Knowl Inf Syst (2017) 50:689–722
DOI 10.1007/s10115-016-0965-5

REGULAR PAPER

Graphlet decomposition: framework, algorithms, and applications

Nesreen K. Ahmed[1] · **Jennifer Neville**[2] · **Ryan A. Rossi**[3] · **Nick G. Duffield**[4] · **Theodore L. Willke**[1]

Received: 14 November 2015 / Revised: 5 March 2016 / Accepted: 4 June 2016 /
Published online: 27 June 2016

Abstract From social science to biology, numerous applications often rely on graphlets for intuitive and meaningful characterization of networks. While graphlets have witnessed a tremendous success and impact in a variety of domains, there has yet to be a fast and efficient framework for computing the frequencies of these subgraph patterns. However, existing methods are not scalable to large networks with billions of nodes and edges. In this paper, we propose a fast, efficient, and parallel framework as well as a family of algorithms for counting k-node graphlets. The proposed framework leverages a number of theoretical combinatorial arguments that allow us to obtain significant improvement on the scalability of graphlet counting. For each edge, we count a few graphlets and obtain the exact counts of others in constant time using the combinatorial arguments. On a large collection of 300+ networks from a variety of domains, our graphlet counting strategies are on average 460× faster than existing methods. This brings new opportunities to investigate the use of graphlets on much larger networks and newer applications as we show in the experiments. To the best of our knowledge, this paper provides the largest graphlet computations to date.

Keywords Graphlet · Motif · Graph mining · Graph kernel · Classification · Graph features · Higher-order graph statistics · Biological networks · Visual graph analytics

✉ Nesreen K. Ahmed
nesreen.k.ahmed@intel.com

1 Parallel Computing Lab, Intel Corporation, Santa Clara, CA 95054, USA

2 Department of Computer Science, Purdue University, West Lafayette, IN 47906, USA

3 Palo Alto Research Center (PARC), Palo Alto, CA 94304, USA

4 Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA

1 Introduction

Recursive decomposition of networks is a widely used approach in network analysis to factorize the complex structure of real-world networks into small subgraph patterns of size k nodes. These patterns are called *graphlets* [33]. Graphlets (also known as motifs [31]) are defined as subgraph patterns recurring in real-world networks at frequencies that are statistically significant from those in random networks. Given a network, we can count the number of embedding of each graphlet in the network, creating a profile of sufficient statistics that characterizes the network structure [40]. While knowing the graphlet frequencies does not uniquely define the network structure, it has been shown that graphlet frequencies often carry significant information about the local network structure in a variety of domains [10,12,21]. This is in contrast to global topological properties (e.g., diameter, degree distribution), where networks with similar/exact global topological properties can exhibit significantly different local structures.

1.1 Graphlets, scalability, and applications

From social science to biology, graphlets have found numerous applications and were used as the building blocks of network analysis [31]. In social science, graphlet analysis (typically known as k-subgraph census) is widely adopted in sociometric studies [12,21]. Much of the work in this vein focused on analyzing triadic tendencies as important structural features of social networks (e.g., transitivity or triadic closure) as well as analyzing triadic configurations as the basis for various social network theories (e.g., social balance, strength of weak ties, stability of ties, or trust [16]). In biology [29,33], graphlets were widely used for protein function prediction [40], network alignment [30], and phylogeny [25] to name a few. More recently, there has been an increased interest in exploring the role of graphlet analysis in computer networking [7,11,18] (e.g., for web spam detection, analysis of peer-to-peer protocols and Internet AS graphs), chemoinformatics [22,34], image segmentation [49], among others [48].

While graphlet counting and discovery have witnessed a tremendous success and impact in a variety of domains from social science to biology, there has yet to be a fast and efficient approach for computing the frequencies of these patterns. For instance, Shervashidze et al. [40] take hours to count graphlets on relatively small biological networks (i.e., few hundreds/thousands of nodes/edges) and use such counts as features for graph classification [45]. Previous work showed that graphlet counting is computationally intensive since the number of possible k-subgraphs in a graph G increases exponentially with k in $\mathcal{O}(|V|^k)$ and can be computed in $\mathcal{O}(|V|.\Delta^{k-1})$ for any bounded degree graph, where Δ is the maximum degree of the graph [40].

To address these problems, we propose a fast, efficient, and parallel framework and a family of algorithms for counting graphlets of size $k = \{3, 4\}$-nodes that take only a fraction of the time to compute when compared to the current methods used. The proposed graphlet counting algorithm leverages a number of theoretical combinatorial arguments for different graphlets. For each edge, we count a few graphlets, and with these counts along with the combinatorial arguments, we obtain the exact counts of others in constant time. On a large collection of 300+ networks from a variety of domains, our graphlet counting strategies are on average 460× faster than the current methods. This brings new opportunities to investigate the use of graphlets on much larger networks and newer applications as we show in our experiments.

To the best of our knowledge, this paper provides the largest graphlet computations to date as well as the largest systematic investigation on over 300+ networks.

Furthermore, a number of important machine learning tasks are likely to benefit from such an approach, including graph anomaly detection [32,36], as well as using graphlets as features for improving community detection [39], role discovery [38], graph classification [45], and relational learning [13,37].

We test the scalability of our proposed approach experimentally on 300+ networks from a variety of domains, such as biological, social, and technological domains. We compare our approach to the state-of-the-art exact counting methods such as RAGE [27], FANMOD [47], and Orca [20]. We found that RAGE [27] took 2400 s to count graphlets on a small 26k node graph, whereas our proposed method is 460× faster on average, taking only 0.01 s. We also note that FANMOD [47], another recent approach, takes 172,800 s and Orca [20] takes 2.5 s for the same small graph. Our exact graphlet analysis is well suited for shared-memory multi-core architectures, distributed architectures (MPI), and hybrid implementations that leverage the advantages of both.

1.2 Contributions

- *Algorithms and theoretical analysis*: A fast, efficient, and parallel graphlet counting framework and a family of algorithms for graphlet counting. In addition, we provide a theoretical analysis of a number of combinatorial arguments that enable our proposed framework to obtain significant improvement on the scalability of graphlet counting.
- *Scalability*: The proposed graphlet counting algorithm is on average 460× faster than the state-of-the-art methods. In addition, we analyze graphlet counts on graphs of sizes that are beyond the scope of the state of the art (e.g., on graphs with billions of nodes and edges).
- *Effectiveness*: Largest graphlet computations to date and largest systematic evaluation on over 300+ large-scale networks from a variety of domains.
- *Applications*: Systematic investigation across a variety of existing and new applications for graphlet counting, such as finding unique patterns in graphs, graph similarity, and graph classification.

2 Background

Graphlets are subgraph patterns recurring in real-world networks at frequencies that are significantly higher than those in random networks [31,33]. Previous work showed that graphlets can be used to define universal classes of networks [31]. Moreover, graphlets are at the heart and foundation of many network analysis tasks (e.g., network classification, network alignment) [19,29,33]. In this paper, we introduce an efficient algorithm to compute the number of embedding of each graphlet of size $k = \{2, 3, 4\}$ nodes in the network (see Table 1 for notation).

2.1 Notation and definitions

Given an undirected simple input graph $G = (V, E)$, a graphlet of size k nodes is defined as any subgraph $G_k \subset G$ which consists of a subset of k nodes of the graph G. In this paper,

Table 1 Summary of graphlet notation

	Graphlet	Description	Complement	ρ	Δ	$\bar{d}$	r	$\lvert T\rvert$	K	χ	D	B	$\lvert C\rvert$
$(k=4)-$Graphlets													
Connected	g_{4_1}	4-Clique		1.00	3	3.0	1.00	4	3	4	1	0	1
	g_{4_2}	4-Chordalcycle		0.83	3	2.5	–0.66	2	2	3	2	1	1
	g_{4_3}	4-Tailedtriangle		0.67	3	2.0	–0.71	1	2	3	2	2	1
	g_{4_4}	4-Cycle		0.67	2	2.0	1.00	0	2	2	2	1	1
	g_{4_5}	3-Star		0.50	3	1.5	–1.00	0	1	2	2	3	1
	g_{4_6}	4-Path		0.50	2	1.5	–0.50	0	1	2	3	2	1
Disconnected	g_{4_7}	4-Node-1-triangle		0.50	2	1.5	1.00	1	2	3	1	0	2
	g_{4_8}	4-Node-2-star		0.33	2	1.0	–1.00	0	1	2	2	1	2
	g_{4_9}	4-Node-2-edge		0.33	1	1.0	1.00	0	1	2	1	0	2
	$g_{4_{10}}$	4-Node-1-edge		0.17	1	0.5	1.00	0	1	2	1	0	3
	$g_{4_{11}}$	4-Node-independent		0.00	0	0.0	0.00	0	0	1	∞	0	4
$(k=3)-$Graphlets													
	g_{3_1}	Triangle		1.00	2	2.0	1.00	1	2	3	1	0	1
	g_{3_2}	2-Star		0.67	2	1.33	–1.00	0	1	2	2	1	1
	g_{3_3}	3-Node-1-edge		0.33	1	0.67	1.00	0	1	2	1	0	2
	g_{3_4}	3-Node-independent		0.00	0	0.00	0.00	0	0	1	∞	0	3
$(k=2)-$Graphlets													
	g_{2_1}	Edge		1.00	1	1.0	1.00	0	1	2	1	0	1
	g_{2_2}	2-Node-independent		0.00	0	0.0	0.00	0	0	1	∞	0	2

Summary of the notation and properties for the graphlets of size $k = \{2, 3, 4\}$. Note that ρ denotes density, Δ and $\bar{d}$ denote the max and mean degree, whereas assortativity is denoted by r. Also, $|T|$ denotes the total number of triangles, K is the max k-core number, χ denotes the Chromatic number, whereas D denotes the diameter, B denotes the max betweenness, and $|C|$ denotes the number of components. Note that if $|C| > 1$, then r, D, and B are from the largest component.

we mainly focus on computing the frequencies of induced graphlets. An *induced* graphlet is an induced subgraph that consists of *all* edges between its nodes that are present in the input graph (as described in Definition 1). In addition, we distinguish between *connected* and *disconnected* graphlets (see Table 1). A graphlet is connected if there is a path from any node to any other node in the graphlet (see Definition 2). Table 1 provides a summary of the notation and properties of all possible induced graphlets of size $k = \{2, 3, 4\}$.

Definition 1 Induced Graphlet: an induced graphlet $G_k = (V_k, E_k)$ is a subgraph that consists of a subset of k vertices of the graph $G = (V, E)$ (i.e., $V_k \subset V$) together with all the edges whose endpoints both are in this subset (i.e., $E_k = \{\forall e \in E \mid e = (u, v) \wedge u, v \in V_k\}$).

Definition 2 Connected Graphlet: a graphlet $G_k = (V_k, E_k)$ is connected when there is a path from any node to any other node in the graphlet (i.e., $\forall u, v \in V_k, \exists P_{u-v} : u, \ldots, w, \ldots, v$, such that $d(u, v) \geq 0 \wedge d(u, v) \neq \infty$, where $d(u, v)$ is the distance between u and v). Assume $|C|$ denotes the number of connected components in a graphlet G_k. By definition, there exists one and only one connected component in a graphlet G_k (i.e., $|C| = 1$) if and only if G_k is connected.

PROBLEM DEFINITION. Given a family of graphlets of size k nodes $\mathcal{G}_k = \{g_{k_1}, g_{k_2}, \ldots, g_{k_m}\}$, our goal is to count the number of embeddings (appearances) of each graphlet $g_{k_i} \in \mathcal{G}_k$ in the input graph G. In other words, we need to count the number of induced graphlets G_k in G that are isomorphic to each graphlet $g_{k_i} \in \mathcal{G}_k$ in the family; such a number is denoted by $\binom{G}{g_{k_i}}$ [17].

A graphlet $g_{k_i} \in \mathcal{G}_k$ is embedded in the graph G, if and only if there is an injective mapping $\sigma : V_{g_{k_i}} \to V$, with $e = (u, v) \in E_{g_{k_i}}$ if and only if $e' = (\sigma(u), \sigma(v)) \in E$. Table 1 shows that $|\mathcal{G}_k| = \{2, 4, 11\}$ when $k = \{2, 3, 4\}$, respectively. Further, given a family $\mathcal{G}_k = \{g_{k_1}, g_{k_2}, \ldots, g_{k_m}\}$ of graphlets of size k nodes, we define $f(g_{k_i}, G)$ as the frequency of any graphlet $g_{k_i} \in \mathcal{G}_k$ in the input graph G.

2.2 Relationship to graph complement

The complement of a graph G, denoted by $\bar{G}$, is the graph defined on the same vertices as G such that two vertices are connected in $\bar{G}$ if and only if they are not connected in G. Therefore, the graph sum $G + \bar{G}$ gives the complete graph on the set of vertices of G. There are direct relationships between the frequencies of graphlets and the frequencies of their complement. For each graphlet g_{k_i}, there exists a non-isomorphic complementary graphlet pattern $\bar{g}_{k_i}$ such that two vertices are connected in $\bar{g}_{k_i}$ if and only if they are not connected in g_{k_i} [17]. For example, cliques and independent sets of size k nodes are pairs of complementary graphlets. Similarly, chordal cycles of size 4 nodes are complementary to the 4-node-1-edge graphlet (see Table 1). It is also worth noting that the 4-path graphlet is a self-complementary pattern, which means the 4-path is isomorphic to itself. From this discussion, it is clear that the number of embeddings of each graphlet $g_{k_i} \in \mathcal{G}_k$ in the input graph G is equivalent to the number of embeddings of its complementary graphlet $\bar{g}_{k_i}$ in the complement graph $\bar{G}$. In other words, $f(g_{k_i}, G) = f(\bar{g}_{k_i}, \bar{G})$ [17].

2.3 Relationship to graph/matrix reconstruction theorems

The graph reconstruction conjecture [17] states that an undirected graph G can be uniquely determined up to an isomorphism, from the set of all possible vertex-deleted subgraphs of G (i.e., $\{G_v\}_{v \in V}$) [28]. Verification of this conjecture for all possible graphs up to 6 vertices was carried by Kelly [23] and later was extended to up to 11 vertices by McKay [28]. Clearly, if two graphs are isomorphic (i.e., $G \cong G'$), then their graphlet frequencies would be the same (i.e., $f_k(G) = f_k(G')$), but the reverse remains a converse for the general case of graphs. In contrast, the matrix reconstruction theorem has been resolved [26], which states that any $N \times N$ matrix can be reconstructed from its list of all possible principal minors obtained by the deletion of the kth row and the kth column [26], which is the foundation of a class of graph kernels called the *graphlet kernel* [40].

2.4 Related work

In this section, we briefly discuss some of the related work, highlighting various graph mining and machine learning tasks that would benefit from our approach. Much of the previous work

focused on counting certain types of graphlets (e.g., only connected graphlets such as cliques and cycles) [20,24,47]. However, a number of graph mining and machine learning tasks rely on counting all graphlets of a certain size.

For example, some previous work used the full spectrum of graphlet frequencies to define a domain-independent coordinate system in which collections of graphs can be compactly represented and analyzed within a common space [44]. Moreover, a variety of graph kernels have been proposed in machine learning (e.g., graphlet, subtree, and random walk kernels) [9,40,45] to bridge the gap between graph learning and kernel methods. And some types of the graph kernels, in particular the graphlet kernel, rely on counting all graphlets. However, a general limitation of most graph kernels (including the graphlet kernel) is that they scale poorly to graphs with more than few hundreds/thousands of nodes [45]. Thus, our fast algorithms would speed up the computations of these methods and their related applications in graph modeling, similarity, and comparisons.

Recently, there is an increased interest in sampling and other heuristic approaches for obtaining approximate counts of various graphlets [3–5,8,15]. However, our approach focuses on exact graphlet counting and thus sampling methods are outside the scope of this paper. Nevertheless, the analysis and combinatorial arguments we show in this paper can be used along with efficient sampling methods to provide more accurate and efficient approximations [2].

In addition, the aim and scope of this paper is different from the aforementioned problem of graph reconstruction. While graph reconstruction tries to test for the notion of isomorphism and structure equivalence between graphs, our goal is to relax the notion of equivalence to some form of *structural similarity* between graphs such that the graph similarity is measured using the feature representation of graphlets.

3 Framework

In this section, we describe our approach for graphlet counting that takes only a fraction of the time to compute when compared to the current methods used. We introduce a number of combinatorial arguments that we show for different graphlets. The proposed graphlet counting algorithm leverages these combinatorial arguments to obtain significant improvement on the scalability of graphlet counting. For each edge, we count only a few graphlets, and with these counts along with the combinatorial arguments, we derive the exact counts of the others in constant time.

3.1 Searching edge neighborhoods

Our proposed algorithm iterates over all the edges of the input graph $G = (V, E)$. For each edge $e = (u, v) \in E$, we define the *neighborhood* of an edge e, denoted by $\mathcal{N}(e)$, as the set of all nodes that are connected to the endpoints of e—i.e., $\mathcal{N}(e) = \{\mathcal{N}(u)\backslash\{v\}\} \cup \{\mathcal{N}(v)\backslash\{u\}\}$, where $\mathcal{N}(u)$ and $\mathcal{N}(v)$ are the set of neighbors of u and v, respectively. Given a single edge $e = (u, v) \in E$, we explore the subgraph surrounding this edge—i.e., the subgraph induced by both its endpoints and the nodes in its neighborhood. We call this subgraph the *egonet* of the edge e, where e is the center (ego) of the subgraph.

We search for possible graphlet patterns of size $k = \{3, 4\}$ in the egonets of all edges in the graph. By searching egonets of edges, we first map the problem to the local (lower-dimensional) space induced by the neighborhood of each edge and then merge the search results for all edges. Searching over a local low-dimensional space of edge neighborhoods

Algorithm 1 Our exact triad census algorithm for counting all 3-node graphlets. The algorithm takes an undirected graph as input and returns the frequencies of all 3-node graphlets $f(\mathcal{G}_3, G)$.

1: **procedure** TRIADCENSUS($G = (V, E)$)
2: Initialize all variables
3: **parallel for** $e = (u, v) \in E$ **do**
4: $\text{Star}_u = \emptyset$, $\text{Star}_v = \emptyset$, $\text{Tri}_e = \emptyset$
5: **for** $w \in \mathcal{N}(u)$ **do**
6: **if** $w = v$ **then continue**
7: Add w to Star_u and set $X(w) = 1$
8: **for** $w \in \mathcal{N}(v)$ **do**
9: **if** $w = u$ **then continue**
10: **if** $X(w) = 1$ **then** ▷ found triangle
11: Add w to Tri_e
12: Remove w from Star_u
13: **else** Add w to Star_v
14: $f(g_{3_1}, G)$ += $|\text{Tri}_e|$
15: $f(g_{3_2}, G)$ += $|\text{Star}_u| + |\text{Star}_v|$
16: $f(g_{3_3}, G)$ += $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$
17: **for** $w \in \mathcal{N}(u)$ **do** $X(w) = 0$
18: **end parallel**
19: Aggregate counts from all workers
20: $f(g_{3_1}, G) = 1/3.f(g_{3_1}, G)$
21: $f(g_{3_2}, G) = 1/2.f(g_{3_2}, G)$
22: $f(g_{3_4}, G) = \binom{|V|}{3} - f(g_{3_1}, G) - f(g_{3_2}, G) - f(g_{3_3}, G)$
23: **return** $f(\mathcal{G}_3, G)$

is clearly more efficient than searching over the global high-dimensional space of the whole graph. Moreover, searching over a local low-dimensional space of edge neighborhoods is amenable to parallel implementation, which offers additional speedup over iterative methods. Note that exhaustive search of the egonet of any edge $e \in E$ yields at least $\mathcal{O}(\Delta^{k-1})$ asymptotically, where Δ is the maximum degree in G. Clearly, exhaustive search is computationally intensive for large graphs, and our approach is more efficient as we will show next.

3.2 Counting graphlets of size ($k = 3$) nodes

Algorithm 1 (TRIADCENSUS) shows how to count graphlets of size $k = 3$ for each edge. There are four possible graphlets of size $k = 3$ nodes, where only g_{3_1} (i.e., triangle patterns) and g_{3_2} (i.e., 2-star patterns) are connected graphlets (see Table 1).

Connected graphlets of size $k = 3$

Lines 5—13 of Algorithm 1 show how to find and count triangles incident to an edge. For any edge $e = (u, v)$, a triangle (u, v, w) exists, if and only if w is connected to *both* u and v. Let Tri_e be the set of all nodes that form a triangle with $e = (u, v)$, and let $|\text{Tri}_e|$ be the number of such triangles. Then, Tri_e is the set of overlapping nodes in the neighborhoods of u and v—$\text{Tri}_e = \mathcal{N}(u) \cap \mathcal{N}(v)$. Note that Algorithm 1 counts each triangle three times (one time for each edge in the triangle), and therefore we divide the total count by 3 as in Equation (1),

$$f(g_{3_1}, G) = \frac{1}{3}. \sum_{e=(u,v)\in E} |\text{Tri}_e| \tag{1}$$

Now we need to count 2-star patterns (i.e., g_{3_2}). For any edge $e = (u, v)$, let Star_e be the set of all nodes that form a 2-star with e and $|\text{Star}_e|$ be the number of such star patterns. A 2-star pattern (u, v, w) exists, if and only if w is connected to *either* u or v but not both. Accordingly, $\text{Star}_e = \text{Star}_u \cup \text{Star}_v$, where Star_u and Star_v are the set of nodes that form a 2-star with e centered at u and v, respectively. More formally, Star_u can be defined as $\text{Star}_u = \{w \in \mathcal{N}(u)\backslash\{v\} | w \notin \mathcal{N}(v)\}$ and Star_v can be defined as $\text{Star}_v = \{w \in \mathcal{N}(v)\backslash\{u\} | w \notin \mathcal{N}(u)\}$.

Similar to counting triangles, Algorithm 1 counts each 2-star pattern two times (one time for each edge in the 2-star). Thus, we divide the sum for all edges by 2 as follows,

$$f(g_{3_2}, G) = \frac{1}{2}. \sum_{e=(u,v)\in E} |\text{Star}_u| + |\text{Star}_v| \tag{2}$$

Disconnected graphlets of size $k = 3$

There are two disconnected graphlets of size $k = 3$ nodes, g_{3_3} (i.e., the 3-node-1-edge pattern) and g_{3_4} (i.e., the independent set defined on 3 nodes) (see Table 1). Lines 16 and 22 show how to count these patterns.

Equation (3) shows that the number of 3-node-1-edge graphlets per edge e is equivalent to the number of all nodes that are not in the neighborhood subgraph (egonet) of edge e (i.e., $V\backslash\{\mathcal{N}(u) \cup \mathcal{N}(v)\}$),

$$f(g_{3_3}, G) = \sum_{e=(u,v)\in E} |V| - |\mathcal{N}(u) \cup \mathcal{N}(v)| \tag{3}$$

where $|\mathcal{N}(u) \cup \mathcal{N}(v)| = |\text{Tri}_e| + |\text{Star}_e| + |\{u, v\}|$. Note that the number of 3-node-1-edge graphlets can be computed in $o(1)$ for each edge.

Given that the total number of graphlets of size 3 nodes is $\binom{N}{3}$, Equation (4) shows how to compute the frequency of g_{3_4}, which clearly can be done in $o(1)$,

$$f(g_{3_4}, G) = \binom{|V|}{3} - \big(f(g_{3_1}, G) + f(g_{3_2}, G) + f(g_{3_3}, G)\big) \tag{4}$$

The complexity of counting all graphlets of size $k = 3$ is $\mathcal{O}(|E|.\Delta)$ asymptotically as we show next in Lemma 1.

Lemma 1 *Algorithm 1 counts all graphlets of size $k = 3$–nodes in $\mathcal{O}(|E|.\Delta)$.*

Proof For each edge $e = (u, v)$ such that $e \in E$, the runtime complexity of counting all triangle and 2-star patterns incident to e (i.e., Tri_e, Star_e, respectively) is $O(|\mathcal{N}(u)|+|\mathcal{N}(v)|)$ and is asymptotically $O(\Delta)$, where Δ is the maximum degree in the graph. Further, the runtime complexity of counting all 3-node-1-edge patterns of size $k = 3$ incident to e can be counted in constant time $o(1)$. Therefore, the total runtime complexity for counting all graphlets of size $k = 3$ in the graph is $\mathcal{O}\Big(\sum_{e\in E} (\Delta + o(1))\Big) = \mathcal{O}(|E|.\Delta)$. □

Parallelization

Our own implementation of Algorithm 1 uses shared memory; however, we describe the parallelization at a high level such that it could be used with distributed memory architectures

as well. In fact, in these cases the algorithm itself remains the same. We particularly focus our discussion on the general scheme and not on the specific details. We parallelized the main parallel-for loop in the algorithm (Line 3), although other parts could be parallelized as well.

The algorithm starts by initializing all variables to zero (e.g., X, $f(., G)$). We view the main parallel-for loop as a task generator and form the current edge (the next job) out to a worker to count the graphlets incident to this edge. One of the key features of our algorithm is that it is *lock-free*; unlike previous methods, this is due to our theoretical analysis that we use to minimize the graphlet counting computations.

The lock-free nature of our algorithm allows us to minimize the communication cost between workers. Each worker maintains local variables and total counts of each graphlet pattern processed. Upon completion of the parallel-for loop, we use a reduction step to aggregate the counts from all workers. Since there is a very minimal sharing of memory across workers, we can exploit memory locality and avoid cache ping-pong. Note that unlike previous methods, our algorithm achieves near-linear scaling for the multi-core setting (see Sect. 5.2).

Moreover, our algorithm is also memory efficient compared to previous methods, since there is no need to store extra information per vertex/edge, as we aggregate the counts for each worker, and then after the main parallel-for loop, we aggregate the counts from all workers. Our algorithm also uses *dynamic scheduling* to dynamically assign jobs to each worker when more work is requested (i.e., batch size b). This scheme allows us to dynamically load the balance among the workers (see Sect. 5.2 for the effect of the batch size). Finally, another key feature of our algorithm is that it accepts different node/edge ordering (such as degree, k-core) in order to improve memory locality and caching.

4 Counting graphlets of size ($k = 4$) nodes

An exhaustive search of the egonet of any edge to count all 4-node graphlets independently yields $\mathcal{O}(\Delta^3)$ asymptotically, where Δ is the maximum degree in G. Clearly, exhaustive search is computationally intensive for large graphs. On the other hand, our approach is hierarchical and more efficient as we show next.

For each edge $e = (u, v)$, we start by finding triangles and 2-star patterns. Our central principle is that any 4-node graphlet g_{4_i} can be decomposed into four 3-node graphlets [17], obtained by deleting one node from g_{4_i} each time. Thus, we jointly count all possible 4-node graphlets by leveraging the knowledge obtained from finding 3-node graphlets and some combinatorial arguments that describe the relationships between pairs of graphlets. We summarize this procedure in the following steps:

- STEP 1: For each edge e, find all neighborhood nodes forming triangle and 2-star patterns with e.
- STEP 2: For each edge e, use the knowledge from step 1 to count only 4-cliques and 4-cycles.
- STEP 3: For each edge e, use the knowledge from step 1 and some combinatorial arguments to compute unrestricted counts for all 4-node graphlets in constant time.
- STEP 4: Merge the counts from all edges in the graph, and use combinatorial arguments involving unrestricted counts to obtain the counts of all other graphlets.

Note that we refer to the unrestricted counts as the counts that can be computed in constant time and using only the knowledge obtained from step 1. Next, we discuss the details of our approach. We start by discussing the graphlet transition diagram to show the pairwise relationships between different 4-node graphlets. Then, we discuss a general principle

for counting 4-node graphlets, which leverages the graphlet transition diagram and some combinatorial arguments to improve the performance of graphlet counting.

4.1 Graphlet transition diagram

Assume that each graphlet is a state; Fig. 1 shows all possible ± 1 *edge* transitions between the states of all 4-node graphlets. We can transition from one graphlet to another by the deletion (denoted by dashed right arrows) or addition (denoted by solid left arrows) of a single edge. We define six different classes of possible edge roles denoted by the colors from black to orange (see Table in the top-right corner in Fig. 1). An *edge role* is an edge-level connectivity pattern (e.g., a chord edge), where two edges belong to the same role (i.e., class) if they are similar in their topological features.

For each edge, we define a topological feature vector that consists of the number of triangles and 2-stars incident to this edge. Then, we classify edges to one of the six roles based on their feature vectors. Thus, all edges that appear in 4-node graphlets are colored by their roles. In addition, the transition arrows are colored similar to the edge roles to denote which edge type should be deleted/added to transition from one graphlet to another. Note that a single edge deletion/addition changes the role (class) of other edges in the graphlet. The

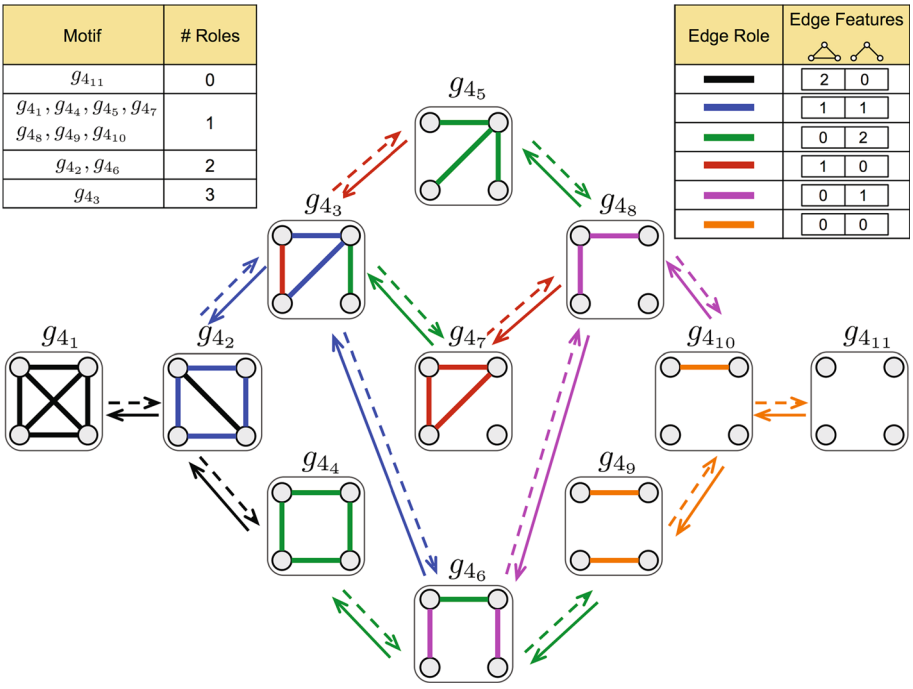

Fig. 1 *(4-node) graphlet transition diagram*: Figure shows all possible ± 1 *edge* transitions between the set of all 4-node graphlets. *Dashed right arrows* denote the deletion of one edge to transition from one graphlet to another. *Solid left arrows* denote the addition of one edge to transition from one graphlet to another. Edges are colored by their feature-based roles, where the set of features are defined by the number of triangles and 2-stars incident to an edge (see *Table* in the top-right corner). We define six different classes of edge roles colored from *black* to *orange* (see *Table* in the top-right corner). *Dashed/solid arrows* are colored similar to the edge roles to denote which edge would be deleted/added to transition from one graphlet to another. The table in the top-left corner shows the number of edge roles per each graphlet (color figure online)

table in the top-left corner of Fig. 1 shows the number of edge roles per each graphlet. For example, consider the 4-clique graphlet (g_{4_1}), where each edge participates exactly in two triangles. Therefore, all the edges in a 4-clique graphlet (g_{4_1}) belong to the first role (denoted by the black color). Similarly, consider the 4-chordalcycle (g_{4_2}), where each edge (except the chord edge) participates exactly in one triangle and one 2-star. Therefore, all edges in a 4-chordalcycle "g_{4_2}" belong to the second role (denoted by the blue color) except for the chord edge which belongs to the first role (denoted by the black color). Fig. 1 shows how to transition from the 4-clique to the 4-chordalcycle "g_{4_2}" by deleting one (any) edge from the 4-clique.

4.2 General principle for counting graphlets of size $k = 4$

Generally speaking, suppose we have $N^{(e)}$ distinct 4-node subgraphs that contain an edge $e = (u, v)$,

$$N^{(e)} = \left|\left\{\{u, v, w, r\} \mid w, r \in V \backslash \{u, v\} \wedge w \neq r\right\}\right| \tag{5}$$

Each subgraph $\{u, v, w, r\}$ in this collection may satisfy one or two properties $a_i, a_j \in A = \{T, S_u, S_v, I\}$. As we show by example in Fig. 2, let T denote the nodes forming triangles with edge (u, v) (i.e., V_2, V_3), whereas S_u and S_v denote the nodes forming 2-stars centered at u and v, respectively (i.e., V_1, V_4), and let I denote the nodes that are not connected to edge e (i.e., V_5, V_6). These properties describe the topological properties of nodes w and r with respect to edge e, such that $A_w = a_i$ if $\{u, v, w\}$ forms subgraph pattern a_i, and $A_r = a_j$ if $\{u, v, r\}$ forms subgraph pattern a_j. For example, $A_w = T$ if w forms a triangle with e, and $A_w = S_u$ or S_v if w forms a 2-star with e centered around u or v, respectively. Also, $A_w = I$ if w is independent (disconnected) from e. We clarify these properties by example in Fig. 2.

Let $N^{(e)}_{a_i,a_j}$ denote the number of distinct 4-node graphlets that contain edge e = (u, v) and have properties $a_i, a_j \in A$,

$$N^{(e)}_{a_i,a_j} = \left|\left\{\{u, v, w, r\} \;\middle|\; \begin{smallmatrix} w,r \in V\backslash\{u,v\} \\ \wedge w \neq r \\ \wedge A_w = a_i, A_r = a_j \end{smallmatrix}\right\}\right| \tag{6}$$

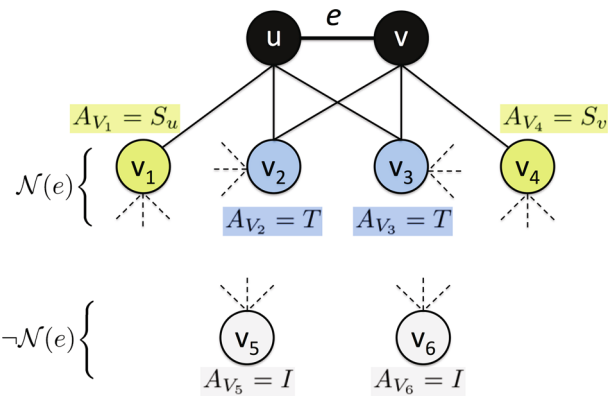

Fig. 2 Let T denote the nodes forming *triangles* with edge (u, v) (i.e., V_2, V_3), whereas S_u and S_v denote the nodes forming 2-stars centered at u and v, respectively (i.e., V_1, V_4), and let I denote the nodes that are not connected to edge e (i.e., V_5, V_6). Further, the *dotted lines* represent edges incident to these nodes

Now that we defined the topological properties of nodes w and r relative to edge e, we need to define whether nodes w and r are connected with themselves. Let e'_{wr} represent whether w and r are connected or not, such that $e'_{wr} = 1$ if $(w, r) \in E$ and $e'_{wr} = 0$ otherwise. Accordingly, let $N^{(e)}_{a_i,a_j,e'_{wr}}$ denote the number of 4-node graphlets $\{u, v, w, r\}$, where w, r satisfy property $a_i, a_j \in A$ and $e'_{wr} \in \{0, 1\}$,

$$N^{(e)}_{a_i,a_j,e'_{wr}} = \left| \left\{ \{u, v, w, r\} \;\middle|\; \begin{matrix} w,r \in V \backslash \{u,v\} \\ \wedge w \neq r \\ \wedge A_w = a_i, A_r = a_j \\ \wedge e'_{wr} \in \{0,1\} \end{matrix} \right\} \right| \tag{7}$$

For example, $N^{(e)}_{T,T,1}$ is the number of all graphlets $\{u, v, w, r\}$ containing edge e, where both w and r are forming triangles with e and there exists an edge between w and r. Using Equations (6) and (7), we provide a general principle for graphlet counting in the following theorem.

Theorem 1 *General Principle for Graphlet Counting: Given a graph G, for any edge $e = (u, v)$ in G, and for any properties $a_i, a_j \in A$, the number of 4-node graphlets $\{u, v, w, r\}$ satisfies the following rule,*

$$N^{(e)}_{a_i,a_j,0} = N^{(e)}_{a_i,a_j} - N^{(e)}_{a_i,a_j,1} \tag{8}$$

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing edge e, where nodes w and r satisfy a_i, a_j properties, respectively, and $(w, r) \in E$. Then the expression on the right side counts this subgraph once in the $N^{(e)}_{a_i,a_j}$ term, and once in the $N^{(e)}_{a_i,a_j,1}$. By the principle of inclusion–exclusion [41], the total contribution of the subgraph $\{u, v, w, r\}$ in $N^{(e)}_{a_i,a_j,0}$ is zero. Thus, $N^{(e)}_{a_i,a_j,0}$ is the number of graphlets having properties a_i, a_j, but $(w, r) \notin E$. □

Clearly, it is sufficient to compute $N^{(e)}_{a_i,a_j}$ and $N^{(e)}_{a_i,a_j,1}$ only, and use Theorem 1 to compute $N^{(e)}_{a_i,a_j,0}$ in constant time. Note that $N^{(e)}_{a_i,a_j}$ is an unrestricted count and can be computed in constant time using the knowledge we have from finding 3-node graphlets.

To simplify the discussion in the following sections, we precisely show how to compute $N^{(e)}_{a_i,a_j}$, the number of 4-node graphlets $\{u, v, w, r\}$ such that w, r satisfy property $a_i, a_j \in A$, respectively. Let $\mathcal{W}_{a_i}$ be the set of nodes with property $a_i \in A$ (i.e., $\mathcal{W}_{a_i} = \{w \in V \backslash \{u, v\} \mid A_w = a_i, \forall a_i \in A\}$), and similarly $\mathcal{R}_{a_j}$ be the set of nodes with property $a_j \in A$ (i.e., $\mathcal{R}_{a_j} = \{r \in V \backslash \{u, v\} \mid A_r = a_j, \forall a_j \in A\}$). If $a_i = a_j$, then $\mathcal{W}_{a_i} = \mathcal{R}_{a_j}$. Thus,

$$N^{(e)}_{a_i,a_i} = \binom{|\mathcal{W}_{a_i}|}{2} = \frac{1}{2}.(|\mathcal{W}_{a_i}| - 1).|\mathcal{W}_{a_i}| \tag{9}$$

However, if $a_i \neq a_j$, then $\mathcal{W}_{a_i}$ and $\mathcal{R}_{a_j}$ are mutually exclusive (i.e., $\mathcal{W}_{a_i} \cap \mathcal{R}_{a_j} = \emptyset$). Thus, we get the following,

$$N^{(e)}_{a_i,a_j} = |\mathcal{W}_{a_i}|.|\mathcal{R}_{a_j}| \tag{10}$$

4.3 Analysis and combinatorial arguments

In this section, we discuss combinatorial arguments involving unrestricted counts that can be computed directly from our knowledge of 3-node graphlets. These combinatorial arguments capture the relationships between the counts of pairs of 4-node graphlets. The proofs of these relationships are based on Theorem 1 and the transition diagram in Fig. 1. For each pair of

graphlets g_{4_i} and g_{4_j}, we show the relationship for each edge in the graph (in Corollary 1–14), and then we show a generalization for the whole graph (in Lemma 2–8).

4.3.1 Relationship between 4-cliques and 4-chordalcycles

Corollary 1 *For any edge $e = (u, v)$ in the graph, the number of 4-cliques containing e is $N^{(e)}_{T,T,1}$.*

Corollary 2 *For any edge $e = (u, v)$ in the graph, the number of 4-chordalcycles, where e is the chord edge of the cycle (denoted by the black color in Fig. 1), is $N^{(e)}_{T,T,0}$.*

Lemma 2 *For any graph G, the relationship between the counts of 4-cliques (i.e., $f(g_{4_1}, G)$) and 4-chordalcycles (i.e., $f(g_{4_2}, G)$) is,*

$$f(g_{4_2}, G) = \sum_{e \in E} \binom{|\text{Tri}_e|}{2} - 6.f(g_{4_1}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{T,T,0} = \sum_{e \in E} N^{(e)}_{T,T} - \sum_{e \in E} N^{(e)}_{T,T,1} \tag{11}$$

Given that $N^{(e)}_{T,T}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = T$, $A_r = T$. Thus, from Eq. (9), $N^{(e)}_{T,T} = \binom{|\text{Tri}_e|}{2}$. From Corollary 1, each 4-clique will be counted 6 times (once for each edge in the clique). Thus, the total count of 4-cliques in G is $f(g_{4_1}, G) = \frac{1}{6} \cdot \sum_{e \in E} N^{(e)}_{T,T,1}$. Similarly, from Corollary 2, each 4-chordalcycle is counted only once for each chord edge. Thus, the total count of 4-chordalcycles in G is $f(g_{4_2}, G) = \sum_{e \in E} N^{(e)}_{T,T,0}$. By direct substitution in Eq. (11), this lemma is true. □

4.3.2 Relationship between 4-cycles and 4-paths

Corollary 3 *For any edge $e = (u, v)$ in the graph, the number of 4-cycles containing e is $N^{(e)}_{S_u,S_v,1}$.*

Corollary 4 *For any edge $e = (u, v)$ in the graph, the number of 4-paths containing e, where e is the middle edge in the path (denoted by the green color in Fig. 1), is $N^{(e)}_{S_u,S_v,0}$.*

Lemma 3 *For any graph G, the relationship between the counts of 4-cycles (i.e., $f(g_{4_4}, G)$) and 4-paths (i.e., $f(g_{4_6}, G)$) is,*

$$f(g_{4_6}, G) = \sum_{e \in E} |\text{Star}_u|.|\text{Star}_v| - 4.f(g_{4_4}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{S_u,S_v,0} = \sum_{e \in E} N^{(e)}_{S_u,S_v} - \sum_{e \in E} N^{(e)}_{S_u,S_v,1} \tag{12}$$

Given that $N^{(e)}_{S_u,S_v}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that w, r $A_w = S_u$, $A_r = S_v$. Thus, from Eq. (10), $N^{(e)}_{S_u,S_v} = |\text{Star}_u|.|\text{Star}_v|$. From Corollary 3,

each 4-cycle will be counted 4 times (once for each edge in the cycle). Thus, the total count of 4-cycles in G is $f(g_{4_4}, G) = \frac{1}{4} \cdot \sum_{e \in E} N^{(e)}_{S_u, S_v, 1}$. Similarly, from Corollary 4, each 4-path is counted only once for each middle edge in the path. Thus, the total count of 4-paths in G is $f(g_{4_6}, G) = \sum_{e \in E} N^{(e)}_{S_u, S_v, 0}$. By direct substitution in Eq. (12), this lemma is true. □

4.3.3 Relationship between 4-tailedtriangles and 4-chordalcycles

Corollary 5 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles where e is part of both the triangle and 2-star patterns (denoted by the blue color in Fig. 1) is $N^{(e)}_{T, S_u \vee S_v, 0}$.*

Corollary 6 *For any edge $e = (u, v)$ in the graph, the number of 4-chordalcycles where e is a cycle edge (denoted by the blue color in Fig. 1) is $N^{(e)}_{T, S_u \vee S_v, 1}$.*

Lemma 4 *For any graph G, the relationship between the counts of 4-chordalcycles (i.e., $f(g_{4_2}, G)$) and 4-tailedtriangles (i.e., $f(g_{4_3}, G)$) is,*

$$2.f(g_{4_3}, G) = \sum_{e \in E} |\text{Tri}_e|.(|\text{Star}_u| + |\text{Star}_v|) - 4.f(g_{4_2}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{T, S_u \vee S_v, 0} = \sum_{e \in E} N^{(e)}_{T, S_u \vee S_v} - \sum_{e \in E} N^{(e)}_{T, S_u \vee S_v, 1} \tag{13}$$

Given that $N^{(e)}_{T, S_u \vee S_v} = N^{(e)}_{T, S_u} + N^{(e)}_{T, S_v}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = T, A_r = S_u \vee S_v$. Thus, from Eq. (10), $N^{(e)}_{T, S_u \vee S_v} = |\text{Tri}_e|.(|\text{Star}_u| + |\text{Star}_v|)$. Now, from Corollary 6, each 4-chordalcycle is counted 4 times (once for each edge in the cycle). Thus, the total count of 4-chordalcycle in G is $f(g_{4_2}, G) = \frac{1}{4} \cdot \sum_{e \in E} N^{(e)}_{T, S_u \vee S_v, 1}$. Similarly, from Corollary 5, each 4-tailedtriangle will be counted 2 times (once for each blue edge as in Fig. 1). Thus, the total count of 4-tailedtriangle in G is $f(g_{4_3}, G) = \frac{1}{2} \cdot \sum_{e \in E} N^{(e)}_{T, S_u \vee S_v, 0}$. By direct substitution in Eq. (13), this lemma is true. □

4.3.4 Relationship between 4-tailedtriangles and 3-stars

Corollary 7 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles with e as the tail edge (denoted by the green color in Fig. 1) and with u is part of the triangle is $N^{(e)}_{S_u, S_u, 1}$.*

In a similar fashion, the number of 4-tailedtriangles with e as the tail edge and v as part of the triangle is $N^{(e)}_{S_v, S_v, 1}$. Thus, the total number of 4-tailedtriangles with e as the tail edge and $u \vee v$ as part of the triangle is $N^{(e)}_{S_\cdot, S_\cdot, 1} = N^{(e)}_{S_u, S_u, 1} + N^{(e)}_{S_v, S_v, 1}$.

Corollary 8 *For any edge $e = (u, v)$ in the graph, the number of 3-star centered around u is $N^{(e)}_{S_u, S_u, 0}$.*

Again, the number of 3-stars centered around v is $N^{(e)}_{S_v, S_v, 0}$. Thus, the total number of 3-stars centered around u or v is $N^{(e)}_{S_\cdot, S_\cdot, 0} = N^{(e)}_{S_u, S_u, 0} + N^{(e)}_{S_v, S_v, 0}$.

Lemma 5 *For any graph G, the relationship between the counts of 3-stars (i.e., $f(g_{4_5}, G)$) and 4-tailedtriangles (i.e., $f(g_{4_3}, G)$) is,*

$$3.f(g_{4_5}, G) = \sum_{e \in E} \left(\binom{|\text{Star}_u|}{2} \right) + \left(\binom{|\text{Star}_v|}{2} \right) - f(g_{4_3}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{S_\cdot, S_\cdot, 0} = \sum_{e \in E} N^{(e)}_{S_\cdot, S_\cdot} - \sum_{e \in E} N^{(e)}_{S_\cdot, S_\cdot, 1} \tag{14}$$

Given that $N^{(e)}_{S_\cdot, S_\cdot} = N^{(e)}_{S_u, S_u} + N^{(e)}_{S_v, S_v}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = S_u \wedge A_r = S_u$ or $A_w = S_v \wedge A_r = S_v$. Thus, from Eq. (9), $N^{(e)}_{S_\cdot, S_\cdot} = \left(\binom{|\text{Star}_u|}{2} \right) + \left(\binom{|\text{Star}_v|}{2} \right)$. Now, from Corollary 8, each 3-star is counted 3 times (once for each edge in the star). Thus, the total count of 3-stars in G is $f(g_{4_5}, G) = \frac{1}{3} \cdot \sum_{e \in E} N^{(e)}_{S_\cdot, S_\cdot, 0}$. Similarly, from Corollary 7, each 4-tailedtriangle will be counted once for each tail edge (denoted by the green color in Fig. 1). Thus, the total count of 4-tailedtriangle in G is $f(g_{4_3}, G) = \sum_{e \in E} N^{(e)}_{S_\cdot, S_\cdot, 1}$. This holds whether the patterns are centered around u or v. By direct substitution in Eq. (14), this lemma is true. □

4.3.5 Relationship between 4-tailedtriangles and 4-node-1-triangles

Corollary 9 *For any edge $e = (u, v)$ in the graph, the number of 4-node-1-triangle is $N^{(e)}_{T,I,0}$.*

Corollary 10 *For any edge $e = (u, v)$ in the graph, the number of 4-tailedtriangles with e participating in the triangle but not connected to the tail edge (denoted by the red color in Fig. 1) is $N^{(e)}_{T,I,1}$.*

Proof Suppose there is a subgraph $\{u, v, w, r\}$ containing e. $\{u, v, w, r\}$ is a 4-tailedtriangle with e participating in the triangle but not connected to the tail edge, if and only if there are some nodes w, r such that $w \in \text{Tri}_e, r \not\in \mathcal{N}(e)$, and $(w, r) \in E$. This means r is independent of e, and w forms a triangle with e. As such, $A_w = T$ and $A_r = I$ and $e'_{wr} = 1$. More generally, any subgraph $\{u, v, w, r\}$ containing e contributes once in the count $N^{(e)}_{T,I,1}$ if and only if it is a 4-tailedtriangle with e participating in the triangle but not connected to the tail edge. In Theorem 1, we showed that $N^{(e)}_{T,I,1} \leq N^{(e)}_{T,I}$. □

Lemma 6 *For any graph G, the relationship between the counts of 4-tailedtriangles (i.e., $f(g_{4_3}, G)$) and 4-node-1-triangles (i.e., $f(g_{4_7}, G)$) is,*

$$3.f(g_{4_7}, G) = \sum_{e \in E} \left(\text{Tri}_e . (|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|) \right) - f(g_{4_3}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{T,I,0} = \sum_{e \in E} N^{(e)}_{T,I} - \sum_{e \in E} N^{(e)}_{T,I,1} \tag{15}$$

Given that $N^{(e)}_{T,I}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = T, A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$.

Thus, from Eq. (10), $N^{(e)}_{T,I} = \text{Tri}_e.\Big(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|\Big)$. Now, from Corollary 10, each 4-tailedtriangle is counted one time (once for the red edge as in Fig. 1). Thus, the total count of 4-tailedtriangles in G is $f(g_{4_3}, G) = \sum_{e \in E} N^{(e)}_{T,I,1}$. Similarly, from Corollary 9, each 4-node-1-triangle will be counted 3 times (once for each edge in the triangle). Thus, the total count of 4-node-1-triangles in G is $f(g_{4_7}, G) = \frac{1}{3}.\sum_{e \in E} N^{(e)}_{T,I,0}$. By direct substitution in Eq. (15), this lemma is true. □

4.3.6 Relationship between 4-paths and 4-node-2-stars

Corollary 11 *For any edge $e = (u, v)$ in the graph, the number of 4-paths where e is the start or end of the path (denoted by the purple color in Fig. 1) is $N^{(e)}_{S_u \vee S_v, I, 1}$.*

Corollary 12 *For any edge $e = (u, v)$ in the graph, the number of 4-node-2-stars where e is one of the star edges (denoted by the purple color in Fig. 1) is $N^{(e)}_{S_u \vee S_v, I, 0}$.*

Lemma 7 *For any graph G, the relationship between the counts of 4-paths (i.e., $f(g_{4_6}, G)$) and 4-node-2-stars (i.e., $f(g_{4_8}, G)$) is,*

$$2.f(g_{4_8}, G) = \sum_{e \in E} |\text{Star}_e|.(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|) - 2.f(g_{4_6}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{S_u \vee S_v, I, 0} = \sum_{e \in E} N^{(e)}_{S_u \vee S_v, I} - \sum_{e \in E} N^{(e)}_{S_u \vee S_v, I, 1} \tag{16}$$

Given that $N^{(e)}_{S_u \vee S_v, I} = N^{(e)}_{S_u, I} + N^{(e)}_{S_v, I}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = S_u \vee S_v$, $A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$. Thus, from Eq. (10), $N^{(e)}_{S_u \vee S_v, I} = |\text{Star}_e|.(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$, such that $|\text{Star}_e| = |\text{Star}_u| + |\text{Star}_v|$. Now, from Corollary 11, each 4-path is counted 2 times (for both the start and end edges in the path, denoted by the purple in Fig. 1). Thus, the total count of 4-paths in G is $f(g_{4_6}, G) = \frac{1}{2}.\sum_{e \in E} N^{(e)}_{S_u \vee S_v, I, 1}$. Similarly, from Corollary 12, each 4-node-2-star will be counted 2 times (once for each edge in the star, denoted by the purple in Fig. 1). Thus, the total count of 4-node-2-star in G is $f(g_{4_8}, G) = \frac{1}{2}.\sum_{e \in E} N^{(e)}_{S_u \vee S_v, I, 0}$. By direct substitution in Eq. (16), this lemma is true. □

4.3.7 Relationship between 4-node-2-edges and 4-node-1-edge

Corollary 13 *For any edge $e = (u, v)$ in the graph, the number of 4-node-2-edges where e is any of the two independent edges in the graphlet (denoted by the orange color in Fig. 1) is $N^{(e)}_{I,I,1}$.*

Corollary 14 *For any edge $e = (u, v)$ in the graph, the number of 4-node-1-edge where e is an isolated/single edge in the graphlet (denoted by the orange color in Fig. 1) is $N^{(e)}_{I,I,0}$.*

Lemma 8 *For any graph G, the relationship between the counts of 4-node-2-edge graphlets (i.e., $f(g_{4_9}, G)$) and 4-node-1-edge graphlets (i.e., $f(g_{4_{10}}, G)$) is,*

$$f(g_{4_{10}}, G) = \sum_{e \in E} \binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2} - 2.f(g_{4_9}, G)$$

Proof From Theorem 1 and the addition principle [41], the total count for all edges in G is,

$$\sum_{e \in E} N^{(e)}_{I,I,0} = \sum_{e \in E} N^{(e)}_{I,I} - \sum_{e \in E} N^{(e)}_{I,I,1} \tag{17}$$

Given that $N^{(e)}_{I,I}$ is the number of 4-node subgraphs $\{u, v, w, r\}$ containing e, such that $A_w = I$, $A_r = I$. And, the number of nodes independent of e is $|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|$. Thus, from Eq. (9), $N^{(e)}_{I,I} = \binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2}$. Now, from Corollary 13, each 4-node-2-edge is counted 2 times (for the two edges in the graphlet, denoted by the orange in Fig. 1). Thus, the total count of 4-node-2-edges in G is $f(g_{4_9}, G) = \frac{1}{2}.\sum_{e \in E} N^{(e)}_{I,I,1}$. Similarly, from Corollary 14, each 4-node-1-edge will be counted once (for the isolated/single edge in the graphlet, denoted by the orange in Fig. 1). Thus, the total count of 4-node-1-edge in G is $f(g_{4_{10}}, G) = \sum_{e \in E} N^{(e)}_{I,I,0}$. By direct substitution in Eq. (17), this lemma is true. □

While it is straightforward to compute $N^{(e)}_{I,I}$ for each edge e, this is not the case for $N^{(e)}_{I,I,1}$ or $N^{(e)}_{I,I,0}$, as they require searching outside the local edge neighborhood. However, since $N^{(e)}_{I,I,1}$ is the number of edges outside the egonet of e, it can be computed as,

$$\begin{aligned} N^{(e)}_{I,I,1} = {} & |E| - |\mathcal{N}(u)\backslash\{v\}| - |\mathcal{N}(v)\backslash\{u\}| - |\{e\}| \\ & - [N^{(e)}_{T,T,1} + N^{(e)}_{T,S_u \vee S_v,1} + N^{(e)}_{T,I,1}] \\ & - [N^{(e)}_{S_\cdot,S_\cdot,1} + N^{(e)}_{S_u,S_v,1} + N^{(e)}_{S_\cdot,I,1}] \end{aligned}$$

Thus, the total number of 4-node-2-edges is,

$$\begin{aligned} 2.f(g_{4_9}, G) &= \sum_{e \in E} N^{(e)}_{I,I,1} \\ &= \sum_{e \in E} |E| - |\mathcal{N}(u)\backslash\{v\}| - |\mathcal{N}(v)\backslash\{u\}| - |\{e\}| \\ &\quad - [6.f(g_{4_1}, G) + 4.f(g_{4_2}, G) + 2.f(g_{4_3}, G)] \\ &\quad - [4.f(g_{4_4}, G) + 2.f(g_{4_6}, G)] \end{aligned} \tag{18}$$

Finally, the number of 4-node-independent graphlets ($g_{4_{11}}$) is,

$$f(g_{4_{11}}, G) = \binom{|V|}{4} - \sum_{i=1}^{10} f(g_{4_i}, G) \tag{19}$$

4.4 Algorithm

Algorithm 2 (GRAPHLETCOUNTING) shows how to count all graphlets of size $k = \{3, 4\}$ nodes efficiently (using Lemmas 2–8). We use similar implementation and parallelization approach as in Algorithm 1. As discussed previously, we start by finding all triangle and 2-star patterns in Lines 5–13 (i.e., STEP 1). Then, in Lines 16—17 we only count 4-cliques and 4-cycles (i.e., STEP 2). Then, Lines 19—30 compute unrestricted counts for all 4-node graphlets in constant time (using knowledge from STEP 1 and 2, i.e., STEP 3), and finally Lines 34—36

compute the final counts (using the lemma proved in Sect. 4.3) (i.e., STEP 4). Our approach counts all 4-cliques and 4-cycles in $\mathcal{O}(m.\Delta.T_{max})$ and $\mathcal{O}(m.\Delta.S_{max})$, respectively, where T_{max} is the maximum number of triangles incident to an edge and $T_{max} \ll \Delta$ for sparse graphs, and S_{max} is the maximum number of stars incident to an edge and $S_{max} \leq \Delta$, as we show in Lemmas 9 and 10 . This is more efficient than $\mathcal{O}(|V|.\Delta^3)$ given by [40], and $\mathcal{O}(\Delta.|E| + |E|^2)$ given by [27].

Lemma 9 *Algorithm 2 counts all 4-cliques in $\mathcal{O}(|E|.\Delta.T_{max})$, where T_{max} is the maximum number of triangles incident to an edge.*

Proof For each edge $e = (u, v) \in E$, the runtime complexity of counting all 4-cliques incident to e is equivalent to finding the set of all edges $e' = (w, w')$ such that $\{e' = (w, w') \in E | w, w' \in \text{Tri}_e \wedge w \neq w'\}$, where Tri_e is the set of triangles incident to e. First, we show in Lemma 1 that the runtime complexity of finding all triangles incident to e is $\mathcal{O}(\Delta)$. Second, as described in Algorithm 2 the runtime complexity of checking whether any two distinct nodes $w, w' \in \text{Tri}_e$ are connected by an edge $e' = (w, w')$ is $\mathcal{O}(\sum_{w \in \text{Tri}_e} \Delta) = \mathcal{O}(|\text{Tri}_e|.\Delta)$, and can be computed asymptotically $\mathcal{O}(T_{max}.\Delta)$, where T_{max} is the maximum triangle degree (i.e., the maximum number of triangles incident to an edge and $T_{max} \ll \Delta$). Therefore, the total runtime complexity is $\mathcal{O}\Big(\sum_{e \in E}(\Delta + T_{max}.\Delta)\Big) = \mathcal{O}(|E|.\Delta.T_{max})$. □

Lemma 10 *Algorithm 2 counts all 4-cycles of size $k = 4$ in $\mathcal{O}(|E|.\Delta.S_{max})$, where S_{max} is the maximum number of 2-stars incident to an edge (proof is similar to Lemma 9).*

Proof For each edge $e = (u, v) \in E$, the runtime complexity of counting all 4-cycles incident to e is equivalent to finding the set of all edges $e' = (w, w')$ such that $\{e' = (w, w') \in E | w \in \text{Star}_u \wedge w' \in \text{Star}_v, w \neq w'\}$. First, we show in Lemma 1 that the runtime complexity of finding all 2-star patterns incident to e is $\mathcal{O}(\Delta)$. Second, Algorithm 2 shows the runtime complexity of checking whether any two distinct nodes $w \in \text{Star}_u$, and $w' \in \text{Star}_v$ are connected by an edge $e' = (w, w')$ is $\mathcal{O}(\sum_{w \in \text{Star}_u} \Delta) = \mathcal{O}(|\text{Star}_u|.\Delta)$ and is asymptotically $\mathcal{O}(S_{max}.\Delta)$ (where S_{max} is the maximum number of 2-stars incident to an edge, and $S_{max} \leq \Delta$). Therefore, the total runtime complexity is $\mathcal{O}\Big(\sum_{e \in E}(\Delta + S_{max}.\Delta)\Big) = \mathcal{O}(|E|.\Delta.S_{max})$. □

5 Experiments

We proceed by first demonstrating how fast our algorithm (Algorithm 2) counts all graphlets of size $k = \{3, 4\}$ (both connected and disconnected graphlets) on various networks. We make our parallel implementation available online.[1]

[1] http://nesreenahmed.com/graphlets/.

Algorithm 2 Our exact graphlet census algorithm for counting all 3, 4-node graphlets. The algorithm takes an undirected graph as input and returns the frequencies of all 3, 4-node graphlets

procedure GRAPHLETCOUNTING($G = (V, E)$)
  Initialize all variables
  parallel for $e = (u, v) \in E$ **do**
    $\text{Star}_u = \emptyset$, $\text{Star}_v = \emptyset$, $\text{Tri}_e = \emptyset$
    for $w \in \mathcal{N}(u)$ **do**
      if $w = v$ **then continue**
      Add w to Star_u and set $X(w) = 1$
    for $w \in \mathcal{N}(v)$ **do**
      if $w = u$ **then continue**
      if $X(w) = 1$ **then** ▷ found triangle
        Add w to Tri_e and set $X(w) = 2$
        Remove w from Star_u
      else Add w to Star_v and set $X(w) = 3$
    Compute $f(\mathcal{G}_3, G)$ as in Lines 14–16 of Algorithm 1
    //Get Counts of 4-Cliques & 4-Cycles
    $f(g_{4_1}, G)$ += CLIQUECOUNT(X, Tri_e)
    $f(g_{4_4}, G)$ += CYCLECOUNT(X, Star_u)
    //Get Unrestricted Counts for 4-Node Connected Graphlets
    $N_{T,T}$ += $\binom{|\text{Tri}_e|}{2}$
    N_{S_u,S_v} += $|\text{Star}_u|.|\text{Star}_v|$
    $N_{T,S_u \vee S_v}$ += $|\text{Tri}_e|.(|\text{Star}_u| + |\text{Star}_v|)$
    $N_{S_u,S_u} = \binom{|\text{Star}_u|}{2}$ and $N_{S_v,S_v} = \binom{|\text{Star}_v|}{2}$
    $N_{S_\cdot,S_\cdot}$ += $N_{S_u,S_u} + N_{S_v,S_v}$
    //Get Unrestricted Counts for 4-Node Disconnected Graphlets
    $N_{T,I}$ += $\text{Tri}_e.(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
    $N_{S_u,I} = |\text{Star}_u|.(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
    $N_{S_v,I} = |\text{Star}_v|.(|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|)$
    $N_{S_u \vee S_v,I}$ += $N_{S_u,I} + N_{S_v,I}$
    $N_{I,I}$ += $\binom{|V| - |\mathcal{N}(u) \cup \mathcal{N}(v)|}{2}$
    $N_{I,I,1}$ += $|E| - |\mathcal{N}(u) \backslash \{v\}| - |\mathcal{N}(v) \backslash \{u\}| - 1$
    for $w \in \mathcal{N}(v)$ **do** $X(w) = 0$
  end parallel
  Aggregate counts from all workers
  Use Lemma 2–6 to compute $f(g_{4_i}, G)$ for $i = 1 : 8$
  Use Eq. (18) to compute $f(g_{4_9}, G)$ and Lemma 8 for $f(g_{4_{10}}, G)$
  Use Eq. (19) to compute $f(g_{4_{11}}, G)$
  return $f(\mathcal{G}_3, G)$, $f(\mathcal{G}_4, G)$

procedure CLIQUECOUNT(X, Tri_e)
  $\text{cliq}_e = 0$
  for each node $w \in \text{Tri}_e$ **do**
    for $r \in \mathcal{N}(w)$ **do**
      if $X(r) = 2$ **then** cliq_e += 1 ▷ found 4-Clique
    $X(w) = 0$
  return cliq_e

procedure CYCLECOUNT(X, Star_u)
  $\text{cyc}_e = 0$
  for each node $w \in \text{Star}_u$ **do**
    for $r \in \mathcal{N}(w)$ **do**
      if $X(r) = 3$ **then** cyc_e += 1 ▷ found 4-Cycle
    $X(w) = 0$
  return cyc_e

In this paper, we show detailed results for 60 networks categorized in 8 broad classes from social, Facebook [43], biological, web, technological, co-authorship, infrastructure, among other domains [35] (see the links[2] for data download). And, in online appendix, we present a more extensive collection of 300+ networks, including both large sparse networks and dense networks from the DIMACs challenge.[3] Note that for all of the networks, we discard edge weights, self-loops, and edge direction.

To the best of our knowledge, this is the largest study for graphlet counting, and these are the largest graphlet computations published to date. Our own implementation of Algorithm 2 is using shared memory, but the algorithm is well suited for other architectures.

5.1 Efficiency and runtime

Table 2 describes the properties of the 60 networks considered here. It also shows the counts of graphlets of size $k = \{3, 4\}$ and states the time (seconds) taken to count all graphlets. We only show counts of connected graphlets due to space limitations; however, all counts are available in online appendix. Notably, Algorithm 2 takes only few seconds to count all graphlets for large social, web, and technological graphs (among others). For example, for a large road network (i.e., inf-road-usa) with 24M nodes and 29M edges, Algorithm 2 takes only 4 s to count all graphlets. Also as shown in Table 2, for large Facebook networks with nearly 2M edges, Algorithm 2 takes only 15 s, and for large web graphs with nearly 8M edges, Algorithm 2 takes only 25 s.

We compare the empirical runtime of Algorithm 2 to the state-of-the-art baseline method RAGE [27]. For social and Facebook networks, we observed that Algorithm 2 is on average 460× faster than RAGE. For all other networks, we observed that Algorithm 2 is on average 600× faster than RAGE. Notably, Algorithm 2 takes only 7 s to count graphlets of Facebook networks with 1.3M edges, while RAGE takes almost an hour for the same networks. For larger networks with millions of nodes/edges, RAGE was timed out (as it did not finish within 30 h of runtime). Moreover, for dense graphs from the DIMACS challenge, RAGE takes almost 17 min, while Algorithm 2 takes less than a second. We also compared to the baseline method FANMOD [47] and Orca [20], and we found that for a Facebook network with 250k edges, FANMOD takes roughly 2.5 h for counting all graphlets, RAGE takes almost 7 min for the same network, and Orca takes almost 10 s, while Algorithm 2 takes less than a second. Note that both RAGE and Orca count only connected graphlets, while our algorithm and FANMOD count both connected and disconnected graphlets.

In Fig. 3, we plot the runtime of Algorithm 2 for a representative subset of 150 social and information networks. The figure shows that our algorithm exhibits nearly linear-time scaling over networks ranging from 1K to 100M nodes.

5.2 Scaling

We used a dual-processor Intel Xeon 3.10 Ghz E5-2687W server; each processor has 8 cores, and each core can run two threads. The two processors share 20MB of L3 cache and 256GB of memory. We evaluate the speedup of our parallel algorithm (i.e., how much faster our proposed algorithm is when we increase the number of cores), and we used the OpenMP library for multi-core parallelization. In the following plots, we show the speedups versus the number of processing units (cores). All speedups are computed relative to the runtime of

[2] http://networkrepository.com/.

[3] http://dimacs.rutgers.edu/Challenges/.

Table 2 Runtime and statistics for a subset of 60 networks

Graph	$\|V\|$	$\|E\|$	$\|g_{3_1}\|$	$\|g_{3_2}\|$	$\|g_{4_1}\|$	$\|g_{4_2}\|$	$\|g_{4_4}\|$	$\|g_{4_6}\|$	$\|g_{4_5}\|$	$\|g_{4_3}\|$	Seconds	
											Algorithm 2	RAGE
soc-brightkite	57k	213k	494k	12M	2.9M	12M	2.7M	533M	1.3B	114M	0.2	273.03
socfb-Berkeley13	23k	852k	5.4M	125M	27M	153M	87M	17B	25B	2.7B	4.94	2514.59
socfb-Wisconsin87	24k	836k	4.9M	107M	23M	121M	59M	12B	21B	1.9B	3.93	1450.31
socfb-FSU53	28k	1.0M	7.9M	130M	63M	242M	95M	16B	10B	2.9B	5.55	2192.94
socfb-MSU24	32k	1.1M	6.5M	139M	33M	183M	106M	16B	32B	2.6B	5.67	1904.09
socfb-Texas80	32k	1.2M	9.6M	160M	68M	316M	122M	21B	11B	3.9B	7.53	2967.01
socfb-Michigan23	30k	1.2M	8.3M	162M	49M	277M	146M	23B	13B	3.5B	7.57	2995.83
socfb-Indiana69	30k	1.3M	9.4M	181M	60M	269M	141M	25B	13B	3.8B	8.44	3212.10
socfb-UIllinois20	31k	1.3M	9.4M	172M	64M	273M	130M	23B	27B	3.8B	7.88	3088.77
socfb-UF21	35k	1.5M	12M	266M	98M	433M	186M	40B	150B	7.2B	14.49	N/A
soc-flickr	514k	3.2M	59M	963M	1.7B	14B	6.7B	244B	326B	90B	182.57	N/A
soc-orkut	3.1M	117M	628M	44B	3.2B	48B	70B	19T	98T	1.5T	2694.55	N/A
soc-sinaweibo	58M	261M	212M	804B	662M	27B	259B	157T	8.48P	3.80T	33359.7	N/A
soc-friendster	65.6M	1.8B	4.17B	708.1B	8.96B	131.4B	307.5B	364.7T	247.3T	5.79T	N/A	N/A
bio-celegans	453	2.0k	3.3k	69k	3.0k	37k	4.5k	495k	2.9M	363k	<0.001	1.7
bio-diseasome	516	1.2k	1.4k	5.4k	1.4k	923	42	18k	27k	19k	<0.001	0.44
bio-dmela	7.4k	26k	2.9k	572k	393	13k	107k	11M	9.2M	312k	0.01	2.47
bio-yeast-protein-inter	1.8k	2.2k	222	11k	41	198	140	31k	72k	2.6k	<0.001	0.53
bio-yeast	1.5k	1.9k	206	11k	39	195	139	31k	72k	2.5k	<0.001	0.43
bio-human-gene2	14k	9.0M	4.9B	10B	2.3T	3.7T	90B	4.4T	5.3T	8.4T	8023.84	N/A
bio-mouse-gene	43k	14M	3.6B	15B	670B	2.1T	223B	9.0T	6.7T	7.7T	5515.6	N/A
ca-CSphd	1.9k	1.7k	8	6.6k	0	5	8	9.4k	32k	93	<0.001	1.25

Table 2 continued

Graph	$\|V\|$	$\|E\|$	$\|g3_1\|$	$\|g3_2\|$	$\|g4_1\|$	$\|g4_2\|$	$\|g4_4\|$	$\|g4_6\|$	$\|g4_5\|$	$\|g4_3\|$	Seconds	
											Algorithm 2	RAGE
ca-GrQc	4.2k	13k	48k	85k	329k	66k	1.1k	553k	406k	628k	<0.001	5.99
ca-dblp-2012	317k	1.0M	2.2M	15M	17M	4.8M	203k	252M	259M	97M	0.48	227.79
ca-cit-HepTh	23k	2.4M	191M	1.6B	13B	47B	7.3B	538B	976B	385B	132.66	N/A
ca-cit-HepPh	28k	3.1M	196M	1.5B	9.8B	34B	6.1B	536B	479B	276B	125.49	N/A
ca-coauthors-dblp	540k	15M	444M	698M	15B	3.4B	31M	42B	27B	67B	40.26	N/A
ca-hollywood-2009	1.1M	56M	4.9B	33B	1.4T	635B	168B	21T	17T	8.9T	13799.6	N/A
tech-as-caida2007	26k	53k	36k	15M	54k	1.7M	407k	285M	7.8B	47M	0.19	36.83
tech-p2p-gnutella	63k	148k	2.0k	1.6M	16	826	42k	15M	8.1M	71k	0.02	7.44
tech-RL-caida	191k	608k	455k	21M	423k	7.4M	40M	583M	1.7B	77M	0.39	71.74
tech-WHOIS	7.5k	57k	782k	5.3M	12M	31M	2.9M	229M	566M	194M	0.14	44.52
tech-as-skitter	1.7M	11M	29M	16B	149M	20B	43B	819B	96T	162B	476.06	N/A
web-BerkStan-dir	685k	6.6M	65M	28B	1.1B	99B	25B	49B	382T	476B	149.17	N/A
web-edu	3.0k	6.5k	10k	81k	40k	4.6k	18	435k	1.3M	186k	<0.001	0.52
web-google-dir	876k	4.3M	13M	687M	40M	382M	38M	4.1B	650B	6.7B	4.45	N/A
web-indochina-2004	11k	48k	210k	481k	1.2M	88k	9.2k	5.5M	12M	4.9M	0.01	24.36
web-it-2004	509k	7.2M	339M	56M	29B	815M	175M	1.1B	1.4B	527M	25.26	N/A
web-baidu-baike	2.1M	17M	25M	31B	28M	4.5B	9.2B	3.3T	571T	327B	3975.81	N/A
web-wikipedia-growth	1.9M	37M	127M	123B	288M	38B	68B	29T	3.1P	3.2T	22389.2	N/A
web-ClueWeb09-50m	148M	447M	1.2B	494B	5.6B	243B	774B	34T	24P	3.4T	15665.9	N/A
inf-italy-osm	6.7M	7.0M	7.4k	8.2M	0	244	47k	9.9M	992k	27k	0.85	N/A
inf-openflights	2.9k	16k	73k	639k	286k	1.5M	319k	17M	17M	9.0M	0.01	2.46
inf-power	4.9k	6.6k	651	17k	90	385	324	38k	20k	5.1k	<0.001	0.58

Table 2 continued

Graph	$\|V\|$	$\|E\|$	$\|g3_1\|$	$\|g3_2\|$	$\|g4_1\|$	$\|g4_2\|$	$\|g4_4\|$	$\|g4_6\|$	$\|g4_5\|$	$\|g4_3\|$	Seconds	
											Algorithm 2	RAGE
inf-roadNet-CA	2.0M	2.8M	120k	5.6M	40	13k	249k	11M	2.4M	521k	0.35	N/A
inf-roadNet-PA	1.1M	1.5M	67k	3.2M	16	5.7k	152k	6.2M	1.4M	295k	0.19	N/A
inf-road-usa	24M	29M	439k	50M	90	21k	1.6M	81M	18M	1.5M	4.05	N/A
ia-email-EU-dir	265k	364k	267k	194M	581k	10M	6.7M	4.4B	221B	341M	1.52	887.18
ia-enron-only	143	623	889	4.8k	779	2.7k	648	29k	17k	14k	<0.001	0.12
ia-reality	6.8k	7.7k	400	497k	63	1.7k	2.8k	1.6M	26M	93k	<0.001	1.39
ia-wiki-Talk-dir	2.4M	4.7M	9.2M	13B	65M	1.0B	924M	1.2T	192T	64B	281.33	N/A
ia-wikiquote-user-edits	93k	238k	279k	636M	411k	70M	44M	8.9B	2.4T	2.5B	2.41	691.28
ia-wiki-user-edits-page	2.1M	5.6M	6.7M	550B	10M	70B	44B	4.8T	88P	2.0T	5691.92	N/A
brock200-3	200	12k	291k	570k	3.2M	12M	4.1M	11M	3.5M	16M	0.02	22.96
brock200-4	200	13k	373k	584k	5.2M	16M	4.3M	8.9M	3.0M	17M	0.02	21.85
brock400-3	400	60k	4.4M	4.5M	184M	372M	63M	84M	28M	251M	0.4	997.15
brock400-4	400	60k	4.4M	4.5M	185M	373M	63M	84M	28M	250M	0.4	1010.26
brock800-1	800	208k	23M	38M	1.3B	4.1B	1.1B	2.4B	801M	4.4B	4.11	N/A
brock800-2	800	208k	23M	38M	1.3B	4.2B	1.1B	2.4B	794M	4.4B	4.15	N/A
brock800-3	800	207k	23M	38M	1.3B	4.1B	1.1B	2.4B	802M	4.4B	4.1	N/A

The numbers are appended by K for thousands, M for millions, B for billions, T for trillions, and P for quadrillions
N/A denotes algorithm was timed out after 30 h of runtime

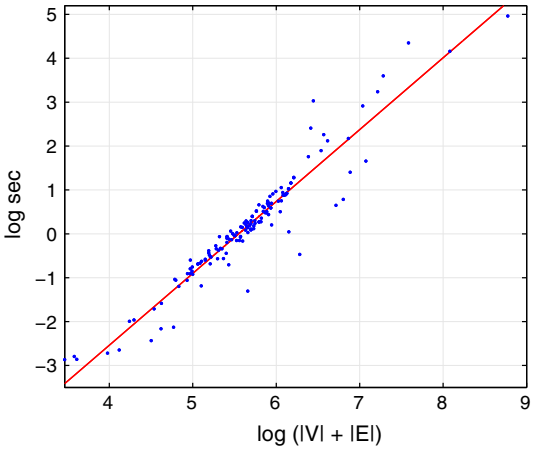

Fig. 3 The empirical runtime of our exact graphlet counting (Algorithm 2) in social and information networks scales almost linearly with the network dimension

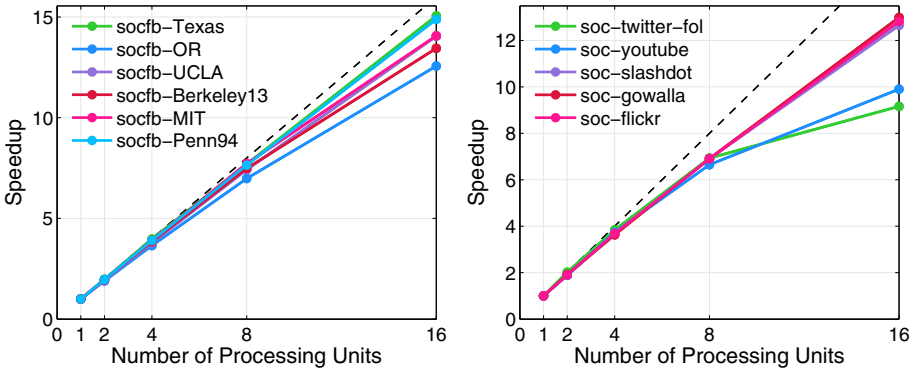

Fig. 4 Strong scaling results for Facebook and social networks

Algorithm 2 with one processor. To reduce the possible variance, all experiments are repeated 5 times and averaged. Figures 4 and 5 show the speedup plots for a variety of graphs. We discuss a few observations from the plots presented here.

The first and most important observation that we make is that we obtain significant speedups from the parallel implementation of Algorithm 2. Figures 4–5 show strong scaling results for a variety of graphs from social, web, and technological domains. Algorithm 2 scales to 16 cores and yields a 10- to 15-fold speedup. For example, as shown in Fig. 4, we achieve almost linear scaling for the socfb-Penn94 graph (15-fold speedup for 16 cores).

The second observation links the performance of Algorithm 2 to the characteristics of the graphs. We observe that we obtain the most significant speedups for social and Facebook networks (see Fig. 4). We obtain near linear speedup as we increase the number of cores. Social networks are computationally intensive relative to the other graphs. This is due to their clustering characteristics and the existence of a large number of small communities (i.e., triangles, cliques, and cycles) in social networks.

Finally, we study the optimal number of problems to dynamically assign to each processing unit when more work is requested (i.e., batch size b). That is the optimal performance that would be achieved when b jobs are assigned in batch. Overall, we observed small performance fluctuations and found the optimal value of b when we changed between 1 and 256 edges,

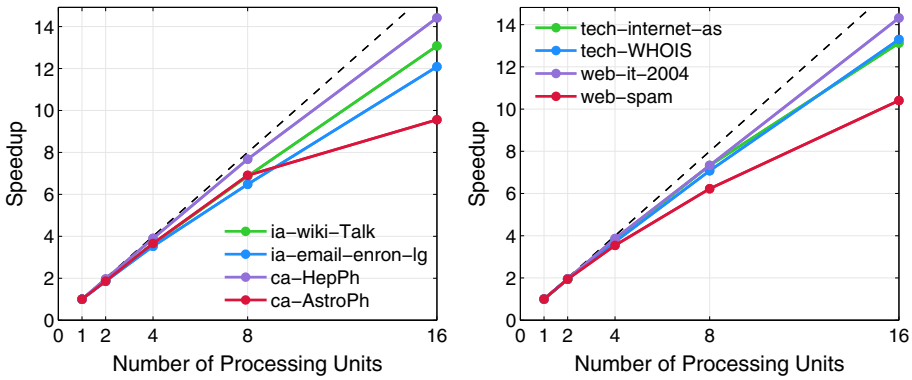

Fig. 5 Strong scaling results for interaction, collaboration, technological, and web networks

respectively. Interestingly, this observation is largely true only for sparse graphs, whereas graphs that are relatively dense (e.g., DIMACs graphs) work better when b is small (e.g., even as small as $b = 1$). This is likely due to the properties of these graphs and the auto-optimizer that we built into the library which automatically adapts the implementation of the algorithms to use additional data structures and achieve better performance for those relatively dense graphs at the cost of using additional space. Thus, our auto-optimizer appropriately balances the time and space trade-offs.

Note that the results for the job size experiments use node degree for ordering the neighbors of each node in the succinct graph representation as well as for ordering the edge jobs to solve. In both cases, the ordering is from largest to smallest.

6 Applications

We also show some applications that could benefit from our fast graphlet counting algorithm (Algorithm 2), which facilitates exploring and understanding networks and their structure. Graphlets provide an intuitive and meaningful characterization of a network at the global macro-level as well as the local micro-level; thus, they are useful for numerous applications. At the macro-level, graphlets are useful for finding similar networks (graph similarity queries), or finding networks that disagree most with that set (graph anomalies), or exploring a time series of networks, among numerous other possibilities. Alternatively, graphlets are also extremely useful for characterizing networks and their behavior at the local node/edge level as known as the micro-level. For instance, given an edge $(u, v) \in E$, find the top-k most similar edges (with applications in security, role discovery, entity resolution, link prediction, and other related matching/similarity applications). Also, graphlets could be used for ranking nodes/edges to find unique patterns and anomalies such as large stars, cliques.

6.1 Large-scale graph comparison and classification

Graphlets are also useful for large-scale comparison and classification of graphs. In this case, we relax the notion of equivalence and isomorphism to some form of *structural similarity* between graphs such that the graph similarity is measured using feature-based graphlet counts. In this section, we show how graphlets could be useful for network analysis, anomaly detection, and graph classification.

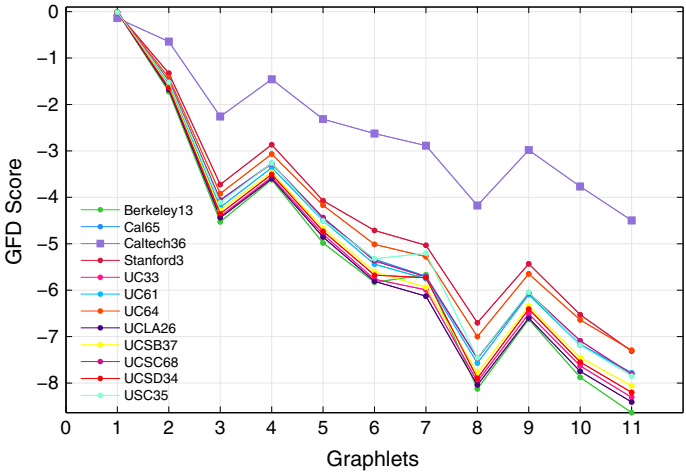

Fig. 6 Facebook social networks of California Universities. Using the space of graphlets of size $k = 4$, Caltech is noticeably different than others, which is consistent with the findings in [43]

Table 3 Accuracy and standard error for classification of large collection of biological and chemical graphs

Graph	Type	No. graphs	Accuracy (%)	Total time (s)	Avg time per G (s)
D&D	Protein	1178	76.13 ± 0.03	1.05	8.95×10^{-4}
MUTAG	Chemicals	188	86.4 ± 0.21	0.14	7.47×10^{-4}

We used counts of all graphlets of size $k = \{2, 3, 4\}$ as features

First, we study the full data set of Facebook100, which contains 100 Facebook networks that represent a variety of US schools [43]. We plot the GFD (i.e., graphlet frequency distribution) score pictorially in Fig. 6 for all California schools. The GFD score is simply the normalized frequencies of graphlets of size k [33]. In our case, we use $k = 4$. The figure shows Caltech noticeably different than others, consistent with the results in [43] which shows how Caltech is well known to be organized almost exclusively according to its undergraduate "Housing" residence system, in contrast to other schools that follow the predominant "dormitory" residence system. The residence system seems to impact the organization of the social community structures at Caltech.

Second, we use counts of graphlets of size $k = \{2, 3, 4\}$-nodes as features to represent each graph in a large collection of graphs. Using the graphlet feature representation, we learn a model to predict the unknown label of each unlabeled graph (e.g., the label could be the function of protein graphs). We test our approach on protein graphs (D&D collection of 1178 protein graphs) and chemical compound graphs (MUTAG collection of 188 chemical compound graphs) [45]. We extract the graphlet features using Algorithm 2. Then, we learn a model using SVM (RBF kernel), and we use 10-fold validation for evaluation. Table 3 shows the accuracy of this approach is 76 % for protein function prediction, and 86 % for mutagenic effect prediction. Note that by using all graphlet-based features up to size 4 nodes, we were able to obtain better accuracy than previous work (which achieved maximum 75 and 83 % accuracy for D&D and MUTAG, respectively [40]). Moreover, Algorithm 2 extracts all the features (graphlet counts) in nearly 1 s. This yields a significant improvement over the

graphlet feature extraction approach that was proposed in [40], which takes 2.45 h to extract graphlet features from the D&D collection.

Third, we compute graphlet counts on a 2 billion edge social network called Friendster. Friendster is an online gaming network. Before re-launching as a game Web site in 2011, Friendster was an online social network where users can form friendship links with each others. These data are provided by The Web Archive Project before the death of the social network. In these experiments, we use the induced subgraph of the nodes that either belong to at least one community or are connected to other nodes that belong to at least one community. Table 2 shows a significantly large number of 4-path (chains of 4 connected nodes) and 3-stars compared to the number of 4-cliques and triangles. Although the induced subgraph that we used from Friendster is clearly biased toward communities, the patterns that represent communities, such as cliques and triangles, are less likely in the induced graph. For example, the frequency of 4-path patterns is 0.58, while the frequency of 4-cliques is 0.000014. These results indicate that something wrong happened to the social network. Previous work on the autopsy of Friendster showed that there was a collapse in the community structure of Friendster, a cascade in user departure due to bad decisions in the design and interface changes. In a similar fashion, the low frequency of community-related graphlets (e.g., cliques) in Friendster also indicates the collapse of the social network.

6.2 Finding large stars, cliques, and other patterns fast

How can we quickly and efficiently find large cliques, stars, and other unique patterns? Further, how can we identify the top-k largest cliques, stars, etc.? Note that many of these problems are NP-hard, e.g., finding the clique of maximum size is a well-known NP-hard problem [17]. To answer these and other related queries, we leverage the proposed parallel graphlet counting method in Algorithm 2.

Figure 7 provides a visualization of the human diseasome network [14], where we used Algorithm 2 to rank (weight) all the edges in the network by the number of star patterns of size 4 nodes. The intuition behind the method is that if an edge (or node) has a (relatively) large number of stars of 4 nodes (cliques, or another graphlet of interest), then it is also likely to be part of a star of a large size. Recall that removing a node from a k-star or k-clique forms a star or clique of size $k-1$ [17]. Accordingly, edges with large weights are likely to be members of large stars. Thus, as shown in Fig. 7, a visualization based on our fast graphlet counting method can help to quickly highlight such large stars by using the counts (of stars of size 4 nodes) as edge weights or colors.

Notably, Fig. 7 highlights the few phenotypes (such as colon cancer, leukemia, and deafness) corresponding to hubs (large stars) that are connected to a large number of distinct disorders, which is consistent with the findings in [14]. Note that the same approach is also applicable for finding cliques and other interesting patterns, since edges with a high number of 4-cliques are likely to be members of the largest clique in the network. Figure 8 shows how we can find large cliques in the terroristRel data [50].

6.3 Real-time visual graphlet mining

Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces [6,42]. This work develops an interactive visual graph analytics platform based on the proposed fast graphlet decomposition algorithm. In particular, we integrate interactive

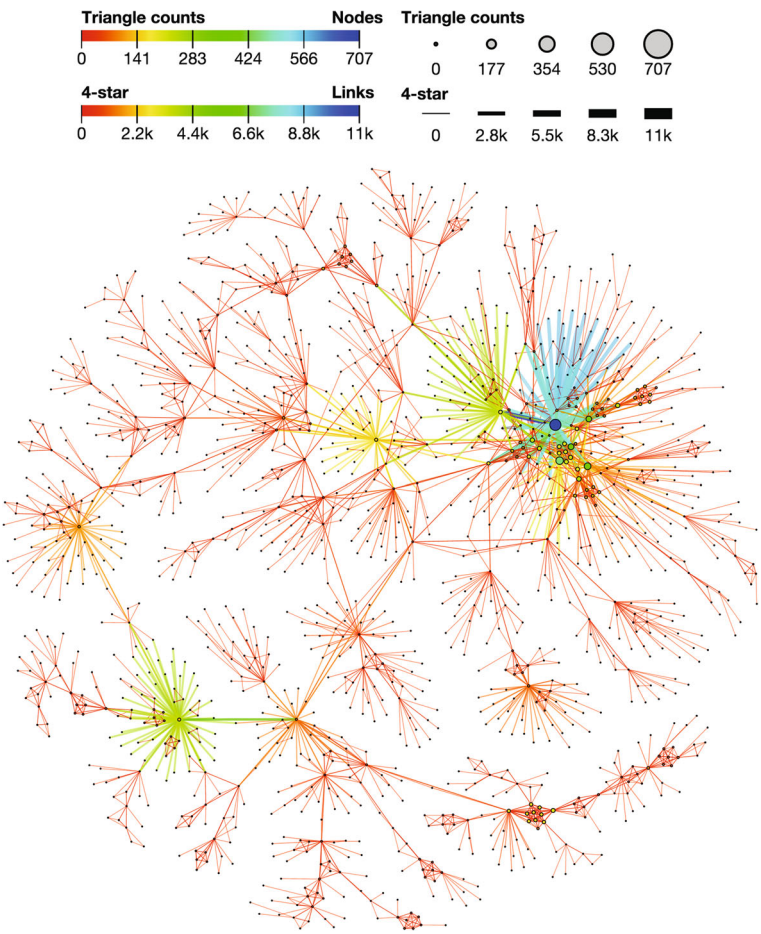

Fig. 7 *Visualization of the human diseasome network*: A network of disorders and disease genes linked by known disorder–gene associations [14]. Edges are weighted/colored by their number of incident star graphlets of size 4 nodes; nodes are weighted/colored by their triangle counts. The large star on the right denoted by *light blue color* corresponds to colon cancer; the large star on the lower *left* denoted by *lime green color* corresponds to deafness; and the large star on the *right* denoted by *lime green color* corresponds to leukemia. Notably, this figure highlights the few phenotypes (such as colon cancer, leukemia, and deafness) corresponding to hubs (large stars) that are connected to a large number of distinct disorders, which is consistent with [14] (color figure online)

visualization with our state-of-the-art parallel graphlet decomposition algorithm in order to support discovery, analysis, and exploration of such data in real-time. We utilize this multi-level graphlet analysis engine that uses graphlets as a basis for exploring, analyzing, and understanding complex networks interactively in real time. And, we highlight other key aspects including filtering, querying, ranking, manipulating, and a variety of multi-level network analysis and statistical techniques based on graphlets.

Notably, our proposed algorithm is shown to be fast and efficient for *real-time* interactive exploration and mining of graphlets. We expect this tool to be extremely useful to biologists and others interested in understanding biological (protein, brain networks, etc.) as well as chemical networks. There are a number of important and unique challenges in designing

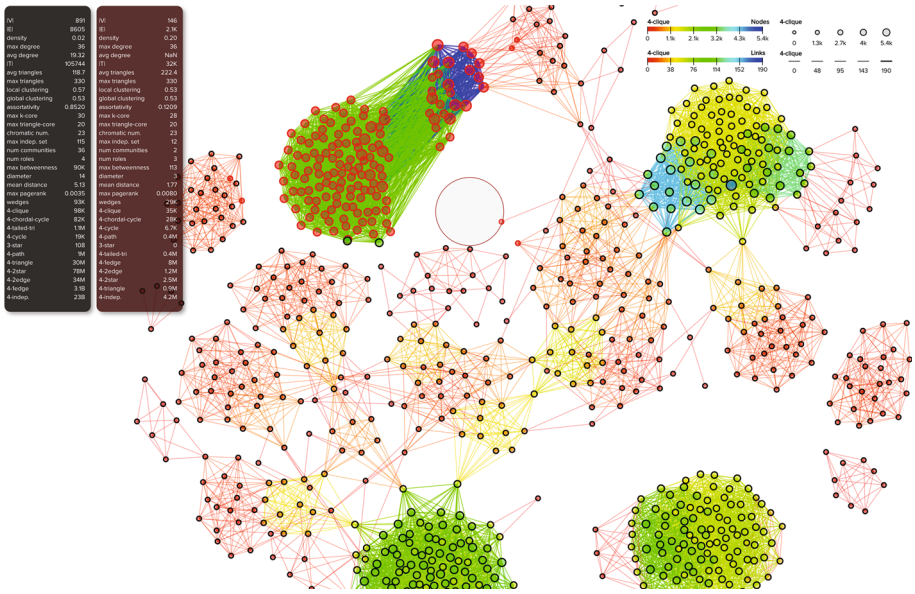

Fig. 8 *Visualization of the terroristRel network*: A network of terrorists and their relationships. Terrorists are linked to each other if they contact each other, use the same facility, are members of the same family, or belong to the same terrorist organization. Edges and nodes are weighted/colored by their number of incident cliques of size 4 nodes. Notably, this figure highlights how the structure of terrorist networks is decomposed of various clique patterns (terrorist organization) and how these cliques are interconnected. The figure highlights the largest clique on the *top left* denoted by *dark blue* (color figure online)

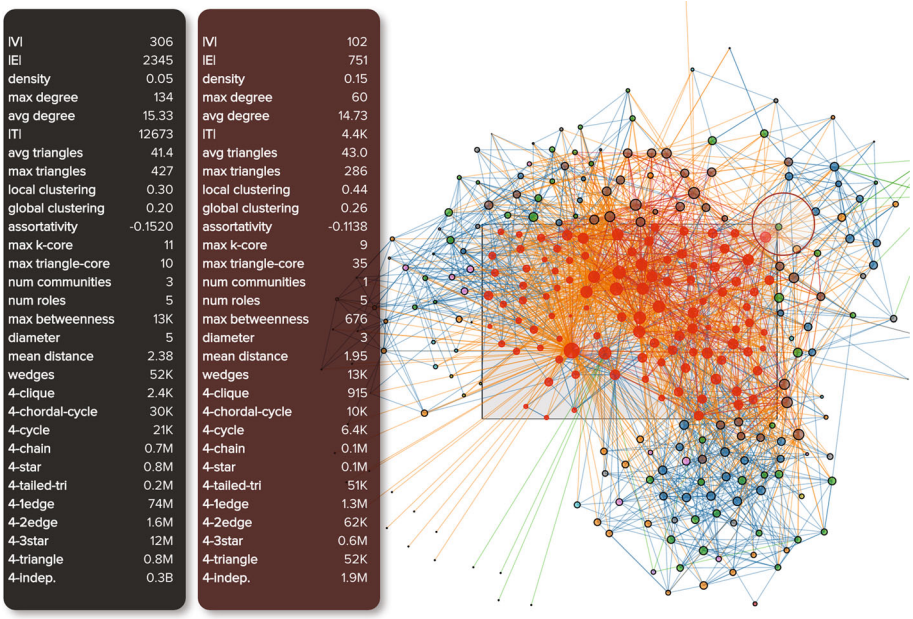

Fig. 9 Exploration of the brain neural network of C. Elegans [46] using our *interactive graphlet visual analytics tool*. Nodes are colored by their k-core number and weighted by betweenness, whereas the links are colored by eccentricity

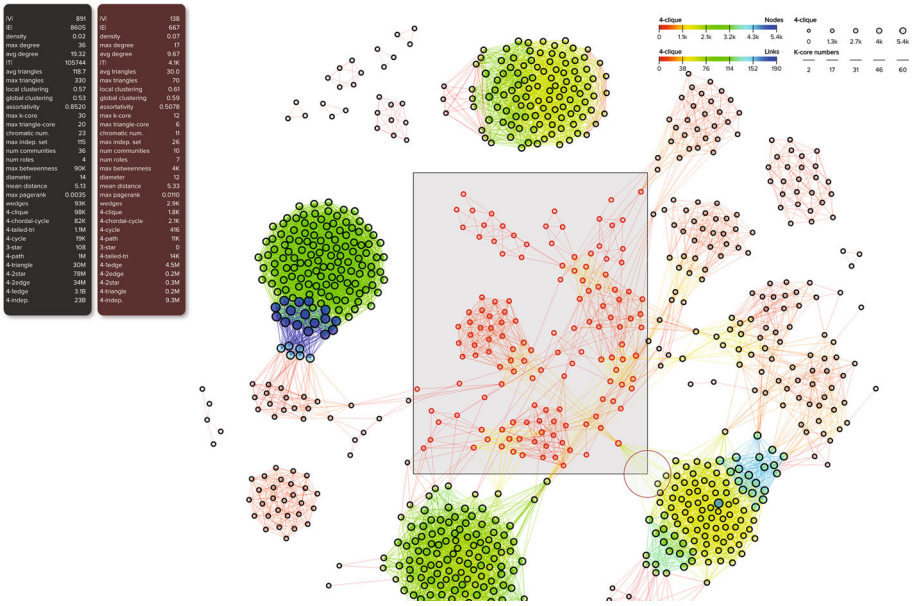

Fig. 10 Illustration of the graphlet methods for real-time interactive graphlet analysis. This demonstrates the efficiency and effectiveness of the proposed methods for interactive real-time graphlet computations. In the screenshot above, the user selects a subgraph to interactively analyze via direct manipulation of the visualization using the mouse. That is, the user adjusts the rectangular region above to highlight the subgraphs to analyze. The graphlet statistics are updated each time a node/edge is added or removed from the rectangular region used to select the subgraph to explore via graphlets. Thus, the user can see how the graphlet statistics change as nodes and edges are added (or removed) from the user-specified rectangular region (which in turn indicates the nodes and edges to include in the analysis). Note that we leverage localized graphlet update methods to achieve the performance required for real-time interactive graphlet mining and sense-making

methods for interactive exploration and mining of graphlets in real time. In particular, the real-time requirement of such a system is fast parallel methods to achieve real-time interactive rates (e.g., with response times within microseconds or less). In particular, we derived dynamic update methods that are localized, that is, the update methods leverage the local structure of the graph for efficiently updating the counts when nodes/edges are selected, inserted, removed, etc. Thus, given a single node or edge, the method updates the graphlet counts for that edge (as opposed to recomputing the full graphlet decomposition).

Figure 9 uses the interactive graphlet mining tool for real-time exploration of the brain neural network from C. Elegans [46]. Additionally, the tool is also useful for exploring many other types of networks, e.g., a terrorist relationship network is shown in Fig. 10, whereas Fig. 11 uses graphlets as a basis for understanding and characterizing the communities and their structure. As an aside, the graph in Fig. 11 is generated using the block Chung-Lu graph model. Thus, it is straightforward to see how graphlets can be used to characterize synthetic graph generators and for evaluating their utility (e.g., if the synthetic graph preserves the distribution of graphlets observed in a real-world network.).

The *visual graphlet analytics tool* is designed for rapid interactive visual exploration and graph mining (Figs. 9, 10 and 11). Graphlets are computed on-the-fly upon a simple drag-and-drop of a graph file into the web browser. Additionally, the graphlet counts are updated efficiently after each selection, insertion, deletion, or change to the graph data. Furthermore,

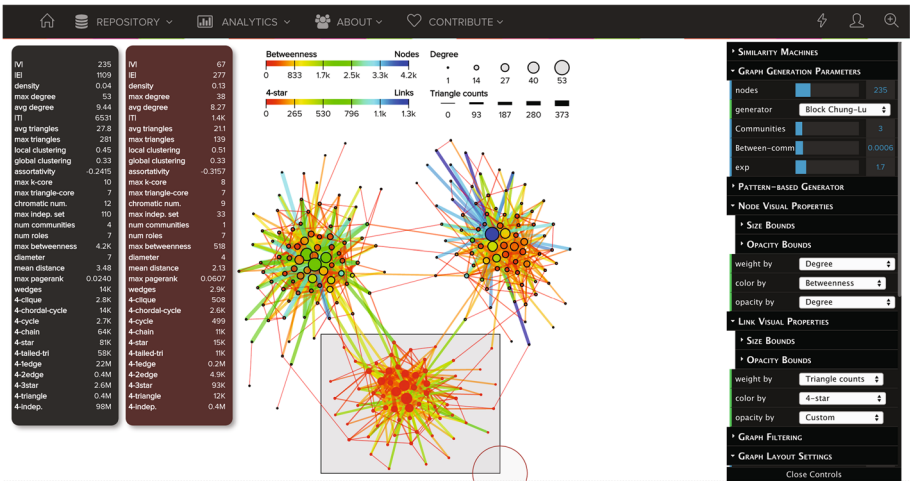

Fig. 11 Interactive graphlet exploration of community structure via direct manipulation and selection of the visual representation

it is designed to be consistent with the way humans learn via immediate feedback upon every user interaction (e.g., change of a slider for filtering) [1,42]. Users have rapid, incremental, and reversible control over all graph queries with immediate and continuous visual feedback.

7 Conclusion and future work

In this paper, we proposed a fast, efficient, and parallel framework as well as a family of algorithms for counting k-node graphlets. We provided a novel theoretical analysis of combinatorial arguments that capture the relationships between various graphlets. The proposed framework leverages these combinatorial arguments to obtain significant improvement on the scalability of graphlet counting. For each edge, we count a few graphlets and obtain the exact counts of others in constant time using the combinatorial arguments. We systematically investigate the scalability of our algorithm on a large collection of 300+ networks from a variety of domains. Finally, we provided a systematic investigation across a variety of existing and new applications for graphlet counting, such as finding unique patterns in graphs, graph similarity, and graph classification. In future work, we aim to extend our proposed framework to higher-order graphlets, as well as extend our approach to the distributed setting.

References

1. Ahlberg C, Williamson C, Shneiderman B (1992) Dynamic queries for information exploration: an implementation and evaluation. In: Proceedings of SIGCHI, pp 619–626
2. Ahmed NK, Duffield N, Neville J, Kompella R (2014) Graph sample and hold: a framework for big-graph analytics. In: SIGKDD
3. Ahmed NK, Neville J, Kompella R (2010) Reconsidering the foundations of network sampling. In: Proceedings of the 2nd Workshop on Information in Networks
4. Ahmed NK, Neville J, Kompella R (2012) Space-efficient sampling from social activity streams. In: Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications, pp 53–60

5. Ahmed NK, Neville J, Kompella R (2014) Network sampling: from static to streaming graphs. ACM Trans Knowl Discov Data (TKDD) 8(2):1–56
6. Ahmed NK, Rossi RA (2015) Interactive visual graph analytics on the web. In: Proceedings of the Ninth International AAAI Conference on Web and Social Media
7. Becchetti L, Boldi P, Castillo C, Gionis A (2008) Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: SIGKDD
8. Bhuiyan MA, Rahman M, Rahman M, Al Hasan M (2012) Guise: uniform sampling of graphlets for large graph analysis. In: ICDM
9. Costa F, De Grave K (2010) Fast neighborhood subgraph pairwise distance kernel. In: ICML
10. Faust K (2010) A puzzle concerning triads in social networks: graph constraints and the triad census. Soc Netw 32(3):221–233
11. Feldman D, Shavitt Y (2008) Automatic large scale generation of internet pop level maps. In: IEEE GLOBECOM
12. Frank O (1988) Triad count statistics. Ann Discrete Math 38:141–149
13. Getoor L, Taskar B (2007) Introduction to statistical relational learning. MIT Press, Cambridge
14. Goh K-I, Cusick ME, Valle D, Childs B, Vidal M, Barabási A-L (2007) The human disease network. PNAS 104(21):8685–8690
15. Gonen M, Shavitt Y (2009) Approximating the number of network motifs. Internet Math 6(3):349–372
16. Granovetter M (1983) The strength of weak ties: a network theory revisited. Sociol Theory 1(1):201–233
17. Gross JL, Yellen J, Zhang P (2013) Handbook of graph theory, 2nd edn. Chapman & Hall, London
18. Hales D, Arteconi S (2008) Motifs in evolving cooperative networks look like protein structure networks. J Netw Heterog Media 3(2):239–249
19. Hayes W, Sun K, Pržulj N (2013) Graphlet-based measures are suitable for biological network comparison. Bioinformatics 29(4):483–491
20. Hočevar T, Demšar J (2014) A combinatorial approach to graphlet counting. Bioinformatics 30(4):559–565
21. Holland PW, Leinhardt S (1976) Local structure in social networks. Sociol Methodol 7:1–45
22. Kashima H, Saigo H, Hattori M, Tsuda K (2010) Graph kernels for chemoinformatics. Chemoinformatics and advanced machine learning perspectives: complex computational methods and collaborative techniques, p 1
23. Kelly PJ (1957) A congruence theorem for trees. Pac J Math 7(1):961–968
24. Kloks T, Kratsch D, Müller H (2000) Finding and counting small induced subgraphs efficiently. Inf Process Lett 74(3):115–121
25. Kuchaiev O, Milenković T, Memišević V, Hayes W, Pržulj N (2010) Topological network alignment uncovers biological function and phylogeny. J R Soc Interface 7(50):1341–1354
26. Manvel B, Stockmeyer PK (1971) On reconstruction of matrices. Math Mag 44:218–221
27. Marcus D, Shavitt Y (2012) Rage—a rapid graphlet enumerator for large networks. Comput Netw 56(2):810–819
28. McKay BD (1997) Small graphs are reconstructible. Australas J Comb 15:123–126
29. Milenkoviæ T, Pržulj N (2008) Uncovering biological network function via graphlet degree signatures. Cancer Inform 6:257
30. Milenković T, Ng WL, Hayes W, Pržulj N (2010) Optimal network alignment with graphlet degree vectors. Cancer Inform 9:121
31. Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. Science 298(5594):824–827
32. Noble CC, Cook DJ (2003) Graph-based anomaly detection. In: SIGKDD
33. Pržulj N, Corneil DG, Jurisica I (2004) Modeling interactome: scale-free or geometric? Bioinformatics 20(18):3508–3515
34. Ralaivola L, Swamidass SJ, Saigo H, Baldi P (2005) Graph kernels for chemical informatics. Neural Netw 18(8):1093–1110
35. Rossi RA, Ahmed NK (2015a) The network data repository with interactive graph analytics and visualization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence
36. Rossi RA, Gallagher B, Neville J, Henderson K (2013) Modeling dynamic behavior in large evolving graphs. In: Proceedings of WSDM, pp 667–676
37. Rossi RA, McDowell LK, Aha DW, Neville J (2012) Transforming graph data for statistical relational learning. J Artif Intell Res 45(1):363–441
38. Rossi R, Ahmed N (2015b) Role discovery in networks. In: TKDE
39. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64
40. Shervashidze N, Petri T, Mehlhorn K, Borgwardt KM, Vishwanathan S (2009) Efficient graphlet kernels for large graph comparison. In: AISTATS

41. Stanley RP (1986) What is enumerative combinatorics?. Springer, Berlin
42. Thomas JJ, Cook KA (2005) Illuminating the path: the research and development agenda for visual analytics. IEEE Computer Society, Washington
43. Traud AL, Mucha PJ, Porter MA (2012) Social structure of facebook networks. Physica A 391(16):4165–4180
44. Ugander J, Backstrom L, Kleinberg J (2013) Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In: WWW
45. Vishwanathan SVN, Schraudolph NN, Kondor R, Borgwardt KM (2010) Graph kernels. JMLR 11:1201–1242
46. Watts D, Strogatz S (1998) Collective dynamics of small-world networks. Nature 393(6684):440–442
47. Wernicke S, Rasche F (2006) Fanmod: a tool for fast network motif detection. Bioinformatics 22(9):1152–1153
48. Zhang L, Han Y, Yang Y, Song M, Yan S, Tian Q (2013) Discovering discriminative graphlets for aerial image categories recognition. IEEE Trans Image Process 22(12):5071–5084
49. Zhang L, Song M, Liu Z, Liu X, Bu J, Chen C (2013) Probabilistic graphlet cut: exploiting spatial structure cue for weakly supervised image segmentation. In: CVPR
50. Zhao B, Sen P, Getoor L (2006) Event classification and relationship labeling in affiliation networks. In: ICML Workshop on Statistical Network Analysis (SNA)

Nesreen K. Ahmed is a Research Scientist at Intel Labs. She received her Ph.D. from the Computer Science Department at Purdue University in 2015, and her M.S. in statistics and computer science from Purdue University in 2014. She was a visiting researcher at Facebook data science, Adobe Labs, Technicolor Labs, and Intel analytics. Her research interests include machine learning, data mining, the theory and algorithms of large-scale graph mining, relational machine learning, and their applications in social and information networks. Dr. Ahmed has authored numerous papers/tutorials in top-tier conferences/journals. Her research was selected among the best papers of ICDM in 2015 and BigMine in 2012 and covered by popular press such as the MIT Technology Review. She was selected by UC Berkeley among the top female rising stars in computer science and engineering in 2014. Dr. Ahmed holds 2 U.S. patents filed by Adobe and co-authored the open source network data repository (http://networkrepository.com).

Jennifer Neville is the Miller Family Chair Associate Professor of Computer Science and Statistics at Purdue University. She received her Ph.D. from the University of Massachusetts Amherst in 2006. In 2016, she was the PC chair of the 9th ACM International Conference on Web Search and Data. In 2012, she was awarded an NSF Career Award; in 2008, she was chosen by IEEE as one of "AI's 10 to watch"; and in 2007, she was selected as a member of the DARPA Computer Science Study Group. Her research focuses on developing data mining and machine learning techniques for relational domains, which include social, information, and physical networks.

Ryan A. Rossi is a research scientist at Palo Alto Research Center (PARC). His research is grounded in machine learning and spans theory, algorithms, and applications of large relational (graph/network) data that arise from social and physical phenomena. Before joining PARC, Dr. Rossi worked at various industrial, government, and academic research laboratories including Lawrence Livermore National Laboratory (LLNL), Naval Research Laboratory (NRL), NASA Jet Propulsion Laboratory (JPL), University of Massachusetts Amherst, among others. Dr. Rossi earned his Ph.D. and M.S. in Computer Science at Purdue University. His dissertation work focused on relational machine learning and mining of dynamic networks. Dr. Rossi is a recipient of the NSF Graduate Research Fellowship (NSF GRFP), National Defense Science and Engineering Graduate Fellowship (NDSEG), as well as two Purdue University Fellowships. He has authored numerous papers at top conferences and journals as well as a number of patents. He also co-founded the network repository project (http://networkrepository.com).

Nick G. Duffield is a Professor in Electrical and Computer Engineering at Texas A&M University, and Director of the Texas A&M Engineering Big Data Initiative. From 1995 until 2013, he worked at AT&T Labs Research, Florham Park, NJ, where he was a Distinguished Member of Technical Staff and an AT&T Fellow. He also held positions in Dublin, Ireland, Heidelberg, Germany, and Rutgers University, NJ. He received a BA (1982) and a MMath (1983) from the University of Cambridge, UK, and a Ph.D. (1987) in Mathematical Physics from the University of London, UK. His research focuses on data science and it applications. He is Chief Editor for Big Data at Frontiers in ICT, an Editor-at-Large for the IEEE/ACM Transactions on Networking, and a member of the board of directors of ACM SIGMETRICS. He is an IEEE and IET Fellow and was a co-recipient of the ACM SIGMETRICS Test of Time Award in 2012 and 2013.

Theodore L. Willke received the EngScD degree with distinction from Columbia University in 2010. His research interests include parallel computing, image processing, machine learning, graph analytics, and cognitive neuroscience. He is the recipient of the IEEE/MILCOM postgraduate fellowship and two US Department of Energy graduate fellowships. He has authored over 25 papers and holds 14 patents. He won the MASCOTS Best Paper Award in 2013 for work on MapReduce performance modeling. He is currently a Senior Principal Engineer with Intel Labs, where his team focuses on parallel computing challenges. He is also a co-principal investigator in a multi-year grand challenge project on real-time brain decoding with the Princeton Neuroscience Institute. Previously, he founded an Intel venture focused on graph analytics for data science that is now an open source project. In 2014, he won Intel's highest award for this effort. In 2015, he was appointed to the Science and Technology Advisory Committee of the US Department of Homeland Security.