Short paper

Efficient Enumeration of Four Node Graphlets at Trillion-Scale

Yudi Santoso
University of Victoria
BC, Canada
santoso@uvic.ca

Venkatesh Srinivasan
University of Victoria
BC, Canada
srinivas@uvic.ca

Alex Thomo
University of Victoria
BC, Canada
thomo@uvic.ca

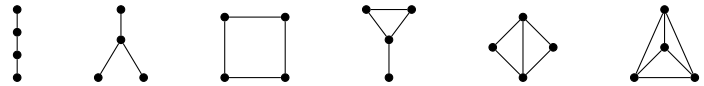

Figure 1: Four node graphlets: a 3-path, a 3-star, a rectangle or 4-cycle, a tailed-triangle, a diamond, and a 4-clique.

ABSTRACT

Graphlet enumeration is known to be a challenging task in graph analysis. This is because the cost is exponential in the order of the graphlet. Triangle is a graphlet of order three that has received special attention because it is relatively small but non-trivial, and can still be enumerated quite fast even for massive graphs of millions of nodes and edges. In this paper, we propose an efficient algorithm for enumerating four node graphlets, such as 4-cycles, 4-cliques, diamonds, etc by leveraging the most efficient algorithm for triangle enumeration. We show that despite the belief that any such enumeration algorithm cannot terminate in reasonable time, our method can handle large graphs containing trillions of such graphlets, using a single commodity machine, within a reasonable amount of time.

1 INTRODUCTION

It is commonly thought that graphlets, beyond three nodes, are difficult to enumerate. This is because the number of possible instances grows as $O(n^k)$, where k is the order of the graphlet and n is the order of the graph. Thus, for massive graphs, it was believed that an enumeration algorithm, which has to touch each graphlet, cannot terminate in a reasonable time [11]. Indeed, previous methods, such as Fanmod [18] and Rage [8], do not scale well and take a very long time to run on million scale graphs. Other proposed solutions, such as Arabesque [17] and PGD [1] use distributed platforms. However, our focus is to explore the limits of what can be achieved using single-machine algorithms.

There are several algorithms proposed in the literature to count the number of the graphlets. They are either estimates using approximation methods, such as Graft [13], or exact counting without full enumeration, notably Orca [6] and Escape [11]. However, what if we need to find each of the graphlet instances? Knowing where the graphlets are is useful in analysing the local structures of the graph. For example, enumerating graphlets is important in detecting cancer through differential graphlet communities [19]. Also, enumeration can yield graphlet degree counts which are useful for uncovering biological network functions [9].

It is worth noting that there have been plenty of studies on triangle enumeration. It was found that triangles can be enumerated quite efficiently using the compact forward edge-iterator algorithm [7]. In general, graphlets of order k can be enumerated using an algorithm with runtime $O(nd^{k-1})$ where d is the maximum degree [15]. However, in [14] it was shown that through careful preprocessing, triangle enumeration using edge-iterator can be significantly faster than $O(nd^2)$ time. Can we achieve a better runtime than $O(nd^{k-1})$, for higher order graphlets?

In this paper we show that efficient enumeration for triangles can be leveraged to enumerate higher order graphlets, in particular four node graphlets. Our algorithm achieves a significantly improved runtime, which depends on the number of three-node graphlets and is able to handle large graphs efficiently on a single machine. Moreover, unlike most in the literature, our solution yields the counts of *all* four node graphlets in a *single* run.

Our contributions are as follows:

(1) We propose a new algorithm to enumerate *all* types of four node graphlets of an undirected graph on a single run. Enumeration is done carefully so that no graphlet is listed more than once.
(2) We provide detailed analyses on the algorithm correctness and time complexity. We refine the time upper-bound of enumeration to depend on the number of three-node graphlets and thus be significantly better than $O(nd^3)$ for real-world networks.
(3) We create an efficient implementation of the algorithm for a single machine. Our algorithm is able to run on graphs of millions nodes and edges, which contain trillions of graphlets, within reasonable time.

2 RELATED WORK

Chiba and Nishizeki published several subgraph listing algorithms [5] which can be considered as a pioneering work on graphlet enumeration. Milo et al. [10] analysed frequent subgraph patterns, and called them network motifs. Since then, there have been many studies on how to find and count small subgraphs within a graph or network, including those we already discussed in the Introduction. Also, Silvestri [16] provided another complexity analysis on subgraph enumeration. To the best of our knowledge, there has not been a solution using the method that we propose here, to simultaneously, and fully, enumerate all the graphlets (of order four) through triangles and wedges, and that can scale to large graphs using a single machine.

3 PRELIMINARIES

In this paper we solely work on simple undirected graphs. We denote a graph by $G(V, E)$ where V is the set of nodes and E is the set of edges. Let $n = |V|$ and $m = |E|$. The degree of a node is the number of edges incident on it. For simple graphs, there is no self-loop and the degree is equal to the number of neighbours. We

Algorithm 1 TRIANGLE ENUMERATION

Input: An undirected graph $G(V, E)$ in an adjacency list representation

1: **for all** vertex $u \in V(G)$ **do**
2: **for all** vertex $v \in N(u)$ **do**
3: **if** $v > u$ **then**
4: **for all** $w \in N(u) \cap N(v)$ **do**
5: **if** $w > v$ **then**
6: EnumerateTriangle (u, v, w)

denote the set of neighbours of node u by $N(u)$, and the degree of u by $d(u) = |N(u)|$. A *subgraph* of $G(V(G), E(G))$ is a graph $H(V(H), E(H))$ such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We use the notation $H \subseteq G$ to say that H is a subgraph of G. A subgraph $H \subseteq G$ is an *induced* subgraph if any edge (u, v) (which = (v, u) for undirected graphs), with $u, v \in V(H)$, is in $E(H)$ if and only if (u, v) is in $E(G)$. A subgraph is *connected* if every pair of nodes in it is connected by a path of edges. We assume the following definition: A *graphlet* is an induced connected subgraph.

There are two kinds of graphlets of three nodes: wedge (Henceforth labeled as g_1) and triangle (g_2). Note that for induced subgraphs, wedges and triangles cannot be on top of each other (i.e., a wedge and a triangle cannot have the same set of three nodes within the same graph.). For four nodes, we have six types of graphlets, depicted in Figure 1. These are 3-path (or four-node-path) (g_3), 3-star (g_4), 4-cycle or rectangle (g_5), tailed-triangle (g_6), diamond (g_7), and 4-clique (g_8). The labels that we use here follow the common labelling used by some papers in the literature [2, 12].

Notice that g_6, g_7 and g_8 contain triangle(s). A g_6 contains one triangle, a g_7 contains two triangles, and a g_8 contains four triangles. This fact suggests that we can find them through the triangles in the graph. Whenever we find a triangle, we can check if this triangle is a part of any g_6, g_7 and/or g_8. Similarly, g_3, g_4 and g_5 contain two, three and four wedges, respectively. Therefore, we can find them through wedges.

We list graphlets by their nodes. Thus, for example, $(u, v, w, z)_8$ is a g_8 with nodes u, v, w and z. In enumeration, some care is needed to avoid multiple listing. Without lost of generality, we can use label 1, 2, 3, 4 to represent the nodes in a graphlet. Clearly, $1 < 2 < 3 < 4$, so 1 represents the smallest node.

There are $3! = 6$ permutations of three nodes. Therefore, for **wedges**, we have $(1, 2, 3)_1$, $(1, 3, 2)_1$, $(2, 1, 3)_1$, $(2, 3, 1)_1$, $(3, 1, 2)_1$ and $(3, 2, 1)_1$. However, $(1, 2, 3)_1$ is the same wedge as $(3, 2, 1)_1$, $(1, 3, 2)_1$ is the same as $(2, 3, 1)_1$, and $(2, 1, 3)_1$ is the same as $(3, 1, 2)_1$. Thus, we have only three possible wedges, only one can be present (for induced case). Our convention is to list with the smaller leg first, i.e. $(1, 2, 3)_1$, $(1, 3, 2)_1$, and $(2, 1, 3)_1$. We can divide these into two types: those with the smallest node at the center of the wedge (type 1), i.e., $(2, 1, 3)_1$, and those with the smallest node at one of the legs (type 2), i.e., $(1, 2, 3)_1$ and $(1, 3, 2)_1$. We will see that they require separate treatment. For **triangles**, all six permutations are isomorphic. Therefore, we only need one to list. We choose the one with the nodes ordered ascendingly: $(1, 2, 3)_2$.

Now for four nodes, there are $4! = 24$ permutations. For **3-paths**, by symmetry we only need half (i.e. twelve) of them. For **3-stars**, we have four distinct ones depending on which one is the centre. For **4-cycles**, the cyclic symmetry gives us a factor of four, while the clockwise counter-clockwise symmetry gives us a factor of two. Therefore, we have only $24/8 = 3$ distinct permutations. For **tailed-triangles**, the distinguishing nodes are the end node and the centre node, giving us $\binom{4}{2}$ or twelve distinct configurations. For **diamonds**, we have a pair of triangles. Let us call the two end nodes of the shared edge as the connecting nodes, and the other two nodes as the opposing nodes. There are symmetries between the two opposing nodes, and between the two connecting nodes, giving us $24/2/2 = 6$ distinct configurations. For **4-cliques**, we can exchange any pair of nodes and get the same clique. Thus there is only one unique configuration, and we choose to list the nodes in order: $(1, 2, 3, 4)_8$.

Algorithm 2 GRAPH-PREP

Input: An undirected graph $G(V, E)$

1: Sort V based on the degrees, in ascending order.
2: Relabel the vertices according to their new order.
3: Build adjacency list of the sorted and relabeled vertices.
4: Cut out the smaller neighbours from each neighbour list.

4 THE ALGORITHMS

The algorithm that we use for triangle enumeration is an edge iterator algorithm (Algorithm 1) combined with nodes ordering. This combination is similar to the Compact-Forward algorithm [7] but with the ordering done in a pre-processing before the enumeration (Algorithm 2).

Algorithm 3 TRIANGLE AND WEDGE ENUMERATION

Input: An undirected graph $G(V, E)$ in an adjacency list representation

1: **for all** vertex $u \in V(G)$ **do**
2: **for all** vertex $v \in N(u)$ **do**
3: **if** $u < v$ **then**
4: **for all** $u' \in N(u)$ and $v' \in N(v)$ **do**
5: **if** $(u' > u) \wedge (v' > u)$ **then**
6: **if** $u' = v' > v$ **then**
7: EnumerateTriangle (u, v, u')
8: **if** $(u' < v') \wedge (u' > v)$ **then**
9: EnumerateWedgeType1 (v, u, u')
10: **if** $u' > v'$ **then**
11: EnumerateWedgeType2 (u, v, v')

The graph preprocessing is based on the following observations: (i) Because of lines 3 and 5 of Algorithm 1 we need to consider only bigger neighbours of every vertex, i.e., $N^{>}(u) = \{v \in N(u) | v > u\}$. (ii) The triangle count in a graph will not change if we relabel the vertices.

Algorithm 1 can be modified to enumerate the wedges as well. This is shown in Algorithm 3. Notice that by condition on line 3 we assure that u is always smaller than v. To avoid multiple listing, when we iterate neighbours of v we consider only those that are bigger than u (line 5). However, we need to include smaller neighbours of v (i.e. those between u and v) to catch all of the wedges.

We extend each of the EnumerateTriangle and EnumerateWedge functions above to search for four-node graphlets. Whenever we find a triangle, $(u, v, w)_2$, we call the ExploreTriangle function (Algorithm 4), which checks for the intersections among the neighbour sets of the three triangle nodes, $N(u)$, $N(v)$ and $N(w)$. If we find a $z \in N(u) \cap N(v) \cap N(w)$, then (u, v, w, z)

Algorithm 4 EXPLORE TRIANGLE

Input: Given triangle $(u, v, w)_2$, $u < v < w$: $N(u)$, $N(v)$, $N(w)$.

Compute intersections among the three neighbour sets.
for all $z \in N(u) \cap N(v) \cap N(w)$ with $z > w$ **do**
    ENUMERATE4CLIQUE $(u, v, w, z)_8$
for all z in two sets and $z >$ opposite node **do**
    ENUMERATEDIAMOND $(.)_7$
for all z in one set only **do**
    ENUMERATETAILEDTRIANGLE $(.)_6$

is a four-clique (i.e. g_8). A node z that is in two of the three neighbour sets gives us a diamond (i.e. g_7), while a z that is in only one of the three neighbour sets gives us a tailed triangle (i.e. g_6). For 4-cliques, we can use the sets of larger neighbours. For diamonds, we can use the sets of neighbours larger than u. For the tailed triangles, however, we need to include all of the neighbours. Due to this last case we lost some of the advantage of the graph preprocessing. As a result, the runtime might be much longer compared to the triangle enumeration time, depending on the maximum degree.

Algorithm 5 EXPLORE WEDGE TYPE-1

Input: Given wedge $(v, u, w)_1$, $u < v < w$: $N^{>u}(u)$, $N^{>u}(v)$, $N^{>u}(w)$.

Compute intersections among the three neighbour sets.
for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ **do**
    ENUMERATERECTANGLE $(u, v, z, w)_5$
for all $z \in N^{>u}(u)$ only **do**
    if $z > w$ **then**
        ENUMERATE3STAR $(u, v, w, z)_4$
for all $z \in N^{>u}(v)$ only **do**
    ENUMERATE3PATH $(w, u, v, z)_3$
for all $z \in N^{>u}(w)$ only **do**
    ENUMERATE3PATH $(v, u, w, z)_3$

For the wedges, we call two different functions depending on the type of the wedge, either Algorithm 5 or 6. In this two functions we only need sets of neighbours that are larger than u, but this is not done in a pre-processing. Notice that in Algorithm 6 w can be smaller than v, which is the center of the wedge.

Algorithm 6 EXPLORE WEDGE TYPE-2

Input: Given wedge $(u, v, w)_1$, $u < v$, $u < w$: $N^{>u}(u)$, $N^{>u}(v)$, $N^{>u}(w)$.

Compute intersections among the three neighbour sets.
for all $z \in N^{>u}(v)$ only **do**
    if $z > w$ **then**
        ENUMERATE3STAR $(v, u, w, z)_4$
for all $z \in N^{>u}(w)$ only **do**
    if $z \neq v$ **then**
        ENUMERATE3PATH $(u, v, w, z)_3$

5 ANALYSIS

THEOREM 1. *Algorithm 3 correctly enumerates wedges and triangles in an undirected graph.*

PROOF. Each edge (u, v), with $u < v$, is iterated once and only once. For each, we enumerate all the intersecting neighbours (i.e., triangles), and non-intersecting neighbours (i.e., wedges). Thus, all wedges and triangles in the graph would be found. For triangles, we avoid multiple listing by imposing condition $u' = v' > v$. For type-1 wedges we impose condition $u' > v$. For type-2 wedges, since $u < v'$ there will be no double counting. □

THEOREM 2. *Algorithms 4, 5 and 6, combined with algorithm 3, correctly enumerate all four node graphlets in an undirected graph.*

PROOF. As proven above, all triangles and wedges are enumerated once. For each triangle, the three neighbour sets are checked. Each node that is in only one of the sets yields a tailed-triangle. All tails would be found in the sets. A node that is in the intersection of two sets yields a diamond. By asserting that this node is larger than the opposite node in the diamond we assure that any diamond would be listed just once. A node that is in the intersection of all three sets yields a 4-clique. We assert that this node is larger than any node in the triangle to assure that the clique has not been listed in any previous iteration. For wedges, similarly, all four node graphlets attached to each wedge would be found. Multiple listing is avoided by considering only 3-paths, 3-stars and 4-cycles, and by careful conditions on the node ordering. For the 3-paths we make sure that the smallest node is always in the first half of the path. For the 3-stars we make sure that the fourth node is greater than the third node. The center node does not need to be the smallest. For 4-cycles we make sure that the fourth node is opposite to the first node. □

THEOREM 3. *The runtime of the four node graphlet enumeration is bounded by $O((N_\Delta + N_\angle)\, d_{\max} + T_{3g})$, where N_Δ ($N_\angle$) is the number of triangles (wedges), and T_{3g} is the time to enumerate triangles and wedges.*

PROOF. For each triangle and wedge the algorithm runs through the neighbor sets to check the intersections with cost $\leq (d(u) + d(v) + d(w))$. □

Note that in general $(N_\Delta + N_\angle) \lesssim nd^2_{\max}$, with the upper value is satisfied by a regular graph. However, for all real world networks, we have $(N_\Delta + N_\angle) \ll nd^2_{\max}$. Also, $T_{3g} \ll nd^2_{\max}$ using efficient enumeration. Therefore, in practice, our runtime is much less than worst case bound of $O(nd^3_{\max})$.

6 EXPERIMENTS

The networks that we study are listed in Table 2. All of the datasets were downloaded from the Laboratory for Web Algorithmics [3, 4], http://law.di.unimi.it/datasets.php. We symmetrized them and got rid of any loops to get simple undirected graphs. We implemented our code in Java with parallel streams, and use Webgraph library [4]. We used a Linux machine with dual Xeon E5-2620 processors of 24 threads and 128 GB of RAM. We notice, however, that the memory usage is < 1 GB throughout the experiment.

The graphlet counts are listed in Table 1. We can check that for all of these graphs, $N_\Delta + N_\angle \ll nd^2_{\max}$ using their $d_{\max}$ values from Table 2. For example, for **amazon**, $N_\Delta + N_\angle \approx$ 42M and $nd^2_{\max} \approx$ 853B, a four order of magnitude difference. For all of the graphs that we consider here the difference is from three to five orders of magnitude.

The runtimes are shown in Table 3. We include the triangle enumeration time, T_Δ, for comparison. As wedges cannot take full advantage of the pre-processing, they take longer time for enumeration, hence T_{3g} is larger than T_Δ. Notice that the preprocessing time, T_{Prep}, is just about the same magnitude as T_Δ.

Table 1: Counts of the graphlets. The dewiki dataset needs longer than our time limit to terminate.

Graph	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8
enron	40,309,453	1,067,993	2,511,039,670	8,043,804,283	21,598,984	582,841,848	46,141,288	5,001,773
cnr	7,798,287,209	20,977,629	6,118,026,632	41,392,015,937,553	37,876,822,234	79,429,334,745	42,974,515,602	159,814,399
dblp	81,529,950	7,005,235	2,678,518,695	3,545,925,764	1,483,611	543,447,587	21,608,538	40,910,658
amazon	38,015,403	4,464,791	372,366,885	609,961,827	2,689,696	9,232,707	13,096,219	4,192,682
dewiki	51,141,107,679	88,611,282	..	..	..	..	..	..
ljournal	8,726,048,197	411,155,444	1,812,284,632,329	8,847,128,736,944	8,551,292,956	189,716,360,703	26,962,410,402	16,129,080,442

Note that T_{4g}, the time required to enumerate all 3 and 4-node graphlets, does not strongly depend on the size of the graph, but rather on the degrees and the numbers of triangles and wedges, validating our analysis. For example, comparing **ljournal** with **amazon**, the ratio of their $(N_\angle + N_\Delta)\, d_{\max}$ values is about four thousand, while the ratio of their T_{4g} values is about six thousand, i.e. approximately the same order. This observation experimentally validates the statement of Theorem 3 relating the runtime to the $(N_\angle + N_\Delta)\, d_{\max}$ value.

Interestingly, **cnr** requires longer runtime than **ljournal**. Even though it is smaller by an order of magnitude it has more graphlets. The **amazon** dataset, which has relatively small maximum degree can be processed in merely 14 seconds. The **dewiki** dataset has enormous number of wedges and large maximum degree and the algorithm did not terminate even after running for four days.

Table 2: The undirected graphs. Here, $d^{\mathrm{BG}}_{\max}$ is the effective maximum degree when only larger neighbours are included after the preprocessing.

Dataset	n	m	$d_{\max}$	$d^{\mathrm{BG}}_{\max}$	d_{avg}
enron	69,244	254,449	1,634	87	7.35
cnr	325,557	2,738,969	18,236	85	16.83
dblp	986,324	3,353,618	979	118	6.80
amazon	735,323	3,523,472	1,077	16	9.58
dewiki	1,532,354	33,093,029	118,246	490	43.19
ljournal	5,363,260	49,514,271	19,432	756	18.46

Table 3: The runtime, in seconds, for preprocessing, for triangle enumeration, for wedges and triangles together, and for all three and four node graphlets together.

Graph	T_{Prep}	T_Δ	T_{3g}	T_{4g}
enron	0.87	1.03	3.49	76.50
cnr	1.93	1.75	57.03	176K
dblp	4.87	1.93	3.45	62.05
amazon	5.80	1.73	2.53	14.05
dewiki	26.45	12.79	517.3	> 300K
ljournal	46.68	32.96	257.1	82K

7 CONCLUSIONS

In this study we have shown that it is possible to enumerate all types of four node graphlets simultaneously with runtime $O((N_\Delta + N_\angle)\, d_{\max} + T_{3g})$. Wedges and triangles can be enumerated relatively fast (in a pre-run) and the result can be used to estimate the time needed to enumerate the four node graphlets. We found that the runtime upper bound depends more on the maximum degree than on the size of the graph. Our algorithm can finish the enumeration in seconds when the maximum degree is around 1K. Moreover, it does not require large memory space, and it would run for even larger graphs (provided that we allow enough time). Notably, we were able to process massive graphs of millions of nodes and edges and enumerate about 40 trillions graphlets in a single run, within a reasonable amount of time.

REFERENCES

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*. IEEE, 1–10.

[2] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*. 91–100.

[3] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World Wide Web*. ACM Press, 587–596.

[4] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM Press, Manhattan, USA, 595–601.

[5] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM Journal on computing* 14, 1 (1985), 210–223.

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

[7] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.* 407, 1-3 (2008), 458–473. https://doi.org/10.1016/j.tcs.2008.07.017

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

[9] Tijana Milenković and Nataša Pržulj. 2008. Uncovering biological network function via graphlet degree signatures. *Cancer informatics* 6 (2008).

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

[11] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1431–1440.

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

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

[14] Yudi Santoso. 2018. *Triangle counting and listing in directed and undirected graphs using single machines*. Master's thesis. University of Victoria.

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

[16] Francesco Silvestri. 2014. Subgraph enumeration in massive graphs. *arXiv preprint arXiv:1402.3444* (2014).

[17] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. 2015. Arabesque: a system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 425–440.

[18] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.

[19] Serene WH Wong, Nick Cercone, and Igor Jurisica. 2015. Comparative network analysis via differential graphlet communities. *Proteomics* 15, 2-3 (2015), 608–617.