithm 8: TRIANGLE_PAIRS

Input: (1) Two nodes consisting an edge (u,v)
(2) Common neighbor nodes set between u and v : $N_{u,v}$
(3) 2-DAG $\vec{G}^{(2)} = (V, \vec{E}^{(\leq 2)})$ of graph G
Output: The set of *effective* pairs triangles $\{(u,v,w),(u,v,w')\}$ that share nodes u and v : $\mathcal{T}_{u,v}$

$\mathcal{T}_{u,v} \leftarrow \emptyset$
for *each* $(w,w') \in \binom{N_{u,v}}{2}$
  $\delta(w,w') \leftarrow$ RETRIEVE_DISTANCE$((w,w'), 2, \vec{G}^{(2)})$
  if $\delta(w,w') \neq \infty$
    $\mathcal{T}_{u,v} \leftarrow \mathcal{T}_{u,v} \cup \{\{(u,v,w),(u,v,w')\}\}$
return $\mathcal{T}_{u,v}$

main operations: (1) computing non-deducible $(2,4)$-graphlets, (2) computing semi-deducible $(2,4)$-graphlets, and (3) computing deducible $(2,4)$-graphlets.

- To count **non-deducible** $(2,4)$-graphlets, we iterate over each node $u \in V$. For each of u's neighbor $v \in \vec{N}_u^{(1)} \cup \vec{N}_u^{(2)}$, we first compute the common neighbors $N_{u,v}$ which takes $O(\min(|\vec{N}_u^{(1)} \cup \vec{N}_u^{(2)}|, |\vec{N}_v^{(1)} \cup \vec{N}_v^{(2)}|)) = O(\vec{\Delta}^2)$ time. Using the common neighbors, the effective triangle pairs are retrieved using TRIANGLE_PAIRS, which takes $O(\vec{\Delta}^4 \log \vec{\Delta})$ time (Lemma 6), and the number of pairs is $O(\vec{\Delta}^4)$. Then for each triangle pair, each of the corresponding triangle's $(2,3)$-graphlet is identified. Using these $(2,3)$-graphlets, the clique is then determined, which takes $O(\log \vec{\Delta})$ time (Lemma 7). Thus, the time complexity of counting non-deducible $(2,4)$-graphlets is $(|V|\vec{\Delta}^4 \log \vec{\Delta})$.
- To count **semi-deducible** $(2,4)$-graphlets, for each node u, we enumerate over its neighboring pairs (v,v'), which takes $O(\vec{\Delta}^4)$ time. Then we iterate each of the $O(\Delta^2)$ common neighbors w of v and v', and identify the non-induced wedge which takes $O(\log \vec{\Delta})$ time (Lemma 8). Then the cycle is identified in $O(1)$ time. Thus, the time complexity of counting semi-deducible $(2,4)$-graphlets is $(|V|\vec{\Delta}^4\Delta^2 \log \vec{\Delta})$.
- To count **deducible** $(2,4)$-graphlets, we use the COMB_FOUR which takes $O(|V|\vec{\Delta}^2)$ time.

As a result, counting semi-deducible (d,s)-graphlets dominate the entire complexity, and thus the overall time complexity of Algorithm 2 is $(|V|\vec{\Delta}^4\Delta^2 \log \vec{\Delta})$.

B EXPERIMENT DETAILS

In this section, we provide further details on our experiments.

B.1 Datasets

The details of the datasets and domains are provided below:

- `collaboration` (ca-DBLP [13], ca-Citeseer [3, 24], ca-HepTh [32]): Collaboration networks from various academic fields, where nodes represent authors and edges represent co-authorship between two authors.

Algorithm 9: GET_CLIQUE

Input: (1) Two triangle types $T_{\circ}^{(2)}, T_{\bullet}^{(2)}$
(2) Four nodes consisting a clique u, v, w, w'
(3) 2-DAG $\vec{G}^{(2)} = (V, \vec{E}^{(\leq 2)})$ of graph G
Output: The corresponding (d, s)-graphlet $Q_*^{(d)} \in \overline{Q}^{(d)}$

$d_{(u,v)} \leftarrow$ RETRIEVE_DISTANCE$(u, v, 2, \vec{G}^{(2)})$
$d_{(u,w)} \leftarrow$ RETRIEVE_DISTANCE$(u, w, 2, \vec{G}^{(2)})$
$d_{(u,w')} \leftarrow$ RETRIEVE_DISTANCE$(u, w', 2, \vec{G}^{(2)})$
$d_{(w,w')} \leftarrow$ RETRIEVE_DISTANCE$(w, w', 2, \vec{G}^{(2)})$
if $d_{(u,v)} = 2$
  if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_1^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_1^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_1^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_2^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_2^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_2^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_2^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_4^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_3^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_4^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_1^{(2)}, T_3^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_6^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_2^{(2)}, 2)$
    if $d_{(u,w)} = d_{(u,w')}$
      $Q_*^{(2)} \leftarrow Q_4^{(2)}$
    else
      $Q_*^{(2)} \leftarrow Q_3^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_2^{(2)}, 1)$
    if $d_{(u,w)} = d_{(u,w')}$
      $Q_*^{(2)} \leftarrow Q_7^{(2)}$
    else
      $Q_*^{(2)} \leftarrow Q_5^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_3^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_5^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_3^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_9^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_3^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_8^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_3^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_{10}^{(2)}$
  // Continue on the right side.

Algorithm 10: GET_CLIQUE (CONTINUED)

else
  if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_2^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_2^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_2^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_3^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_3^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_4^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_3^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_5^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_4^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_7^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_2^{(2)}, T_4^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_9^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_3^{(2)}, 2)$
    if $d_{(u,w)} = d_{(u,w')}$
      $Q_*^{(2)} \leftarrow Q_6^{(2)}$
    else
      $Q_*^{(2)} \leftarrow Q_5^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_3^{(2)}, 1)$
    if $d_{(u,w)} = d_{(u,w')}$
      $Q_*^{(2)} \leftarrow Q_9^{(2)}$
    else
      $Q_*^{(2)} \leftarrow Q_8^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_4^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_9^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_3^{(2)}, T_4^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_{10}^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_4^{(2)}, T_4^{(2)}, 2)$
    $Q_*^{(2)} \leftarrow Q_{10}^{(2)}$
  else if $(T_{\circ}^{(2)}, T_{\bullet}^{(2)}, d_{(w,w')}) = (T_4^{(2)}, T_4^{(2)}, 1)$
    $Q_*^{(2)} \leftarrow Q_{11}^{(2)}$
return $Q_*^{(2)}$

Algorithm 11: GET_NON-INDUCED_WEDGE

Input: (1) Three nodes consisting a wedge (u, v, w)
(2) d-DAG $\vec{G}^{(d)} = (V, \vec{E}^{(\leq d)})$ of graph G
Output: The corresponding (d, s)-graphlet $T_*^{(2)} \in T^{(2)}$

$\delta(u, v) \leftarrow$ RETRIEVE_DISTANCE$(u, v, 2, \vec{G}^{(2)})$
$\delta(v, w) \leftarrow$ RETRIEVE_DISTANCE$(v, w, 2, \vec{G}^{(2)})$
if $(\delta(u, v), \delta(v, w) \in \{(1, 2), (2, 1)\}$
  $T_*^{(2)} \leftarrow T_6^{(2)}$
else
  $T_*^{(2)} \leftarrow T_5^{(2)}$
return $T_*^{(2)}$

- `web` [11, 12, 52] (web-Arabic, web-Indochina): Web networks, where nodes represent web pages and edges represent hyperlinks between pages.
- `social-Facebook` [57, 58] (soc-UCSC, soc-UC, soc-MB): Social friendship networks from Facebook at various US schools, where nodes represent users and edges represent friendship connections between them.
- `tags` [7] (tags-Ubuntu, tags-Math): Tag co-occurrence networks from question-and-answer sites, where nodes represent tags and edges link tags that appear together on the same question post.

Algorithm 12: GET_NON-INDUCED_CYCLE

Input: (1) Two wedge type $T_{\triangle}^{(2)}, T_{\blacktriangle}^{(2)}$
Output: The corresponding (d, s)-graphlet $Q_{*}^{(2)} \in \widetilde{Q}^{(2)}$

```
if (T_△^(2), T_▲^(2)) ∈ {(T_5^(2), T_5^(2))}
    Q_*^(2) ← Q_30^(2)
else if (T_△^(2), T_▲^(2)) ∈ {(T_5^(2), T_6^(2)), (T_6^(2), T_5^(2))}
    Q_*^(2) ← Q_29^(2)
else
    Q_*^(2) ← Q_28^(2)
return Q_*^(2)
```

Table 4: Statistics for 13 real-world graphs across 5 domains: $|E^{(d)}|$ is the number of d-edges, and $|T^{(d)}|$ and $|Q^{(d)}|$ are the counts of size-3 and size-4 (d, s)-graphlets, respectively.

Dataset	$\lvert V\rvert$	$\lvert E^{(1)}\rvert$	$\lvert E^{(2)}\rvert$	$\lvert E^{(3)}\rvert$	$\lvert T^{(2)}\rvert$	$\lvert T^{(3)}\rvert$	$\lvert Q^{(2)}\rvert$
ca-DBLP	317K	1.05M	12.7M	153M	4.89B	769B	3.95T
ca-Citeseer	227K	814K	7.38M	50.4M	1.70B	107B	659B
ca-HepTh	9.88K	26.0K	179K	1.10M	21.8M	797M	3.77B
web-Arabic	164K	1.75M	3.06M	14.9M	755M	12.6B	205B
web-Indochina	11.4K	47.6K	425K	3.70M	121M	2.93B	54.5B
soc-UCSC	8.99K	225K	7.19M	24.8M	12.7B	98.9B	21.9T
soc-UC	6.83K	155K	4.72M	13.5M	7.13B	42.5B	9.80T
soc-MB	3.08K	125K	2.35M	1.96M	2.34B	4.62B	1.79T
tags-Ubuntu	3.03K	133K	3.66M	764K	3.96B	4.59B	3.07T
tags-Math	1.63K	91.7K	1.08M	152K	661M	716M	275B
road-CA	1.97M	2.77M	5.12M	8.07M	45.0M	189M	301M
road-PA	1.09M	1.54M	2.88M	4.58M	25.7M	109M	175M
road-TX	1.38M	1.92M	3.52M	5.55M	30.7M	128M	202M

- `road` [34] (road-CA, road-PA, road-TX): Road networks from various US regions, where nodes represent intersections or road endpoints, and edges represent the roads connecting them.

We removed self-loops for our analysis. The preprocessed datasets can be accessed at **https://github.com/thisis05/EDGE**. All original datasets used in this study are publicly available from [7, 33, 52]. We present the dataset statistics, including the number of nodes, edges, and graphlet instances, in Table 4.

B.2 Importance Scores

Among the $(2, 4)$-graphlets, $Q_6^{(2)}, Q_8^{(2)}, Q_9^{(2)}, Q_{10}^{(2)}, Q_{11}^{(2)}, Q_{19}^{(2)}$ can also be represented in the original size-4 graphlet. Since the 2-edges of these instances can be inferred from the 1-edge (i.e., the distance between nodes that are not connected by a 1-edge is guaranteed to be at most 2.), they are naturally represented in the $(2, 4)$-graphlets as well. The remaining $(2, 4)$-graphlet instances are newly captured local structures, identified by considering distances up to 2. To understand how the newly defined (d, s)-graphlet plays a significant role in characterization, we use the scoring function proposed by [30], which denotes the importance of each graphlet g.

$$Importance(g) = 1 - \frac{dist_{within}(g)}{dist_{across}(g)}$$

Table 5: Importance scores of $(2, 4)$-graphlets for each dataset. Each rank is based on the importance score, and each value represents the index of graphlet instance, with the score shown in parentheses.

Dataset	1st	2nd	3rd	4th	5th
ca-DBLP	**4** (0.91)	**13** (0.90)	**2** (0.86)	**3** (0.85)	**27** (0.76)
ca-Citeseer	**4** (0.82)	**13** (0.81)	**3** (0.76)	**5** (0.76)	**2** (0.70)
ca-HepTh	**4** (0.89)	**13** (0.89)	**2** (0.84)	**5** (0.79)	**3** (0.79)
web-Arabic	**3** (0.99)	**5** (0.98)	**36** (0.94)	**12** (0.94)	8 (0.93)
web-Indochina	**3** (0.99)	**5** (0.98)	**36** (0.94)	**12** (0.94)	8 (0.94)
soc-UCSC	**5** (0.96)	**4** (0.91)	**3** (0.89)	**2** (0.89)	**1** (0.87)
soc-UC	**5** (0.95)	**4** (0.93)	**3** (0.91)	**2** (0.91)	6 (0.89)
soc-MB	**5** (0.93)	**27** (0.89)	**18** (0.88)	**4** (0.85)	6 (0.84)
tags-Ubuntu	6 (0.96)	**20** (0.94)	**3** (0.94)	**22** (0.93)	**28** (0.91)
tags-Math	6 (0.96)	**3** (0.94)	**20** (0.93)	**22** (0.92)	**14** (0.91)
road-CA	**30** (0.99)	**17** (0.99)	**29** (0.99)	**12** (0.99)	**28** (0.99)
road-PA	**17** (0.99)	**30** (0.99)	**12** (0.99)	**14** (0.99)	**29** (0.99)
road-TX	**30** (0.99)	**29** (0.99)	**17** (0.99)	**28** (0.99)	**12** (0.99)

$dist_{within}(g)$ is the average CP distance between other graphs from the same domain, and $dist_{across}(g)$ is the average CP distance between other graphs from different domains. We calculate the importance of $(2, 4)$-graphlets across all 13 datasets and display the top 5 instances for each dataset in Table 5, where the graphlet indices are ranked by the importance score.

B.3 Exact Counting Time for All Algorithms

In Section 6.4, we evaluate the counting time of EDGE from two perspectives: (1) in comparison to conventional graphlet counting algorithms (PGD and ESCAPE) for size-4 (d, s)-graphlets, and (2) against two ablation variants of EDGE (EDGE-D2 and EDGE-D). Table 6 presents the exact counting time of all algorithms along with additional information about graphlet instances.

B.4 Machine Learning Applications

To evaluate the effectiveness of (d, s)-graphlets as feature extractors for graph machine learning tasks, we performed additional experiments on link prediction and graph classification.

Link prediction. We generated concise vectors where each element corresponds to the normalized count of (d, s)-graphlets associated with each node. In order to predict whether two nodes are adjacent, the two corresponding vectors were fed into an MLP classifier. As shown in Table 7, the accuracies achieved using $(3, 3)$-graphlets and $(2, 4)$-graphlets consistently outperform those achieved using conventional graphlets across the three datasets. Notably, even the performances with $(2, 3)$-graphlets outperform those with $(1, 4)$-graphlets in some datasets (e.g., in soc-UC).

Graph classification. Our graph classification task aims to predict which dataset a given ego-network belongs to. To this end, we employed an MLP classifier using the normalized counts of (d, s)-graphlets in the ego-network as input features. For this experiment, we sampled 100 ego-networks from each 10 datasets listed in Table 4, except for those from the road domain due to their large sizes. In total, we sampled 1,000 ego-networks. As a competitor, we employed GraphCL [68], a graph neural network (GNN) trained in

Algorithm 13: COMB_FOUR

Input: (1) Target deducible or semi-deducible $(2,4)$-graphlet $\mathrm{Q}_j^{(2)} \in \widehat{\mathrm{Q}}^{(2)} \cup \widetilde{\mathrm{Q}}^{(2)}$
(2) Intermediate counts of (d,s)-graphlets $\{C(\mathrm{Q}_i^{(d)})\}_{i=1}^{|\mathrm{Q}^{(d)}|}$
(3) The counts of each $\mathrm{T}_i^{(2)}$ per edge $\{C_e(\mathrm{T}_i^{(2)})\}_{i=1}^{|\mathrm{T}^{(2)}|}$
(4) 2-graph $G^{(2)} = (V, E^{(\leq 2)})$ of graph G

Output: The count of the target (d,s)-graphlet $\mathrm{Q}_j^{(d)}$

// Apply the appropriate equation to $\mathrm{Q}_j^{(2)}$. The equations should be applied in the below following order.

$C(\mathrm{Q}_{12}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \binom{C_{(u,v)}(\mathrm{T}_1^{(2)})}{2} - 6C(\mathrm{Q}_1^{(2)}) - C(\mathrm{Q}_2^{(2)})$

$C(\mathrm{Q}_{13}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} \binom{C_{(u,v)}(\mathrm{T}_2^{(2)})}{2} - C(\mathrm{Q}_2^{(2)}) - 2C(\mathrm{Q}_3^{(2)})$

$C(\mathrm{Q}_{14}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} (C_{(u,v)}(\mathrm{T}_1^{(2)}) C_{(u,v)}(\mathrm{T}_2^{(2)})) - 4C(\mathrm{Q}_2^{(2)}) - 2C(\mathrm{Q}_4^{(2)})$

$C(\mathrm{Q}_{15}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \binom{C_{(u,v)}(\mathrm{T}_2^{(2)})}{2} - 4C(\mathrm{Q}_3^{(2)}) - C(\mathrm{Q}_5^{(2)}) - C(\mathrm{Q}_4^{(2)}) - 3C(\mathrm{Q}_7^{(2)})$

$C(\mathrm{Q}_{16}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} (C_{(u,v)}(\mathrm{T}_2^{(2)}) C_{(u,v)}(\mathrm{T}_3^{(2)})) - 2C(\mathrm{Q}_4^{(2)}) - 2C(\mathrm{Q}_5^{(2)})$

$C(\mathrm{Q}_{17}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} (C_{(u,v)}(\mathrm{T}_1^{(2)}) C_{(u,v)}(\mathrm{T}_3^{(2)})) - C(\mathrm{Q}_4^{(2)}) - 3C(\mathrm{Q}_6^{(2)})$

$C(\mathrm{Q}_{18}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} (C_{(u,v)}(\mathrm{T}_2^{(2)}) C_{(u,v)}(\mathrm{T}_4^{(2)})) - 3C(\mathrm{Q}_7^{(2)}) - C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{19}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} \binom{C_{(u,v)}(\mathrm{T}_3^{(2)})}{2} - C(\mathrm{Q}_5^{(2)}) - 4C(\mathrm{Q}_8^{(2)}) - 3C(\mathrm{Q}_6^{(2)}) - C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{20}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_1^{(2)})(|N_u^{(2)}| + |N_v^{(2)}|)\} - 4C(\mathrm{Q}_{12}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - 12C(\mathrm{Q}_1^{(2)}) - 4C(\mathrm{Q}_2^{(2)}) - C(\mathrm{Q}_4^{(2)})$

$C(\mathrm{Q}_{21}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_1^{(2)})(|N_u^{(1)}| + |N_v^{(1)}|)\} - C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_{17}^{(2)}) - 2C(\mathrm{Q}_2^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - 3C(\mathrm{Q}_6^{(2)})$

$C(\mathrm{Q}_{22}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} \{C_{(u,v)}(\mathrm{T}_2^{(2)})(|N_u^{(2)}|+|N_v^{(2)}|)\} - 4C(\mathrm{Q}_{13}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_{15}^{(2)}) - C(\mathrm{Q}_{16}^{(2)}) - 4C(\mathrm{Q}_2^{(2)}) - 8C(\mathrm{Q}_3^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - 2C(\mathrm{Q}_5^{(2)})$

$C(\mathrm{Q}_{23}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_2^{(2)})(|N_u^{(2)}|+|N_v^{(2)}|)\} - \sum_{(u,v)\in E^{(1)}} \{C_{(u,v)}(\mathrm{T}_2^{(2)})(|N_u^{(2)}|+|N_v^{(2)}|)\} - C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_2^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - 3C(\mathrm{Q}_7^{(2)})$

$C(\mathrm{Q}_{24}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_2^{(2)})(|N_u^{(1)}|+|N_v^{(1)}|)\} - \sum_{(u,v)\in E^{(1)}} \{C_{(u,v)}(\mathrm{T}_2^{(2)})(|N_u^{(1)}|+|N_v^{(1)}|)\} - 2C(\mathrm{Q}_{15}^{(2)}) - 4C(\mathrm{Q}_3^{(2)}) - 2C(\mathrm{Q}_5^{(2)}) - C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{25}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} \{C_{(u,v)}(\mathrm{T}_3^{(2)})(|N_u^{(2)}| + |N_v^{(2)}|)\} - \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_3^{(2)})(|N_u^{(2)}| + |N_v^{(2)}|)\} - C(\mathrm{Q}_{16}^{(2)}) - 2C(\mathrm{Q}_{19}^{(2)})$
$\quad - C(\mathrm{Q}_4^{(2)}) - 2C(\mathrm{Q}_5^{(2)}) - 4C(\mathrm{Q}_8^{(2)})$

$C(\mathrm{Q}_{26}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{C_{(u,v)}(\mathrm{T}_3^{(2)})(|N_u^{(2)}| + |N_v^{(2)}|)\}) - C(\mathrm{Q}_{16}^{(2)}) - 2 \cdot C(\mathrm{Q}_{17}^{(2)}) - 2 \cdot C(\mathrm{Q}_4^{(2)}) - 2 \cdot C(\mathrm{Q}_5^{(2)}) - 6 \cdot C(\mathrm{Q}_6^{(2)}) - 2 \cdot C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{27}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} \{C_{(u,v)}(\mathrm{T}_4^{(2)})(|N_u^{(2)}| + |N_v^{(2)}|)\} - 2 \cdot C(\mathrm{Q}_{18}^{(2)}) - C(\mathrm{Q}_9^{(2)}) - 2 \cdot C(\mathrm{Q}_{10}^{(2)}) - 3 \cdot C(\mathrm{Q}_7^{(2)}) - C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{28}^{(2)}) \leftarrow C(\mathrm{Q}_{28}^{(2)}) - C(\mathrm{Q}_{12}^{(2)}) - C(\mathrm{Q}_{13}^{(2)}) - 3 \cdot C(\mathrm{Q}_1^{(2)}) - C(\mathrm{Q}_2^{(2)}) - C(\mathrm{Q}_3^{(2)})$

$C(\mathrm{Q}_{29}^{(2)}) \leftarrow C(\mathrm{Q}_{29}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - C(\mathrm{Q}_{16}^{(2)}) - 2 \cdot C(\mathrm{Q}_2^{(2)}) - 2 \cdot C(\mathrm{Q}_4^{(2)}) - C(\mathrm{Q}_5^{(2)})$

$C(\mathrm{Q}_{30}^{(2)}) \leftarrow C(\mathrm{Q}_{30}^{(2)}) - C(\mathrm{Q}_{15}^{(2)}) - C(\mathrm{Q}_{19}^{(2)}) - 2C(\mathrm{Q}_3^{(2)}) - C(\mathrm{Q}_5^{(2)}) - 2C(\mathrm{Q}_8^{(2)})$

$C(\mathrm{Q}_{31}^{(2)}) \leftarrow \sum_{u\in V} \binom{N_u^{(2)}}{3} - C(\mathrm{Q}_{20}^{(2)}) - C(\mathrm{Q}_{23}^{(2)}) - 2C(\mathrm{Q}_{12}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - 4C(\mathrm{Q}_1^{(2)}) - 2C(\mathrm{Q}_2^{(2)}) - C(\mathrm{Q}_4^{(2)}) - C(\mathrm{Q}_7^{(2)})$

$C(\mathrm{Q}_{32}^{(2)}) \leftarrow \sum_{u\in V} \{\binom{|N_u^{(2)}|}{2} |N_u^{(1)}|\} - C(\mathrm{Q}_{21}^{(2)}) - C(\mathrm{Q}_{22}^{(2)}) - C(\mathrm{Q}_{24}^{(2)}) - C(\mathrm{Q}_{26}^{(2)}) - 2C(\mathrm{Q}_{13}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_{15}^{(2)}) - C(\mathrm{Q}_{16}^{(2)}) - 2C(\mathrm{Q}_{17}^{(2)})$
$\quad -2C(\mathrm{Q}_2^{(2)}) - 4C(\mathrm{Q}_3^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - 2C(\mathrm{Q}_5^{(2)}) - 3C(\mathrm{Q}_6^{(2)}) - C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{33}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{(|N_u^{(2)}| - 1)(|N_v^{(2)}| - 1) - C_{(u,v)}(\mathrm{T}_1^{(2)})\} - 4C(\mathrm{Q}_{28}^{(2)}) - C(\mathrm{Q}_{29}^{(2)}) - 2C(\mathrm{Q}_{20}^{(2)}) - C(\mathrm{Q}_{22}^{(2)}) - 6C(\mathrm{Q}_{12}^{(2)}) - 4C(\mathrm{Q}_{13}^{(2)})$
$\quad -2C(\mathrm{Q}_{14}^{(2)}) - C(\mathrm{Q}_{15}^{(2)}) - C(\mathrm{Q}_{16}^{(2)}) - 12C(\mathrm{Q}_1^{(2)}) - 6C(\mathrm{Q}_2^{(2)}) - 4C(\mathrm{Q}_3^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - C(\mathrm{Q}_5^{(2)})$

$C(\mathrm{Q}_{34}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} \{|N_u^{(1)}|(|N_v^{(2)}| - 1) + (|N_u^{(2)}| - 1)|N_v^{(2)}| - \tfrac{1}{2} C_{(u,v)}(\mathrm{T}_2^{(2)})\} - 2C(\mathrm{Q}_{29}^{(2)}) - 2C(\mathrm{Q}_{21}^{(2)}) - 2C(\mathrm{Q}_{23}^{(2)}) - C(\mathrm{Q}_{26}^{(2)})$
$\quad -3C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_{16}^{(2)}) - 4C(\mathrm{Q}_{17}^{(2)}) - 2C(\mathrm{Q}_{18}^{(2)}) - 4C(\mathrm{Q}_2^{(2)}) - 6C(\mathrm{Q}_4^{(2)}) - 2C(\mathrm{Q}_5^{(2)}) - 6C(\mathrm{Q}_6^{(2)}) - 6C(\mathrm{Q}_7^{(2)}) - 2C(\mathrm{Q}_9^{(2)})$

$C(\mathrm{Q}_{35}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(1)}} (|N_u^{(2)}||N_v^{(2)}| - C_{(u,v)}(\mathrm{T}_2^{(2)})) - C(\mathrm{Q}_{29}^{(2)}) - 2C(\mathrm{Q}_{30}^{(2)}) - C(\mathrm{Q}_{22}^{(2)}) - 2C(\mathrm{Q}_{25}^{(2)}) - 2C(\mathrm{Q}_{13}^{(2)}) - C(\mathrm{Q}_{14}^{(2)}) - 2C(\mathrm{Q}_{15}^{(2)})$
$\quad -2C(\mathrm{Q}_{16}^{(2)}) - 3C(\mathrm{Q}_{19}^{(2)}) - 2C(\mathrm{Q}_2^{(2)}) - 4C(\mathrm{Q}_3^{(2)}) - 2C(\mathrm{Q}_4^{(2)}) - 3C(\mathrm{Q}_5^{(2)}) - 4C(\mathrm{Q}_8^{(2)})$

$C(\mathrm{Q}_{36}^{(2)}) \leftarrow \sum_{(u,v)\in E^{(2)}} (|N_u^{(1)}||N_v^{(1)}| - C_{(u,v)}(\mathrm{T}_3^{(2)})) - 2C(\mathrm{Q}_{30}^{(2)}) - 2C(\mathrm{Q}_{24}^{(2)}) - 3C(\mathrm{Q}_{15}^{(2)}) - 2C(\mathrm{Q}_{19}^{(2)}) - 4C(\mathrm{Q}_3^{(2)}) - 3C(\mathrm{Q}_5^{(2)}) - 4C(\mathrm{Q}_8^{(2)})$
$\quad -2C(\mathrm{Q}_9^{(2)}) - 2C(\mathrm{Q}_{10}^{(2)})$

Table 6: Dataset statistics and exact counting time (sec.) for each algorithm. $|T^{(d)}|$ represents the number of instances of size-3 (d,s)-graphlets, and $|Q^{(d)}|$ represents the number of instances of size-4 (d,s)-graphlets. Additionally, the table includes the counting time for the conventional 4-size graphlet counting algorithms, PGD and ESCAPE, as well as the counting time for our method EDGE and the two baselines (EDGE-D2, EDGE-D).

Datasets	$\|T^{(2)}\|$	$\|T^{(3)}\|$	$\|Q^{(1)}\|$	$\|Q^{(2)}\|$	PGD	ESCAPE	EDGE-D2-(2,3)	EDGE-D-(2,3)	EDGE-(2,3)	EDGE-D2-(3,3)	EDGE-D-(3,3)	EDGE-(3,3)	EDGE-(2,4)
ca-DBLP	4.89B	769B	629M	3.95T	0.381	0.297	82.3	17.5	4.31	19.0K	4.64K	913	62.5
ca-Citeseer	1.70B	107B	806M	659B	0.206	0.216	40.1	11.6	3.84	2.64K	651	154	807
ca-HepTh	21.8M	797M	3.99M	3.77B	0.004	0.003	0.40	0.09	0.03	23.1	5.52	1.32	0.17
web-Arabic	755M	12.6B	779M	205B	0.427	1.25	23.2	7.58	2.72	308	69.0	18.0	607
web-Indochina	121M	2.93B	23.2M	54.5B	0.009	0.009	1.23	0.30	0.06	89.3	24.2	6.70	1.46
soc-UCSC	12.7B	98.9B	1.97B	21.9T	0.361	0.323	363	103	27.6	5.80K	1.82K	627	3.87K
soc-UC	7.13B	42.5B	1.60B	9.80T	0.296	0.256	212	60.0	15.7	2.53K	789	273	2.75K
soc-MB	2.34B	4.62B	1.99B	1.79T	0.372	0.339	95.8	27.4	8.14	344	102	36.5	2.40K
tags-Ubuntu	3.96B	4.59B	14.2B	3.07T	1.95	1.11	170	54.0	17.6	327	89.9	33.0	9.74K
tags-Math	661M	716M	4.76B	275B	1.17	0.938	33.0	9.80	3.20	51.4	13.2	5.14	936
road-CA	45.0M	189M	14.0M	301M	0.312	0.285	7.76	7.66	7.30	19.0	17.5	15.5	7.88
road-PA	25.7M	108.9M	8.01M	175M	0.187	0.149	2.73	2.64	2.53	7.16	5.86	5.19	2.75
road-TX	30.7M	128M	9.52M	202M	0.238	0.188	4.24	4.20	3.96	10.4	9.33	8.2	4.31

Table 7: Link prediction performance. $(1,3)$ and $(1,4)$ correspond to the original graphlets, while $(2,3)$, $(3,3)$, and $(2,4)$ correspond to our proposed (d,s)-graphlets. (d,s)-graphlets presents overall better link prediction performance compared to original graphlets.

Dataset	Metrics	$(1,3)$	$(1,4)$	$(2,3)$	$(2,4)$	$(3,3)$
web-Indochina	ACC	0.720	0.772	0.774	**0.803**	0.799
	AP	0.837	0.918	0.934	**0.982**	0.965
	AUC	0.838	0.921	0.934	**0.984**	0.970
soc-UC	ACC	0.645	0.684	0.696	**0.735**	0.728
	AP	0.694	0.799	0.817	**0.866**	0.851
	AUC	0.727	0.800	0.824	**0.869**	0.859
ca-HepTh	ACC	0.696	0.733	0.736	**0.765**	0.756
	AP	0.781	0.886	0.864	**0.944**	0.916
	AUC	0.802	0.888	0.878	**0.947**	0.922

a self-supervised manner for node feature learning. Since most of our datasets lack external node features, the input node features were derived by applying PCA to the Laplacian matrix of each ego-network. The features were processed by a GNN to generate latent node features, which were then aggregated into ego-network-level features. The accuracy of graph classification with features learned from GraphCL was **0.705**. In contrast, the accuracies with features derived from (2,3)- and (3,3)-graphlets were **0.718** and **0.780**, respectively.

These extra results on link prediction and graph classification demonstrate the effectiveness of (d,s)-graphlets as a feature extractor, especially when external node features are not available.