E-CLoG: Counting Edge-Centric Local Graphlets

Vachik S. Dave*, Nesreen K. Ahmed+ and Mohammad Al Hasan*

Abstract*— In recent years, graphlet counting has emerged as an important task in topological graph analysis. However, the existing works on graphlet counting obtain the graphlet counts for the entire network as a whole. These works capture the key graphical patterns that prevail in a given network but they fail to meet the demand of the majority of real-life graph related prediction tasks such as link prediction, edge/node classification, etc., which require to build features for an edge (or a vertex) of a network. To meet the demand for such applications, efficient algorithms are needed for counting local graphlets within the context of an edge (or a vertex). In this work, we propose an efficient method, titled E-CLoG, for counting all $3, 4$ and 5 size local graphlets with the context of a given edge for its all different edge orbits. We also provide a shared-memory, multi-core implementation of E-CLoG, which makes it even more scalable for very large real-world networks. In particular, We obtain strong scaling on a variety of graphs (14x–20x on 36 cores). We provide extensive experimental results to demonstrate the efficiency and effectiveness of the proposed method. For instance, we show that E-CLoG is faster than existing work by multiple order of magnitudes; for the Wordnet graph E-CLoG counts all 3,4 and 5-size local graphlets in 1.5 hours using a single thread and in only a few minutes using the parallel implementation, whereas the baseline method does not finish in more than 4 days. We also show that local graphlet counts around an edge are much better features for link prediction than well-known topological features; our experiments show that the former enjoys between 10% to 45% of improvement in the AUC value for predicting future links in three real-life social and collaboration networks.

I. Introduction

Frequency distribution of small induced subgraphs (aka graphlets) captures key connectivity patterns in a given network, hence this distribution is increasingly being used for various network analysis tasks. For instance, Rahman et al. [1] and Ahmed et al. [2] have used graphlet frequencies for network classification; Ugander et al. [3] have used the same for modeling network structures. Besides these, graphlet frequencies have also been used for solving problems in various other disciplines, examples include biological network comparisons [4], [5], image classification [6], and building graph kernels for chemoinformatics [7]. In all the above applications, a vector representing the frequency (normalized or unnormalized) of small-sized graphlets (typically 3 to 5 vertices) induced in a network is used as a signature for capturing the connectivity patterns of the entire network as a whole. Obtaining such a vector is a costly task, but several recent algorithms have been proposed for solving it efficiently [1], [2]. To overcome the lack of scalability issue, parallel [2], [8] and sampling-based approximation algorithms [9], [10], [11], [12] have also been proposed.

Global graphlet frequencies are useful for network analysis tasks where the entire network as a whole needs to be modeled for the task of network-level classification or comparison. Unfortunately, the majority of the real-life network analysis tasks do not consider the network as a whole, rather they consider vertices or edges as first-class entities, and perform prediction on the attributes of nodes or edges. For example, popular tasks on social networks, such as, expert search, and role discovery are performed by some form of node classification [13]. On the other hand, tasks like link prediction [14], relationship-type prediction, and product recommendation are solved by edge classification [15]. For solving such tasks, a global graphlet count considering the entire network is not much useful. We rather want to obtain graphlet counts within the local context of a vertex or an edge for capturing the topological neighborhood around that vertex or edge. This variant of graphlet counting is called local graphlet counting, which is increasingly being used for network analysis tasks. For instance, V. Vacic et al. [16] have used local graphlet counts to classify protein residues, A. Nabhan et al. [17] have used local graphlet counts for keyword identification from text. Many of the link prediction features, such as Adamic-Adar, Jaccard-Coefficient, and Preferential Attachment are functions of local frequency of a specific graphlet, namely, a triangle.

Although very useful, there are not many works that have considered counting the local graphlet frequencies, beyond triangles. An ad-hoc solution to this deficiency is to restrict a large graph within a local context by building a contextual subgraph and then apply global graphlet count algorithm on that subgraph. For instance Hermansson et al. [18] have built an ego network of a given node and then used global graphlet counting on the ego network to obtain a local graphlet counter. Such a method is an approximation, and more importantly they are an overkill because a global graphlet counting method needs to be called for all vertices (for obtaining node-centric graphlet counts) or for all edges (for obtaining edge-centric counting). There exist few works which address local graphlet counting for the case of triangles. For example, Y. Lim et al. [19] and Ahmed et al. [20] have proposed estimation method to count local/global triangles from streaming data. T. Hočevar et al. [21] have proposed node-centric local grpahlet counting. Recently, Ahmed et al. [22] have proposed an efficient parallel algorithm for exact and approximate local graphlet counting upto size $3, 4$-vertex graphlets, which enumerates only a few graphlets and computes the reamining graphlets

*Department of Computer Science, Indiana University Purdue University Indianapolis, IN, USA. vsdave,alhasan@iupui.edu
+Intel Labs, Santa Clara, CA. nesreen.k.ahmed@intel.com

in constant time (using combinatorial arguments). However, the existing local graphlet counting methods do not find the local graphlet counts for different vertex and/or edge orbits. This is important because orbits represent the role of a vertex or an edge within a given graphlet. An aggregated count of a graphlet, without considering orbits, misrepresents the local topological configuration around a vertex or an edge.

In this paper, we provide an efficient method, namely E-CLOG[1] for obtaining local graphlet counts with respect to a given edge in a graph. E-CLOG counts all $3, 4$, and 5-size local graphlets considering *all possible* edge orbits of a graphlet. By considering edge orbits there are 8 of size-4 and 32 of size-5 graphlets, and E-CLOG obtains the frequency of all of them. The method is efficient because it does not enumerate all of the above graphlets, rather it only enumerates 4 out of 8 size-4 graphlets, and 14 out of 32 size-5 graphlets. The remaining graphlet counts are obtained in constant time by combinatorial calculation over carefully-designed data structures. We also provide a multi-core parallel implementation of E-CLOG, which is highly scalable.

We claim the following contributions:

1) We propose E-CLOG, an efficient method for counting edge-centric local graphlets up-to size-5 considering edge orbits. To the best of our knowledge, E-CLOG is the first work that obtains local graphlet counts for size-5 graphlets. It is also the first work which considers edge orbits in its counts.
2) We also provide a shared-memory, multi-core implementation of E-CLOG, which makes it even more scalable for very large real-world networks.
3) We show experiments which validate E-CLOG's effectiveness as local edge features for network analysis, specifically for the task of link prediction.

II. PROBLEM FORMULATION

Let $G(V, E)$ be a simple, undirected graph, where V is the set of vertices and E is the set of edges. For a vertex $u \in V$, neighbors of u are represented by $\Gamma(u) = \{v|(u, v) \in E\}$ and degree of u is represented by $d(u) = |\Gamma(u)|$.

Definition 1 (Induced graphlet): An undirected graph $g(V', E')$ is an induced graphlet of $G(V, E)$, if V' is a subset of V and E' consists of all of the edges in E that have both endpoints in V'. If $|V'| = k$, we call it k-size (or size-k) induced graphlet, or in-short, k-graphlet in G. We also consider that all k-graplets are connected. ■

In this work, given a graph $G(V, E)$ and a specific edge $e = (u, v) \in E$, we count the frequency of edge centric local graphlets. Such an edge centric local graphlet (say, g) must include the given edge e; besides, the remaining vertices of g must be a neighbor of u and/or a neighbor of v. Below is a formal definition of edge-centric local graphlets.

Definition 2 (Edge centric local graphlet): Say, $g(V', E')$ is a k-graphlet of a graph $G(V, E)$. Now, given an edge $(u, v) \in E$, g is called a size-k edge centric local graphlet (or size-k local graphlet), if $(u, v) \in E'$ and $w \in V' \setminus \{u, v\} \Rightarrow w \in \Gamma(u) \cup \Gamma(v)$. ■

Say, $\mathcal{G}_k(u, v)$ is the set of all connected graphical topologies such that each member of this set is a possible k-size edge centric local graphlet with respect to an edge (u, v). In Figures 1, we show all members of $\mathcal{G}_3(u, v)$ and $\mathcal{G}_4(u, v)$, and in Figure 3, we show all members of $\mathcal{G}_5(u, v)$. In each of these graphlets, the edge (u, v) is clearly identified. Notice that, if we ignore the vertex labels u and v, there are multiple graphlets in these figures, which are topologically identical. For examples, in Figure 1 graphlets $g7$ and $g8$ have the same 4-size graph structure (lets call it two-triangles), but they are represented as two distinct graphlets ($g7$ and $g8$). We identify these structurally identical graphlets differently, based on edge orbit of the given edge (u, v). Below we formally define edge orbit and other necessary terminologies.

Definition 3 (Graphlet Isomorphism): A Graphlet $g_1(V_1, E_1)$ is said to be isomorphic to another graphlet $g_2(V_2, E_2)$, if there exists a bijective function $f_{iso} : V_1 \rightarrow V_2$ such that $(u, v) \in E_1 \Leftrightarrow (f_{iso}(u), f_{iso}(v)) \in E_2$ and the bijection function f_{iso} is called graphlet isomorphism from g_1 to g_2. ■

Definition 4 (Graphlet Automorphism): A graphlet automorphism is graphlet isomorphism with itself, i.e. a bijection function f_{auto} that maps the set of vertices V_1 back to V_1 with a different permutation and satisfies the properities of graphlet isomorphism. ■

Definition 5 (Edge orbit): For a given graphlet $g_1(V_1, E_1)$, edges $(u, v), (u', v') \in E_1$ are in same orbit if an automorphism f_{auto} of g_1 exists such that $u = f_{auto}(u')$ and $v = f_{auto}(v')$. ■

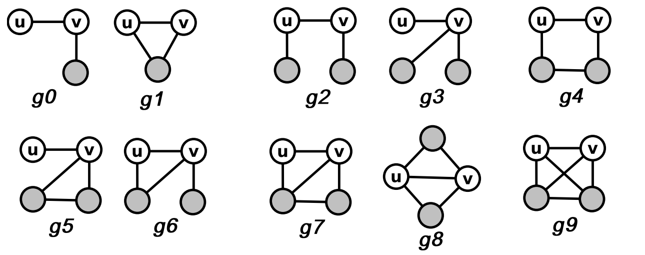


Fig. 1: 3,4-size local graphlets

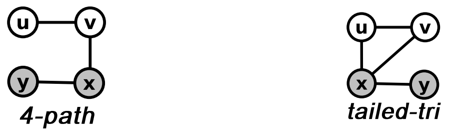


Fig. 2: Example of 4-size non local graphlets. Left: a non-local edge orbit of 4-path, structurally identical to $g2$; right: a non-local edge orbit of tailed-triangle, structurally identical to $g5$ and $g6$.

For the two-triangles structure, any two edges from the rectangle (excluding the diagonal edge) are in the same orbit as we can easily map incident nodes of the first edge to incident nodes of the second edge. Hence, selecting any edge from these four edges (rectangle) as (u, v) represents the exact same graphlet $g7$ in Figure 1. Notice that, if two edges are in the same orbit they must have the same pair of degrees

[1]E-CLOG stands for **E**dge **C**entric **Lo**cal **G**raphlet.

for their incident vertices. On the other hand, if two edges do not have the same pair of degrees then they are definitely in different orbits. For example, in Figure 1 graphlets $g7$ and $g8$ are separated based on the orbit of (u, v), here $d(u) = 2, d(v) = 3$ in $g7$ and $d(u) = 3, d(v) = 3$ in $g8$.

In this study, each local graphlet, separated by topological structure or by edge orbit of the given edge, is referred as a graphlet type or a graphlet, and we obtain its count. For instance in Figure 1, there are 10 ($g0 - g9$) graphlet types. Notice that, we do not count some graphlet types which have different edge orbits because they are not identified as a local graphlet. For example in Figure 2 both 4-path and tailed-triangle graphlets have different edge orbits than corresponding graphlet types in Figure 1, but they are not local because the vertex y in both the graphlets in Figure 2 is neither neighbor of u nor neighbor of v. Hence, we do not count frequency for such non-local graphlet types.

A. *Problem definition*

Given an edge (u, v) of an undirected graph $G(V, E)$, our goal is to count the number of appearances of each k $\{3, 4, 5\}$-size local graphlet type for the edge (u, v) in the graph G.

III. Proposed method

In existing works, there are two distinct philosophies for counting graphlets: counting by enumeration and counting with algebraic expressions. In the first philosophy, each of the graphlet instances are enumerated at least once. It becomes very costly for large graphs as the number of graphlets in these graphs easily exceeds hundreds of billions. On the other hand, counting with algebraic expression is cheap, as counting is performed by using a combinatorial approach. Unfortunately, combinatorial counting is easy for size-3 or size-4 graphlets, but it becomes difficult for size-5 graphlets because the number of graphlet configurations is substantially higher for the case of size-5 than size-4. Also, for the case of local graphlet counting, edge orbits need to be considered, which makes it even more difficult.

Our proposed method for local graphlet counting is a hybrid approach, which enumerates only a subset of local graphlets and obtains the count of the remaining local graphlets in constant time by using algebraic expression. Another key innovation of our method is that it utilizes the count of sub-graphlets to efficiently compute the count of a larger size graphlet, which contains the sub-graphlet. Thus, our method first counts the size-3 local graphlets, and then uses these to count size-4 local graphlets, and then iteratively uses those counts to count size-5 local graphlets. Finally, our proposed method uses a generic counting algorithm, which counts the frequency of different graphlets by setting the value of a small number of template variables, which makes the method easy to understand and implement. In Section III-A, we first discuss the method used to count 3 and 4-size local graphlet types. Then, in Section III-B, we discuss the method for counting 5-size graphlets. In Table I we show all the notations that are used in the discussion.

TABLE I: Summary of the notations

Notations	Meaning
(u, v)	Edge for which local graphlets are being computed
$d(u)$	Degree of the node u
$\Gamma(u)$	Set of neighboring nodes of the node u
T	Set of nodes creating triangles with edge (u, v)
N_u	Set of nodes that is only neighbor of u not v
N_v	Set of nodes that is only neighbor of v not u
i, j, k	Variables, instantiated as 3rd , 4th and 5th nodes (after u and v) of 5 size graphlets.
T_i	Set of neighboring nodes of i which are also in T
N_{ui}	Set of neighboring nodes of i which are also in N_u
N_{vi}	Set of neighboring nodes of i which are also in N_v
S_1, S_2, S_3	Set variables, which take set as a value, depending on the membership of i, j and k, respectively.
l_1, l_2, l_3	Binary variables, $l_1 = 1$ if i, j is connected, else 0 similarly, $l_2 = 1 \mid l_3 = 1$ if $i, k \mid j, k$ is connected, else 0.
$C(i, j)^{S_1, S_2, S_3}_{l_1, l_2, l_3}$	Count of set of graphlet type(s) for given i, j and d/different values of S_3, l_2, l_3.
t	Takes values from $\{1, 2, 3, 4\}$, based on values of l_2 and l_3 (1 for 11, 2 for 01, 3 for 10, and 4 for 00).
b_i	Bias value need to be deducted to maintain the $i \neq j \neq k$ property.
d_i, d	Dividing factor(s) to handle duplicate counting.

A. 3 *and* 4 *size local graphlet counting*

Given an edge (u, v), counting size-3 local graphlets is easy. There are only two such graphlets, open triple (g_0) and triangle (g_1), and both the structures have only one edge orbit. We count them from $\Gamma(u)$ and $\Gamma(v)$ by finding three disjoint sets of vertices: T, N_u, and N_v, where $T = \Gamma(u) \cap \Gamma(v)$, $N_u = \Gamma(u) \setminus T$, and $N_v = \Gamma(v) \setminus T$. Note that, vertices u and v are not included in any of these sets, i.e. $u, v \notin (T \cup N_u \cup N_v)$, this is because we consider only simple graph without any self loop so no vertex is a neighbor of itself. Also T, N_u and N_v are pair-wise disjoint i.e.

$$N_u \cap T = \phi \qquad \& \qquad N_v \cap T = \phi \qquad \& \qquad N_u \cap N_v = \phi$$

T contains the vertices which are neighbors of both u and v, thus forming a triangle and N_u contains neighbors of u which are not neighbors of v, so these vertices are terminal vertices of open triples centered at u, and identically, N_v contains the terminal vertices of open triples centered at v. Then $f0 = |N_u| + |N_v|$ and $f1 = |T|$. This completes the counting of size-3 local graphlets for the given edge (u, v).

There are total eight size-4 local graphlets, of which we enumerate 4 of them, and we obtain the count of the remaining graphlets in constant time by using algebraic expression. Besides, for efficient enumeration, we use the set T, N_u and N_v which we obtain while counting the size-3 local graphlets.

Counting $g9$ (4-clique): Say, a vertex $x \in T$ forms a triangle with (u, v). Now, if another vertex $y \in \Gamma(x)$ also forms a triangle with (u, v), the induced topology (u, v, x, y) forms a 4-clique. To avoid double counting, we use the condition that the identifier of x is higher than that of y.

Counting $g4$ (4-cycle): If $x \in N_u$, y is a neighbor of x, and $y \in N_v$, then (u, v, x, y) forms a 4-cycle.

Counting $g5$ (tailed-triangle): Based on edge orbits this structure includes two graphlet types $g5$ and $g6$, of which we enumerate $g5$ (tailed-triangle with (u, v) as tail), and obtain

$g6$ in constant time. If, $x \in N_u$, and y is a neighbor of x, and $y \in N_u$, then (v, u, x, y) forms a $g5$, in which v is the tail vertex. To avoid double counting, we use the constraint $x > y$. A symmetric enumeration where $x \in N_v$ obtains tailed-triangle with u as the tail vertex.

Counting $g7$ **(two-triangles):** This graphlet has two edge orbits, $g7$ and $g8$, of which we enumerate $g7$ and obtain $g8$ in constant time. If $x \in N_v$, and $y \in \Gamma(x)$, and $y \in T$, (u, y, x, v) forms a two triangle with (v, y) as the diagonal edge. A symmetric enumeration where $x \in N_u$ obtains two-triangles with (u, y) as the diagonal edge.

The frequencies of the remaining four graphlet types $(g2, g3, g6, g8)$ can be calculated in constant time by using the following equations:

$$f2 \leftarrow |N_u| \times |N_v| - f4 \tag{1}$$

$$f3 \leftarrow \binom{|N_u|}{2} + \binom{|N_v|}{2} - f5 \tag{2}$$

$$f6 \leftarrow |T| \times f0 - f7 \tag{3}$$

$$f8 \leftarrow \binom{|T|}{2} - f9 \tag{4}$$

The detailed steps for the method are available in Algorithm 1. In this algorithm we use f to represent the frequency vector that contains frequency of all the graphlet types (size-3, 4 and 5) and fN represents frequency of a specific graphlet type gN. In the algorithm lines 5-14 generates three distinct sets T, N_u and N_v. The map data structure, $status\_map$, maps a vertex to values α, β, or γ to represent the membership from sets T, N_u, or N_v, respectively. In the Algorithm 1, Lines 17-20 show iteration over elements of T and counting 4-clique frequency and Lines 21-28 (29-34) show iteration over elements of N_u (N_v) to count frequencies of graphlet types $g4, g5$, and $g7$.

B. 5*-size Local Graphlet Counting*

For counting 5-size local graphlet types, we utilize the enumeration of 4-size local graphlets. Given edge (u, v), and a 4-size local graphlet, we first obtain all feasible fifth nodes, and based on the connections of the fifth node with the 4-size graphlet nodes, we identify and count different 5-size graphlet types. This method appears straight-forward, but counting all 5-size local graphlet types efficiently while avoiding enumeration of all graphlet types is a challenging task. Our method enumerates only 14 size-5 local graphlet types and calculates other 18 local graphlet types in constant time using algebraic expressions. See the Table II.

A key observation regarding a size-5 local graphlet for a given edge (u, v) is that the remaining three vertices of this graphlet must be from T, N_u, or N_v. This is due to the definition of local edge-centric graphlet, which requires that the remaining three vertices to be neighbors of u, or v or both (see Definition 2). We denote these vertices as i, j and k, such that they are distinct, i.e., $i \neq j \neq k$; the 3^{rd} vertex is represented as i, the 4^{th} as j and the 5^{th} as k. Now, there are total 9 different combinations based on the joint membership of i, j, and k within the sets T, N_u, and N_v and each of these combinations lead to different sets of local graphlets. Besides, these combinations are exhaustive i.e., they cover each and every local edge-centric graphlet given the edge (u, v). Also, there is no false-positive, that is every choice of three vertices i, j, and k within these the sets T, N_u and N_v leads to a valid graphlet which we need to consider in our counting. So, for counting all size-5 local graphlets given (u, v), it suffices that we count all graphlets by efficiently enumerating i, j, and k over the three sets T, N_u and N_v.

To write a generic algorithm that enumerates and counts graphlets within each of the above 9 combinations, we use three selector variables S_1, S_2, S_3, which take values from sets T, N_u, and N_v based on the membership of i, j, k within these sets. For instance, for an enumeration, if $i, j, k \in T$ we have $S_1 = S_2 = S_3 = T$. Thus, values of S_1, S_2, and S_3 denote an equivalence class, and size-5 local graphlets within one equivalence class can be counted efficiently. Now, conditioned on the equivalence class, edges between the vertices $\{u, v\}$ and $\{i, j, k\}$ are already fixed. But, based on the existence of edges between i, j, and k one equivalence

Algorithm 1: GET_3 − 4_GRAPHLETCOUNT(G, u, v)

```
1:  initialize frequencies                                              // $f0 - f41 \leftarrow 0$
2:  initialize unique neighbor sets of $u$ and $v$                      // $N_u \leftarrow \{\}$ , $N_v \leftarrow \{\}$
3:  initialize common neighbor set                                      // $T \leftarrow \{\}$
4:  initialize status of each vertex in $G$    // for each $x$ do $status\_map(x) \leftarrow \phi$
5:  for each $x \in \Gamma(u)$ do
6:      if $x \neq v$ then
7:          $N_u \leftarrow N_u \cup x$, $status\_map(x) \leftarrow \alpha$ // $\alpha$ represents membership of $N_u$
8:  for each $x \in \Gamma(v)$ do
9:      if $x \neq u$ then
10:         if $status\_map(x) = \alpha$ then
11:             $T \leftarrow T \cup x$, $status\_map(x) \leftarrow \gamma$    // overwrite $\alpha$ with $\gamma$ (membership of $T$)
12:             $N_u \leftarrow N_u \setminus x$                // remove the element $x$ from $N_u$
13:         else
14:             $N_v \leftarrow N_v \cup x$, $status\_map(x) \leftarrow \beta$ // $\beta$ represents membership of $N_v$
15: $f0 \leftarrow |N_u| + |N_v|$             // number of unique neighbors (no triangles)
16: $f1 \leftarrow |T|$                                                  // number of triangles
17: for each $x \in T$ do
18:     for each $y \in \Gamma(x)$ do
19:         if $status\_map(y) == \gamma$ and $y < x$ then
20:             $f9 \leftarrow f9 + 1$                                   // 4clique
21: for each $x \in N_u$ do
22:     for each $y \in \Gamma(x)$ do
23:         if $status\_map(y) == \alpha$ and $y < x$ then
24:             $f5 \leftarrow f5 + 1$                    // tail-triangle with $(u, v)$ as tail
25:         else if $status\_map(y) == \beta$ then
26:             $f4 \leftarrow f4 + 1$                                   // 4cycle
27:         else if $status\_map(y) == \gamma$ then
28:             $f7 \leftarrow f7 + 1$           // 2triangles with $(u, y)$ as diagonal link
29: for each $x \in N_v$ do
30:     for each $y \in \Gamma(x)$ do
31:         if $status\_map(y) == \beta$ and $y < x$ then
32:             $f5 \leftarrow f5 + 1$                    // tail-triangle with $(u, v)$ as tail
33:         else if $status\_map(y) == \gamma$ then
34:             $f7 \leftarrow f7 + 1$           // 2triangles with $(v, y)$ as diagonal link
35: calculate $f2, f3, f6, f8$ using Equations 1, 2, 3, 4.
36: return $f, T, N_u, N_v, status\_map$
```

TABLE II: List of enumerating and non-enumerating 5-size local graphlet types

Enumerated	$g11, g14, g17, g20, g22, g23, g26, g27, g28, g33, g34, g37, g39, g41$
Not enumerated	$g10, g12, g13, g15, g16, g18, g19, g21, g24, g25, g29, g30, g31, g32, g35, g36, g38, g40$

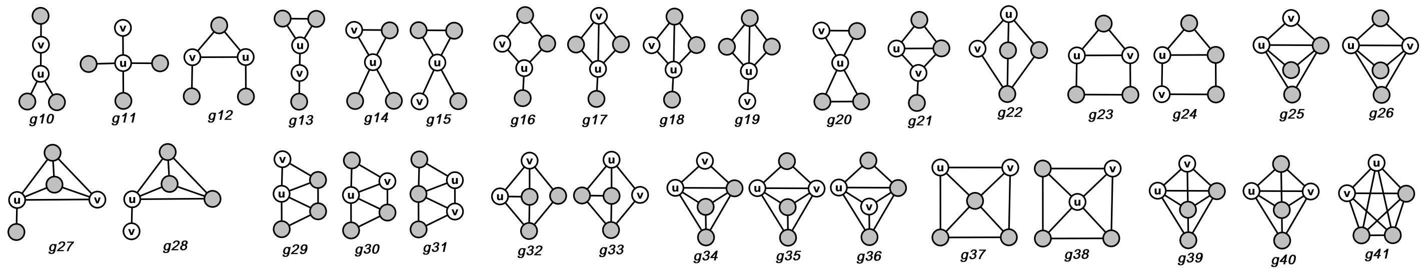
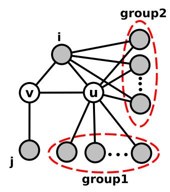
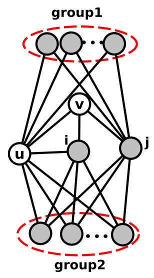


Fig. 3: 5-size local graphlets



i: Example of graphlet types created using variable $l_2 = 0/1$ and fixed $S_3 = N_u$ and $l_3 = 0$



ii: Example of graphlet types created using variable $l_2 = 0/1$ and fixed $S_3 = N_u$ and $l_3 = 1$

Fig. 4: Illustration of how different values of l_2 and l_3 generates different graphlet types

class may lead to more than one local graphlet. We use the binary variable $l_1, l_2, l_3 \in \{1, 0\}$ for denoting edge existence between i, j, and k; $l_1 = 1$ if $(i, j) \in E$ else 0, $l_2 = 1$ if $(i, k) \in E$ else 0 and $l_3 = 1$ if $(j, k) \in E$ else 0. This leads to 8 possible choices within an equivalent class.

From the above discussion, we can obtain a simple size-5 local graphlet enumerator, using three nested for loops for vertices i, j, and k, each iterating over the set T, N_u and N_v and within the body of the innermost for loop, the enumerator determines the graphlet type based on the value of l_1, l_2, and l_3. However, such a method enumerates all local graphlet instances and hence is not efficient.

Our approach for counting 5-size graphlets is that we enumerate over each 4-size graphlets and then without enumeration we count the number of different graphlets based on the topological disposition of possible 5^{th} node. This saves a large number of enumerations and yields a vastly improved local graphlet counting algorithm. Besides, the degree of freedom of search space becomes smaller, because when a 4-graphlet is given, the third and fourth vertices, i, j are fixed, and the values of S_1, S_2 and l_1 are known; then the values of S_3, l_2 and l_3 decide the specific type of the size-5 graphlet.

Example: In Figure 4, we show a specific enumeration, in which $i \in T$ and $j \in N_v$, and we would like to count all local graphlets for which the fifth vertex k belongs to N_u, i.e., the selector variable, $S_3 = N_u$. The possible candidates for the vertex k are shown within dotted ovals. Also, in this example the vertex i, and j are not connected, so $l_1 = 0$. Now, the four possible values of l_2, l_3 create four different types of size-5 graphlets, which are $g12, g21, g23$, and $g32$. The left figure shows the case for $l_3 = 0$ (k is not connected with j) and the right figure shows the case for $l_3 = 1$ (k and j are connected). For both the left and the right figures, the vertices shown in the dotted oval labeled as “group1” exhibit $l_2 = 0$ and produce $g12$ (on left figure) and $g23$ (on the right figure). Likewise, the vertices in “group2” stands for $l_2 = 1$ and they produce $g21$ (on left figure) and $g32$ (on right figure). ■

To facilitate effective counting by iterating over 4-graphlets, we compute a few more vertex-sets beyond T, N_u and N_v, by utilizing $\Gamma(i)$, the adjacency vector of the third vertex i. For different values of l_1, the 4^{th} vertex j belongs to different subsets of the sets, T, N_u or N_v. For example, if $S_2 = T$, then for $l_1 = 1$, j must belong to $\Gamma(i) \cap T$ and for $l_1 = 0$, j belong to $T \setminus \Gamma(i)$. We represent these sets as below:

$$T_i = \Gamma(i) \cap T, \qquad N_{ui} = \Gamma(i) \cap N_u, \qquad N_{vi} = \Gamma(i) \cap N_v$$
$$\overline{T_i} = T \setminus \Gamma(i), \qquad \overline{N_{ui}} = N_u \setminus \Gamma(i), \qquad \overline{N_{vi}} = N_v \setminus \Gamma(i)$$

In our template algorithm we refer to the above sets by using their selector variable. For example, for the selector variable S_3, we will write the set as S_{3i} which is equal to $\Gamma(i) \cap S_3$; thus S_{3i} can be T_i, or N_{ui} or N_{vi} depending on whether S_3 is T, N_u or N_v.

As mentioned earlier, for a given 4-size graphlets, say, (u, v, i, j), we count (without enumeration) the number of different graphlets based on the topological disposition of possible 5^{th} node . To facilitate this, we define the term $C(i, j)^{S_1,S_2,S_3}_{l_1,l_2,l_3}$. It represents the total count of size-5 graphlets which contain (u, v, i, j) as their sub-graphlet. In $C(i, j)^{S_1,S_2,S_3}_{l_1,l_2,l_3}$, i and j are given, thus S_1, S_2, and l_1 are already fixed. Thus the value of $C(i, j)^{S_1,S_2,S_3}_{l_1,l_2,l_3}$ is the number of 5-graphlets for the assigned values of S_3, l_2 and l_3. By fixing these three variables, we can obtain the count of a specific graphlet. For example, $C(i, j)^{S_1,S_2,S_3}_{l_1,1,0}$ represents the count of a graphlet type generated from the current i, j vertices and all possible $k \in S_3$ such that $l_2 = 1$, $l_3 = 0$. More specific example is $C(i, j)^{T,T,T}_{1,1,1}$, which represents the count of 5-size cliques for given i, j values. Because, here all three vertices i, j, k are connected to both u and v and they themselves are also pair-wise connected ($l_1 = l_2 = l_3 = 1$), which creates a fully connected size-5 subgraph (a 5-clique). Similarly, $C(i, j)^{T,T,T}_{1,0,1}$ represents count of graphlet type $g40$ for given i, j values.

For unknown values of l_2 and/or l_3, we use $*$, hence total count of graphlet types for given values of i, j, S_3 and l_2 but unknown value of l_3 can be represented as $C(i, j)^{S_1,S_2,S_3}_{l_1,l_2,*}$. For example, total count of graphlet types $\{g12, g18\}$ in

Figure 4i can be represented as $C(i,j)^{T,N_v,N_u}_{0,*,0}$ and total count from Figure 4ii is $C(i,j)^{T,N_v,N_u}_{0,*,1}$. Similarly, if both l_2 and l_3 are unknown then we use $C(i,j)^{S_1,S_2,S_3}_{l_1,*,*}$, so $C(i,j)^{T,N_v,N_u}_{0,*,*}$ is the total count for all the four kinds of graphlets in Figure 4. Now the value of $C(i,j)^{S_1,S_2,S_3}_{l_1,*,*}$ can be obtained by using the following theorem.

Theorem 1:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,*,*} = \left\{\begin{array}{ll} |S_3|, & \text{if } (S_3 \neq S_1 \wedge S_3 \neq S_2) \\ |S_3|-1, & \text{if } ((S_3 = S_1 \wedge S_3 \neq S_2) \\ & \quad \vee (S_3 \neq S1 \wedge S_3 = S_2)) \\ |S_3|-2, & \text{if } (S_3 = S_1 \wedge S_3 = S_2) \end{array}\right\}$$

Proof:

For each value of $k \in S_3$, the induced graphlet (u,v,i,j,k) matches a specific local graphlet type. However, it will always add a count to $C(i,j)^{S_1,S_2,S_3}_{l_1,*,*}$ regardless of the graphlet type. Hence, the count of $C(i,j)^{S_1,S_2,S_3}_{l_1,*,*}$ is the same as the cardinality of S_3. Now, if vertices i and/or j are also from the same set as S_3 i.e. $S_1 = S_3$ and/or $S_2 = S_3$ (note that S_1, S_2, S_3 can take only three different values from $\{T, N_u, N_v\}$, and they can have same values), then we need to deduct the total count by 1 if only i or only j is from the same set and deduct the count by 2 if both i,j are from the same set as S_3; total count reduces by 1 (or 2) because all three vertices (i,j,k) are distinct $(i \neq j \neq k)$. ■

For the case when the value of l_2, or l_3 is also known, we have the following theorem.

Theorem 2:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,1,*} = \begin{cases} |S_{3i}|-1, & \text{if } (S_3 = S_2 \wedge l_1 = 1) \\ |S_{3i}|, & \text{otherwise} \end{cases}$$

$$C(i,j)^{S_1,S_2,S_3}_{l_1,*,1} = \begin{cases} |S_{3j}|-1, & \text{if } (S_3 = S_1 \wedge l_1 = 1) \\ |S_{3j}|, & \text{otherwise} \end{cases}$$

Proof:

For $C(i,j)^{S_1,S_2,S_3}_{l_1,1,*}$, we know that $k \in S_3 \cap \Gamma(i)$, hence total possible count is equal to the size of the set $S_{3i} = S_3 \cap \Gamma(i)$. We also need to ensure that $j \neq k$, hence if $S_2 = S_3$ and j is also a neighbor of i (to make a symmetry to the condition that k is a neighbor of i), we need to deduct the count by one. A similar argument also holds for $C(i,j)^{S_1,S_2,S_3}_{l_1,*,1}$. ■

Now, the following Theorem and corollaries help us to count several graphlets in constant time simply by using algebraic expression.

Theorem 3:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,*,*} = C(i,j)^{S_1,S_2,S_3}_{l_1,0,0} + C(i,j)^{S_1,S_2,S_3}_{l_1,0,1} + C(i,j)^{S_1,S_2,S_3}_{l_1,1,0} + C(i,j)^{S_1,S_2,S_3}_{l_1,1,1}$$

Corollary 1:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,0,0} = C(i,j)^{S_1,S_2,S_3}_{l_1,*,*} - C(i,j)^{S_1,S_2,S_3}_{l_1,*,1} - C(i,j)^{S_1,S_2,S_3}_{l_1,1,*} + C(i,j)^{S_1,S_2,S_3}_{l_1,1,1}$$

Corollary 2:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,1,0} = C(i,j)^{S_1,S_2,S_3}_{l_1,1,*} - C(i,j)^{S_1,S_2,S_3}_{l_1,1,1}$$

Corollary 3:

$$C(i,j)^{S_1,S_2,S_3}_{l_1,0,1} = C(i,j)^{S_1,S_2,S_3}_{l_1,*,1} - C(i,j)^{S_1,S_2,S_3}_{l_1,1,1}$$

Algorithm 2: Template Algorithm for N^{th} Graphlet

```
1: assign $S_1$, $S_2$, $S_3$ to the right set for graphlet $N$ from Table III
2: select $\mathbf{t} \in \{1,2,3,4\}$, $b_i$, $d_i$, and $d$ for graphlet $N$from Table III
3: $\langle v1_i, v2_i, v3_i, v4_i \rangle \leftarrow \mathbf{0}$                // initialization
4: for all $i \in S_1$ do
5:     $\langle v1^{old}_{ij}, v2^{old}_{ij}, v3^{old}_{ij}, v4^{old}_{ij} \rangle \leftarrow \mathbf{0}$      // initialization
6:     for all $j \in S_2$ do
7:         $\langle v1^{new}_{ij}, v2^{new}_{ij}, v3^{new}_{ij}, v4^{new}_{ij} \rangle \leftarrow$
               $\langle v1^{old}_{ij}, v2^{old}_{ij}, v3^{old}_{ij}, v4^{old}_{ij} \rangle$+ get_freq_k$(i,j,S_3)$
8:         $\langle v1^{old}_{ij}, \cdots, v4^{old}_{ij} \rangle \leftarrow \langle v1^{new}_{ij}, \cdots, v4^{new}_{ij} \rangle$
       // end of loop for $j$
9:     $v\mathbf{t}_i \leftarrow v\mathbf{t}_i + (v\mathbf{t}^{new}_{ij} - b_i)/d_i$
   // end of loop for $i$
10: $fN \leftarrow fN + v\mathbf{t}_i/d$
```

Algorithm 3: get_freq_k(i,j,S_3)

```
1: $S_{3ij} \leftarrow S_{3i} \cap S_{3j}$
2: $v1 = |S_{3ij}|$
3: $v2 = |S_{3i}| - |S_{3ij}|$
4: $v3 = |S_{3j}| - |S_{3ij}|$
5: $v4 = |S_3| - |S_{3i}| - |S_{3j}| + |S_{3ij}|$
6: return $\langle v1, v2, v3, v4 \rangle$
```

C. Generic Counting Algorithm

In Algorithm 2, we provide a generic algorithm for counting all size-5 local graphlets. In this generic algorithm, we consider the following variables, $S_1, S_2, S_3, t, b_i, d_i$ and d as template variables and a specific set of values for these variables gives the frequency of a specific graphlet type. Among these S_1, S_2 and S_3 are selector variables (for vertices i, j and k), $\mathbf{t}$ is an integer between 1 and 4 depending on the joint value of l_2 and l_3, b_i is the bias which is the count adjustment when multiple selector variables have the same value. The bias values are computed by using Theorem 1 and Theorem 2. Even after addressing for the obvious duplication by maintaining order among the vertices i, j, and k, some graphlets are generated multiple times, d_i and d are normalizing factors to ensure that each of the local graphlets are counted once and exactly once.

The detailed information on graphlet type and the associated values of the template variables are shown in Table III. As shown in this table, the variable S_1 takes values from N_u or T, S_2 takes values conditioning on the fact whether the second vertex is adjacent to first vertex or not. So, it takes values from T_i, N_{ui}, N_{vi} or their complements $\overline{T_i}$, $\overline{N_{ui}}$, $\overline{N_{vi}}$. And S_3 takes value from N_u, N_v, or T. The bias and normalizing factor values are also shown in this table. In this table, we also have three other variables, S_{1r}, S_{2r} and S_{3r}. They will be discussed in Section III-D.

Algorithm 2 uses a sub-routine, Algorithm 3, where $|S_{3ij}|$ represents value of $C(i,j)^{S_1,S_2,S_3}_{l_1,1,1}$. Using Theorem 2, Corollary 2 and 3, we calculate frequencies of specific graphlet types represented as $f2_{ij}$ and $f3_{ij}$ in the algorithm.

Similarly, line 5 of the algorithm gives frequency of a graphlet type where k is not connected to either i or j using Theorem 1 and Corollary 1. The deduction from the total count (in both Theorem 1 and 2) for maintaining $i \neq j \neq k$ property, is adopted as a bias value b_i in the Algorithm 2.

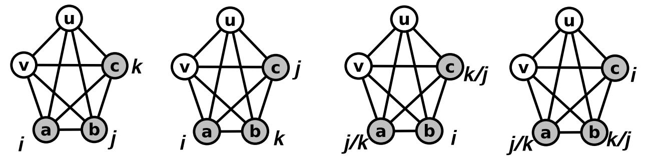
Example: Frequency of the 5-size clique i.e. graphlet type $g41$ can be calculated using template variable values $S_1 = T$, $S_2 = T_i$, $S_3 = T$, **t**$= 1$, $b_i = 0$, $d_i = 2$ and $d = 3$. For 5 size clique, as shown in the Figure 5, three vertices (a, b, c) other than u and v need to be from set T, hence $S_1 = T$, $S_2 = T$ and $S_3 = T$ and all three vertices need to be interconnected, so we use $S_2 = T_i$ ($l_1 = 1$) and $\mathbf{t} = 1$ ($f1_{ij} \Rightarrow l_2 = 1, l_3 = 1$). Now, as we can see for each i, selection of j and k are interchangeable i.e. in Figure 5i j and k are interchangeable between vertices b and c. Therefore this graphlet will be counted twice for each i, which leads to $d_i = 2$. Similarly, in Figure 5ii, we can see that we select all the 3 vertices (a, b, c) as an i one by one (Algorithm 2: line 1), hence we need to divide the total count by 3 i.e. $d = 3$.

D. Counting frequency of non-symmetric local graphlets

Definition 6 (Vertex orbit): For a given graphlet $g(V, E)$, vertexes $u, v \in V$ are in same orbit if an automorphism f_{auto} of g exists such that $u = f_{auto}(v)$. ■

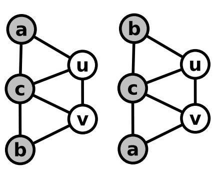
Definition 7 (Symmetric local graphlets): For a given edge (u, v) a local graphlet g is called symmetric if vertices u and v are in the same vertex orbit for g. ■

For symmetric local graphlets we do not need to count frequency for reverse sequence of the vertices v, u (instead of u, v). For a symmetric local graphlet, $d(u) = d(v)$ and also graphlet structure remains the same after exchanging neighborhood of the vertices u and v. For example, as shown in Figure 6i for graphlet type $g31$, $d(u) = d(v) = 3$ and when we exchange neighbors of u and v the graphlet structure remains the same, because vertex u and v are in the same orbit. On the other hand, Figure 6ii shows two examples of non-symmetric graphlets of type $g19$ and $g27$.

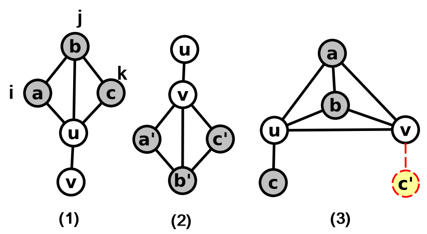


i: 5 clique graphlet $(i = a)$ ii: 5 clique graphlet $(i = b/c)$

Fig. 5: 5 clique graphlet counting



i: Example of edge Symmetric graphlet



ii: Examples of edge non-symmetric graphlets

Fig. 6: Example of edge symmetric and non-symmetric graphlets

TABLE III: Set of values for template variables to count various 5 size local graphlets

graphlet	S_1	S_2	S_3	**t**	b_i	d_i	d	S_{1r}	S_{2r}	S_{3r}
$g10$	N_u	$\overline{N_{ui}}$	N_v	4	0	1	2	N_v	$\overline{N_{vi}}$	N_u
$g11$	N_u	$\overline{N_{ui}}$	N_u	4	$2 \cdot \lvert S_2 \rvert$	2	3	N_v	$\overline{N_{vi}}$	N_v
$g12$	T	$\overline{N_{ui}}$	N_v	4	0	1	1	–	–	–
$g13$	N_u	N_{ui}	N_v	4	0	1	2	N_v	N_{vi}	N_u
$g14$	T	$\overline{N_{ui}}$	N_u	4	$\lvert S_2 \rvert$	2	1	T	$\overline{N_{ui}}$	N_v
$g15$	N_u	N_{ui}	N_u	4	0	1	2	N_v	N_{vi}	N_v
$g16$	N_u	$\overline{N_{ui}}$	N_v	3	0	1	1	N_v	$\overline{N_{vi}}$	N_u
$g17$	T	$\overline{T_i}$	$N_u + N_v$	4	0	1	2	–	–	–
$g18$	T	N_{ui}	N_u	4	0	1	1	T	N_{vi}	N_v
$g19$	N_u	N_{ui}	N_u	3	$\lvert S_2 \rvert$	1	2	N_v	N_{vi}	N_v
$g20$	N_u	N_{ui}	T	4	0	1	2	N_v	N_{vi}	T
$g21$	T	N_{ui}	N_v	4	0	1	1	T	N_{vi}	N_u
$g22$	N_u	$\overline{N_{ui}}$	N_v	1	0	1	2	N_v	$\overline{N_{vi}}$	N_u
$g23$	T	$\overline{N_{ui}}$	N_v	3	0	1	1	–	–	–
$g24$	N_u	N_{ui}	N_v	3	0	1	1	N_v	N_{vi}	N_u
$g25$	T	N_{ui}	N_u	2	0	2	1	T	N_{vi}	N_v
$g26$	T	$\overline{T_i}$	T	4	$2 \cdot \lvert S_2 \rvert$	2	3	–	–	–
$g27$	T	T_i	$N_u + N_v$	4	0	1	2	–	–	–
$g28$	N_u	N_{ui}	N_u	1	0	2	3	N_v	N_{vi}	N_v
$g29$	T	N_{ui}	N_u	3	0	1	1	T	N_{vi}	N_v
$g30$	T	N_{ui}	T	4	0	1	1	T	N_{vi}	T
$g31$	T	N_{ui}	N_v	2	0	1	1	–	–	–
$g32$	T	N_{ui}	N_v	3	0	1	1	T	N_{vi}	N_u
$g33$	N_u	N_{ui}	N_v	1	0	1	2	N_v	N_{vi}	N_u
$g34$	T	N_{ui}	N_u	1	0	2	1	T	N_{vi}	N_v
$g35$	T	T_i	T	4	0	1	2	–	–	–
$g36$	T	N_{ui}	T	2	0	1	1	T	N_{vi}	T
$g37$	T	N_{ui}	N_v	1	0	1	1	–	–	–
$g38$	T	N_{ui}	T	3	$\lvert S_2 \rvert$	1	2	T	N_{vi}	T
$g39$	T	N_{ui}	T	1	0	1	2	T	N_{vi}	T
$g40$	T	T_i	T	3	$\lvert S_2 \rvert$	1	2	–	–	–
$g41$	T	T_i	T	1	0	2	3	–	–	–

To calculate the correct frequency for non-symmetric graphlets, we need to count occurrence of the graphlet with reverse sequence of the vertices. For example, in Figure 6ii(1) a, b and c all are neighbors of u and $d(u) = 4$. However to count the correct frequency of the graphlet type $g19$, we need to count frequency for the graphlet shown in Figure 6ii(2) where $d(v) = 4$ and a', b' and c' all are neighbors of v. To count the correct frequency for the graphlet in Figure 6ii(1), template variables take values $S_1 = N_u$, $S_2 = N_{ui}$ and $S_3 = N_u$. But to count frequency of the graphlet in Figure 6ii(2), the variable values become $S_{1r} = N_v$, $S_{2r} = N_{vi}$ and $S_{3r} = N_v$, where S_{1r} represents reverse version of S_1 as shown in Table III. Template variables t, b_i, d_i and d remain the same for both cases. Because when we calculate $f3_{ij}$ using Algorithm 3, N_{uj} (S_{3j}) also includes vertex represented by i (here a), hence we need to subtract 1 from $f3_{ij}$ for each j. We subtract this value combined using bias $b_i = |S_2|$. Lastly, we also count the same graphlet two times considering $i = a$ and $i = c$, hence we divide the global count by $d = 2$.

Another example of non-symmetric graphlet is graphlet type $g27$. To count correct frequency for this type, we just need to sum up possible unique neighbors of u and unique neighbors of v (with red dotted line). To calculate both cases as shown in Table III, we just need to put two different values (N_u and N_v) of template variable S_3 and sum the resultant frequency values.

E. complexity

For 5-size local graphlets counting, any regular algorithm generally takes $O(\Delta^3)$ time for each edge [1], where Δ is maximum degree in the graph G. However, time complexity of our method for the same (Algorithm 2) is $O((T^{max} + N_u^{max} + N_v^{max})^3)$, where T^{max} is the largest value of $|T|$ out of all edges of the graph. Similarly N_u^{max} and N_v^{max} represent largest value of $|N_u|$ and $|N_v|$ respectively. We can claim that for real world networks $O((T^{max} + N_u^{max} + N_v^{max})^3) < O(\Delta^3)$, because for any edge (u, v), $|N_u| = d(u) - |T|$ and $|N_v| = d(v) - |T|$, and for any real-world sparse network $0 \leq T^{max} \ll \Delta$. Also, for any node with highest degree there is very low probability that $|T| = 0$, hence $|T| > 0$ and $N_u^{max} < \Delta$ and $N_v^{max} < \Delta$.

F. *Parallel version of* E-CLOG

E-CLOG is embarrassingly parallelizable, as the main work is performed over two for loops and operations inside the loops are independent for each iteration. Additionally, each edge can have unique independent data structure (T, N_u, N_v) values, hence parallel computation over edges is highly effective for large number of edges. We use only second strategy in our implementation.

IV. EXPERIMENTS AND RESULTS

We conducted three different experiments to show efficiency, scalability and usability of the proposed method. In the first experiment, we compare running time of our method with GRAFT [1]. In the second experiment we show that, nearly linear speed-up can be achieved for the parallel version of the E-CLOG. Lastly, in the third experiment, we show the utility of local graphlet frequencies for solving link prediction problem.

We collected 18 different graph datasets from different domains from KONECT[2]. We have 3 biological networks: ARENAS-META is a metabolic network, MAAYAN-VIDAL and REACTOME are protein interaction networks. *Arenas-png* is an interaction network of users of the Pretty Good Privacy (PGP) algorithm. AC-CAIDA and TOPOLOGY are networks of autonomous systems of the Internet. CA-ASTROPH and COM-DBLP are co-authorship graphs, and COM-AMAZON is a co-purchase network of Amazon. There are 4 different types of online social networks: DOUBAN is an online recommendation based social network, FACEBOOK-WOSN is an online friendship network, LOC-BRIGHTKITE is a location based social network and PETSTER-HAMSTER is a friendship network of pet owners. We have few infrastructure based networks, such as OPSAHL-POWERGRID is a power-grid network, and three road networks ROADNET-CA, ROADNET-PA and ROADNET-TX for three different states of the USA. WORDNET-WORDS is a lexical network of words from the WordNet dataset. We generated two synthetic networks[3] using a small-world phenomenon to check scalability of our method for huge networks with higher average degree. Some basic statistics such as number of vertices ($|V|$), number of edges ($|E|$), and average degree ($Avg.Deg$) for the datasets are shown in the Table IV.

[2]Koblenz Network Collection: konect.uni-koblenz.de/

[3]NetworkX-Python Library: networkx.github.io/

TABLE IV: Dataset statistics

Datasets	$\vert V\vert$	$\vert E\vert$	$Avg.Deg$
ARENAS-META	453	2,025	8.94
MAAYAN-VIDAL	3,023	6,149	4.07
OPSAHL-POWERGRID	4,941	6,594	2.67
PETSTER-HAMSTER	2,426	16,630	13.71
ARENAS-PGP	10,680	24,316	4.55
AC-CAIDA	26,475	53,381	4.03
TOPOLOGY	34,761	107,719	6.20
REACTOME	6,229	146,160	46.93
CA-ASTROPH	18,771	198,050	21.10
LOC-BRIGHTKITE	58,228	214,078	7.35
DOUBAN	154,908	327,161	4.22
WORDNET-WORDS	146,005	656,999	9.00
FACEBOOK-WOSN	63,731	817,035	25.64
COM-AMAZON	334,863	925,872	5.53
COM-DBLP	317,080	1,049,866	6.62
ROADNET-CA	1,965,206	2,766,607	2.82
ROADNET-PA	1,088,092	1,541,898	2.83
ROADNET-TX	1,379,917	1,921,660	2.79
SMALLWORLD-1M	1,000,000	7,499,767	14.99
SMALLWORLD-5M	5,000,000	37,501,514	15.00

A. *Runtime Comparison*

There exist no methods that perform local edge-centric graphlet counting for size-5 graphlets. There exist two recent global methods for exact counting of size-5 graphlets: GRAFT [1], and ESCAPE [23]; from which, GRAFT iterates over each of the edges of the input graph and aggregates the sub-counts of the graphlets that are incident to the edge of that iteration. At the end, it divides the duplicity factors of each graphlet to obtain the global graphlet counts. Thus GRAFT, indirectly, is an edge centric local graphlet counting method, which produces count of local 5-graphlets for each edge. In this experiment, we compare GRAFT's running time with E-CLOG's running time by finding the total time of computing local graphlets over all the edges. Note that, This comparison is a bit unfair for E-CLOG as it generates frequencies of local 5-graphlets of all edge orbits, totalling 32 graphlets, but GRAFT generates frequency of only 21 size-5 graphlets. Also note, among the 21 topologies that GRAFT counts, two topologies 5-path and 5-cycle are not counted by E-CLOG as they are not local graphlets as per definition. For this comparison, we extend E-CLOG and compute these two counts also[4]. Lastly, GRAFT does not provide parallel implementation, hence we use single thread (serial) computation for our method for fair comparison. The other exact global graphlet counting method, ESCAPE, does not iterate over the edges so it cannot produce counts for local edge graphlets and is not comparable with our method.

Table V shows the runtime comparison between E-CLOG and GRAFT over all 20 different graph datasets listed in Table IV. First column is the dataset name, the second and third columns show timing for the E-CLOG and the GRAFT, respectively. The reported time for both the methods is the time for counting local graphlets for all edges for each dataset after averaging the run-time over 5 runs[5]. For a better understanding, we write different time units (such as, **s** for seconds, **m** for minutes, and so on.) besides the time

[4]Algorithm is omitted because of space limitation.

[5]Single run timing for datasets with un-finished run of the GRAFT.

TABLE V: Runtime comparison between E-CLoG and GRAFT (**d** = days, **h** = hours, **m** = minutes, **s** = seconds)

Datasets	E-CLoG (serial)	GRAFT	E-CLoG (parallel)
ARENAS-META	5.2 **s**	2.67 **m**	0.36 **s**
MAAYAN-VIDAL	1.71 **s**	47.39 **s**	0.11 **s**
OPSAHL-POWERGRID	0.0556 **s**	0.3292 **s**	0.01 **s**
PETSTER-HAMSTER	43.44 **s**	54.79 **m**	3.24 **s**
ARENAS-PGP	14.6 **s**	13.49 **m**	1.06 **s**
AC-CAIDA	160.98 **m**	> 8.51 **d**	8.56 **m**
TOPOLOGY	510.7 **m**	> 7.15 **d**	26.17 **m**
REACTOME	98.9 **m**	> 7.15 **d**	6.72 **m**
CA-ASTROPH	23.51 **m**	3 **d**	1.58 **m**
LOC-BRIGHTKITE	43.93 **m**	> 7.15 **d**	2.54 **m**
DOUBAN	5.41 **m**	10.02 **h**	15.43 **s**
WORDNET-WORDS	93.13 **m**	> 4.01 **d**	5.43 **m**
FACEBOOK-WOSN	210.94 **m**	> 4.01 **d**	13.03 **m**
COM-AMAZON	4.3 **m**	9.82 **h**	24.22 **s**
COM-DBLP	14.32 **m**	1.56 **d**	56.31 **s**
ROADNET-CA	17.71 **m**	2.2 **m**	5.1 **m**
ROADNET-PA	5.52 **m**	1.07 **m**	1.57 **m**
ROADNET-TX	8.64 **m**	1.38 **m**	2.49 **m**
SMALLWORLD-1M	20.582 **m**	> 3.07 **d**	64.69 **s**
SMALLWORLD-5M	226.45 **m**	> 3.06 **d**	43.89 **m**

values. As we can see, for most of the datasets E-CLOG runs one or two orders of magnitude faster than GRAFT. For 8 datasets, GRAFT is unable to complete the counting even after few days (> in the Table V), while E-CLOG completed the counting in few hours on those datasets. A specific example can be WORDNET-WORDS dataset, for which E-CLOG took only 93.13 minutes (= 1.5 hours), but GRAFT did not finish in 4 days! By using combinatorial approach, the E-CLOG avoids enumeration of more than a half of the graphlets and that contributes significantly towards its speed. In 17 of 20 graphs, E-CLOG performs very good. Other 3 graphs in which GRAFT did better than E-CLOG, interestingly, are all road networks. A possible reason for this is during enumeration GRAFT first aligns an edge of a tree graphlet with the given edge, and then performs costly checks for counting cyclic graphlets. We found that these road networks are mostly tree networks, hence they run very fast on GRAFT. However, most of the other real-life networks have a substantial number of cycles, for those graphs, GRAFT is very poor.

In this table, we also show the runtime of the parallel version of E-CLOG in Column 4. This parallel version uses 72 threads. For most of the graphs, the parallel version improves the runtime significantly. For instance, for facebook graph the parallel version runs in 13 minutes whereas the single-thread version runs in 211 minutes.

B. Scalability

This section investigates the parallel performance of our proposed algorithm. The parallel algorithm for local graphlet counting has lock-free updates due to the partitioning of edges across the processing units and each edge is guaranteed to be processed by a single worker. For these experiments, we used a machine with two Intel Xeon E5-2699 v3 platform with 2.30GHz CPUs. Each processor has 18 cores with 46MB of L3 cache and 256KB of L2 cache. The machine has 256GB of memory, however, E-CLOG never came close to using all of it. E-CLOG scales well as the number of processing units increase. In particular, strong scaling is observed in Figure 7 for the 5 graphs having the worst running time for serial execution. We obtain 14x–20x speedup using 70 threads for most graphs.

TABLE VI: Comparison results for Link Prediction Problem

	ROC-AUC		PR-AUC	
Datasets	Topo-feat	LGFD	Topo-feat	LGFD
FACEBOOK-WOSN	0.5519	**0.9101**	0.6784	**0.9351**
TOPOLOGY	0.8179	**0.9366**	0.8837	**0.9415**
DBLP	0.6897	**0.7411**	0.7703	**0.7822**

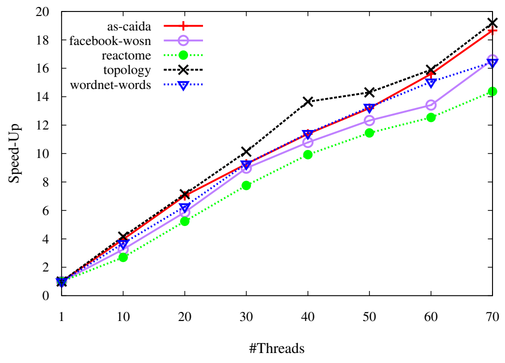


Fig. 7: Strong scaling results for a variety of graphs. We obtain 14x–20x speedup using 70 threads.

C. Link Prediction

Given a pair of non-adjacent vertices u and v in a social network, the goal of the link prediction task is to predict whether the vertices will form a link in future. In a supervised classification set-up, link prediction is typically solved by using a fixed set of topological features which determine the topological similarity between u and v. In this experiment we will demonstrate that local graphlet frequency distribution (LGFD) for the edge (u, v) captures topological properties (around the vertices u, v) that have substantially more predictive power than the traditionally used topological features for link prediction.

For this experiment, we use three datasets which are time-stamped networks. From Table IV, only two datasets (FACEBOOK-WOSN and TOPOLOGY) have time-stamps affiliated with edges. Here, FACEBOOK-WOSN has $333,923$ unique time-stamps and TOPOLOGY has $23,768$ unique time stamps, which we use. We also create DBLP co-authorship network with time-stamps (yearly) from AMiner [`https://aminer.org/data`]. To create this network, we select a set of authors who have published 2 or more papers in database and data mining conferences and from these selected authors we generate induced co-authorship graph. This network has $4,545$ authors with $20,491$ connections over 45 years.

For experiment, we divide the time-stamps of each dataset into three chronologically ordered partitions, network growing period, train period and test period; from the beginning up to 70% of total time-stamps is network growing period, 70% to 85% is train period, and from 85% till the end is the test period. An edge created in train period is a positive train instance and an edge created (for the first time) during

TABLE VII: Useful graphlets for link prediction

Datasets	Graphlets with high individual AUC
FACEBOOK-WOSN	$g0, g5, g12, g14, g16, g18, g20, g21, g25, g29, g34.$
TOPOLOGY	$g0, g5, g11, g14, g15, g18, g19, g20, g25, g29, g34.$
DBLP	$g0, g2, g5, g12, g14, g18, g20, g21, g25, g27, g29, g34, g36, g39.$

this test period is a positive test instance. We take random node pairs which do not have a connecting edge till the end of the training period as negative train instances and random disconnected node pairs as negative test instances. In this experiment, we use local graphlet frequencies normalized over all local graphlet types (42) as a features set. For comparison with these local graphlet frequencies, we use 10 traditional topological features : number of common neighbors, Jaccard's coefficient, preferential attachment, adamic-adar and Katz for 5 different β values $(0.1, 0.05, 0.01, 0.005, 0.001)$. Note that, Katz, adamic-adar and Jaccard's coefficient are proven to be the best topological features for link prediction. Also, computing some of these link prediction features on a large network is substantially more costly than computing local graphlet frequency. For instance, for all datasets computing Katz takes several days! For supervised classification we use linear SVM (Support Vector Machine), for which the regularization coefficient C is chosen by grid-search from values $\{0.001, 0.01, 0.1, 1.0, 10.0, 100.0\}$ using a random 20% of test instances as a validation set. We evaluate the link prediction results using Area Under Curve ROC (ROC-AUC) and Precision-Recall AUC (PR-AUC)

In Table VI we show the link prediction results. For both the metrics, LGFD substantially improves the link prediction performance, typically 10% to 45% improvement in both kinds of AUC has been observed. This is not surprising because local graphlet frequency actually encodes information that traditional topological features encode. These results show high discriminative power of graphlet frequencies to identify future links, but not all local graphlet types are equally important. So we conducted an analysis study to find out frequencies for which specific local graphlets are highly impactful. For the study, we find normalized graphlet frequency of each graphlet which is treated as a predicted probability of a classifier to calculate individual AUC value for each graphlet type. In Table VII, we show list of graphlet types for which AUC is above 85% for FACEBOOK-WOSN and TOPOLOGY datasets and above 70% for DBLP dataset. This table shows which graphlet type is an important feature by itself, and it also highlights the fact that 5-size local graphlets are good features for link prediction.

V. Conclusion

We present an efficient algorithm for computing the frequencies of edge-centric local graphlets upto size 5 nodes. Our experimental results show that the proposed method is extremely efficient and highly scalable for large real-life networks from different domains. We also show the utility of local graphlet counts for predicting future links in social or collaboration networks and prove that local graphlet frequency vector is a superior edge feature as compared to widely used topological features.

References

[1] M. Rahman, M. A. Bhuiyan, and M. A. Hasan, "GRAFT: an efficient graphlet counting method for large graph analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2466–2478, Oct 2014.

[2] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, "Efficient graphlet counting for large networks," in *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, ser. ICDM '15, 2015, pp. 1–10.

[3] J. Ugander, L. Backstrom, and J. Kleinberg, "Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections," in *Proceedings of the 22Nd International Conference on World Wide Web*, 2013, pp. 1307–1318.

[4] W. Hayes, K. Sun, and N. Pržulj, "Graphlet-based measures are suitable for biological network comparison," *Bioinformatics*, vol. 29, no. 4, p. 483, 2013.

[5] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, p. e177, 2007.

[6] L. Zhang, R. Hong, Y. Gao, R. Ji, Q. Dai, and X. Li, "Image categorization by learning a propagated graphlet path," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 674–685, March 2016.

[7] H. Kashima, H. Saigo, M. Hattori, and K. Tsuda, "Graph kernels for chemoinformatics," *Chemoinformatics and adv. machine learning perspectives: complex computational methods and collaborative techniques*, p. 1, 2010.

[8] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph pattern mining," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2015.

[9] M. Jha, C. Seshadhri, and A. Pinar, "Path sampling: A fast and provable method for estimating 4-vertex subgraph counts," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15, 2015, pp. 495–505.

[10] M. Bhuiyan, M. Rahman, M. Rahman, and M. A. Hasan, "GUISE: uniform sampling of graphlets for large graph analysis," in *12th IEEE International Conference on Data Mining, Brussels, Belgium*, 2012, pp. 91–100.

[11] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Estimation of graphlet statistics," *arXiv preprint arXiv:1701.01772*, 2017.

[12] T. K. Saha and M. Al Hasan, "Finding network motifs using mcmc sampling." in *CompleNet*, 2015, pp. 13–24.

[13] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," *CoRR*, 2011. [Online]. Available: http://arxiv.org/abs/1101.3291

[14] M. A. Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social Network Data Analytics*, 2011, pp. 243–275.

[15] C. Aggarwal, G. He, and P. Zhao, "Edge classification in networks," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, 2016, pp. 1038–1049.

[16] V. Vacic, L. M. Iakoucheva, S. Lonardi, and P. Radivojac, "Graphlet kernels for prediction of functional residues in protein structures," *Journal of Computational Biology*, vol. 17, no. 1, pp. 55–72, 2010.

[17] A. R. Nabhan and K. Shaalan, "Keyword identification using text graphlet patterns," in *International Conf. on Applications of Natural Language to Information Systems*. Springer, 2016, pp. 152–161.

[18] L. Hermansson, T. Kerola, F. Johansson, V. Jethava, and D. Dubhashi, "Entity disambiguation in anonymized graphs using graph kernels," in *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, ser. CIKM '13, 2013, pp. 1037–1046.

[19] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *Proceedings of the ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, 2015, pp. 685–694.

[20] N. K. Ahmed, N. Duffield, T. L. Willke, and R. A. Rossi, "On sampling from massive graph streams," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1430–1441, 2017.

[21] T. Hocevar and J. Demsar, "A combinatorial approach to graphlet counting," *Bioinformatics*, vol. 30, no. 4, p. 559, 2014.

[22] N. K. Ahmed, T. Willke, and R. A. Rossi, "Estimation of local subgraph counts," in *Proceedings of the IEEE International Conference on Big Data (IEEE Big Data)*, 2016, pp. 1–10.

[23] A. Pinar, C. Seshadhri, and V. Vaidyanathan, "Escape: Efficiently counting all 5-vertex subgraphs," in *Proceedings of the 24th International Conference on World Wide Web*, 2017.