Distributed Enumeration of Four Node Graphlets at Quadrillion-Scale

Xiaozhou Liu
University of Victoria
BC, Canada

Yudi Santoso
University of Victoria
BC, Canada

Venkatesh Srinivasan
University of Victoria
BC, Canada

Alex Thomo
University of Victoria
BC, Canada

ABSTRACT

Graphlet enumeration is a basic task in graph analysis with many applications. Thus it is important to be able to perform this task within a reasonable amount of time. However, this objective is challenging when the input graph is very large, with millions of nodes and edges. Known solutions are limited in terms of scalability. Distributed computing is often proposed as a solution to improve scalability. However, it has to be done carefully to reduce the overhead cost and to really benefit from the distributed solution. We study the enumeration of four-node graphlets in undirected graphs using a distributed platform. We propose an efficient distributed solution which significantly surpasses the existing solutions. With this method we are able to process larger graphs that have never been processed before and enumerate quadrillions of graphlets using a modest cluster of machines. We show the scalability of our solution through experimental results. Finally, we also extend our algorithm to enumerate graphlets in probabilistic graphs and demonstrate its suitability for this case.

KEYWORDS

distributed computing, graph mining, massive networks, subgraph enumeration

1 INTRODUCTION

Many problems in network/graph analysis require enumeration and/or counting of small subgraphs. Such problems can be found in various fields: in biology [13, 20] chemistry [8, 26], social study [4, 11], network analysis and classification [36], and more. Here we focus on graphlets, which are defined as small *induced* subgraphs. Furthermore, we are only interested in connected graphlets, hence throughout this paper graphlet is defined as a small induced connected subgraph. Some applications require the enumeration of all graphlets up to a certain degree. For example, Milenkovic and Przulj [20] used 2, 3, 4, and 5 node graphlets to analyse Protein-Protein-Interaction (PPI) networks.

Graphlet enumeration problem is challenging when the size of the input graph is large. In fact, until the recent work [28] (for single machine), it was believed that subgraphs beyond three nodes are difficult to enumerate, and that an enumeration algorithm, which has to touch each subgraph, cannot terminate in a reasonable time [24]. The computational complexity grows exponentially on the order of subgraphs that we want to enumerate. This can be understood combinatorially. Suppose we want to find subgraphs of k nodes in a graph of n nodes, then there are $\binom{n}{k} = n(n-1)\dots(n-k+1)$ possible combinations that we need to check. If $n \gg k$, this is approximately n^k. With a graph of a million nodes, each increment of the subgraph order, k, would cost a million times more in the computation. On the other hand, an order of magnitude increase in n would increase the complexity by 10^k. Of course, in practice, not all combinations need to be checked. Efficient algorithms were built by minimizing unnecessary checking. Nonetheless, it is generally true that the number of subgraphs grows rapidly with the size of the graph.

From this perspective, triangle is a special case. It is small enough to enumerate and yet has important role in graph analysis including computing the clustering coefficient [21] and truss decomposition [34]. The best known solution for triangle enumeration can process a graph of five billion nodes [22]. Four-node subgraph enumeration is already challenging. Known solutions are limited in term of scalability, or the size of the graph that can be processed in a reasonable amount of time. Counting (either exact [12, 24] or approximate [3, 25]) can do more, but we focus on enumeration which is a more challenging problem. Using a distributed platform is an obvious option to increase the computing power, where we can add more compute nodes to get more done within a given time budget. Nevertheless, bringing a single machine solution to a distributed platform has its own challenge. If it is not done properly, we will get a poor scalability which would not justify the economical cost of the distributed platform [19].

Induced subgraphs are more difficult to enumerate than the non-induced ones, because for induced subgraphs we need to check all possible connections as well as non-connections among the nodes. Recently, an efficient algorithm for four-node graphlets enumeration had been proposed in [28]. It has a run time which is much better than $O(nd^{k-1})$, where d is the maximum degree. Moreover, it enumerates all types of four-node graphlets in a single run. This is different from other solutions which do one pattern at a time. Nonetheless, it is designed for a single machine, hence restricting its scalability. This motivates us to search for methods to bring this algorithm onto a distributed platform, such that the solution is optimal for this particular problem.

Furthermore, it is well known that some real world networks are not deterministic. That is, the edges in these networks do not exist with certainty, only with some probabilities. In this case, the graphs are known as probabilistic graphs, or uncertain graphs [10, 16, 33]. This provides us with a motivation to extend our solution to enumerate graphlets in probabilistic graphs.

1.1 Contribution

Our contributions are summarized as follows:

- We devised an efficient distributed algorithm for enumerating *all* induced four-node graphlets on a single run. This includes optimizations that are particular to this problem. We provide detailed analyses on this algorithm to prove its correctness and efficiency.
- We built an implementation of our proposed solution in Spark. Available at https://github.com/D4GE/D4GE.

- We did extensive experimentation with it on several massive graphs, and we compare our code with the state of the art (SotA). The results show that our solution has better performance compared to the SotA. We succeeded in processing a larger graph of a size that had never been processed before in a reasonable amount of time, enumerating quadrillions of graphlets using a modest cluster of machines.
- We also extended our algorithm to enumerate 4-node graphlets in probabilistic graphs. We tested it through experiments on large datasets and demonstrate its effectiveness.

2 RELATED WORK

Subgraph enumeration has received a lot of attention in the literature quite recently. There are many papers on triangle enumeration, including [29] and [14]. On higher order subgraphs, most papers focus on the counting, such as [1, 2, 12], among others. Previous solutions for the induced subgraph enumeration, such as FanMod [35] and Rage [17], do not perform well on million-scale graphs. Cliques, or complete subgraphs, are easier to enumerate, with the latest solution is in [7].

Distributed solutions have been studied extensively in the last two decades. For the triangle enumeration problem, partition-based solution was discussed in [30], where some load-balancing problem was exposed. Park et al. proposed a more efficient solution called PTE (Pre-partitioned Triangle Enumeration) [22]. They later generalized PTE to enumerate query subgraphs of various order, and called their solution PSE (Pre-partitioned Subgraph Enumeration) [23]. It uses VF2 algorithm [6] as its serial enumeration algorithm. We consider PSE as the state of the art for distributed solution of subgraph enumeration problem.

There are several distributed methods/frameworks for graph processing published in the literature, notably Arabesque [32] and its descendant Fractal [9], DistGraph [31], GraphZero [18], G-Miner [5], G-thinker [37], RADS [27] and DISC [38]. They are multipurpose graph applications that can be used to enumerate subgraphs as well. They are query based, where the users need to supply the subgraph finding algorithm. In general, they are more suitable for non-induced subgraphs.

In [28], an algorithm to enumerate all induced four-node (as well as three-node) graphlets in a single run on a single machine has been proposed. We use this algorithm, with some modification, as the serial algorithm for our distributed solution.

Probabilistic graphs have recently gained more attention [10, 16, 33]. In a probabilistic graph, each edge has a probability of existence. The probabilities add more complexity to the graphlet enumeration problem as we have to consider whether to output a graphlet. Considering all possibilities is computationally expensive [16].

To the best of our knowledge, there has not been a distributed solution to simultaneously, and fully, enumerate all the induced 4-node graphlets, both for deterministic and probabilistic cases, that can scale to large graphs using a modest cluster of machines.

3 PRELIMINARIES

3.1 Graphs and Partitions

Graph. A graph, denoted by $G = (V, E)$, is an entity consisting of a set of nodes/vertices, V, and a set of edges among the vertices, E.

Figure 1: Three node graphlets: a wedge and a triangle.

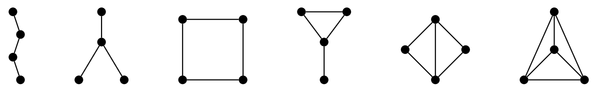

Figure 2: Four node graphlets: a 3-path, a 3-star, a rectangle or 4-cycle, a tailed-triangle or lollipop, a diamond, and a 4-clique.

The number of nodes, $n = |V|$, is the order of the graph, and the number of edges, $m = |E|$, is the size of the graph. Here we assume that the graph is undirected, and without multi-edges or self-loops. The set of neighbouring vertices of vertex u is denoted by $N(u)$. The degree of vertex u is denoted by $d(u) = |N(u)|$.

Induced subgraph. A subgraph $H = (V_H, E_H) \subseteq G = (V_G, E_G)$ is an induced subgraph if for every pair of nodes $u, v \in V_H$, edge $uv \in E_H$ if and only if $uv \in E_G$.

Edge-induced subgraph. An edge-induced subgraph is formed by choosing a set of edges from the graph and include all and only end nodes of those edges.

Note that edge-induced subgraph is not the same as induced subgraph, as induced subgraph is a subgraph of chosen vertices while edge-induced subgraph is a subgraph of chosen edges. As we will see below, induced subgraphs are related to graphlets while edge-induced subgraphs are used for the partitioning.

Graphlets. A graphlet is an induced connected subgraph. There are two types of three node graphlets: wedge and triangle as shown in Fig. 1. There are six types of four node graphlets, shown in Fig. 2. They are 3-path, 3-star, rectangle, lollipop, diamond and 4-clique.

Cliques. A complete graph is a graph with any pair of nodes connected by an edge. In a complete graph of order n (denoted by K_n) the number of edges is therefore $n(n+1)/2$. We may think of cliques as complete sub-graphs. However, these terms are often interchanged in the literature. Thus, a triangle is also a 3-clique, and a K_n is an n-clique.

For distributed computation we partition the graph using a coloring scheme as in [22]. Here are definitions for this partition scheme.

Coloring. Coloring refers to a technique of applying a modulo function with respect to a chosen number of colors, ρ, to each edge $uv \in E$. An edge uv has "color" (i, j) where $i = u\%\rho$, $j = v\%\rho$ and % is the mod operator. Edges with the same color can be grouped together to form an edge-induced sub-graph.

Edge-orientation. Edge-orientation is a technique widely used in sub-graph enumeration because following an orientation helps eliminating duplicate outputs and speeds up the enumeration. It assigns orientation to each edge in an undirected graph by following a prescribed rule. A common rule is as follows. First, define a function η which determines a total ordering of the nodes in V. An edge $uv = vu$ is orientated by η, such that if $\eta(u) < \eta(v)$, we list only uv but not vu. This oriented edge is then denoted by $(\overrightarrow{u,v})$. As is common in practice, we use the degrees of the nodes to define the total ordering η, i.e., if $d(u) < d(v)$ then $\eta(u) < \eta(v)$. If the degrees are equal we just use the node labels to determine the order.

Directed acyclic graph. Using edge-orientation, the undirected input graph is transformed into a directed acyclic graph (DAG), denoted by $\overrightarrow{G}(V, \overrightarrow{E})$. The out-neighbouring vertices of vertex u is denoted as $N^+(u)$. The out-degree of vertex u is denoted as $d^+(u)$.

Edge set. An edge set E_{ij} is an edge-induced sub-graph of the undirected input graph formed by all edges with color (i, j).

Symmetrization. Symmetrization is the process of making all edges in a directed graph bi-directional.

Directed edge set. A directed edge set E^*_{ij} is an edge-induced sub-graph of the edge-oriented DAG, where each edge $(\overrightarrow{u,v})$ of E^*_{ij} points from color i to color j. Directed edge set E^*_{ij} is a subset of edge set E_{ij}. For $i \neq j$, $E^*_{ij} \cup E^*_{ji} = E_{ij}$. For $i = j$, $E^*_{ii} = E_{ii}$.

Sub-problems. A sub-problem refers to the union of edge-sets of particular colors, or more precisely the problem of finding the graphlets in that union-set. For a k-order graphlet enumeration, we denote sub-problems by $S_{\{c_0,c_1,...,c_l\}}$ where $|\{c_0, c_1, ..., c_l\}| \in \{1, 2, ...k\}$ and $c_l \in \{1, 2, ...\rho\}$. For example, for $\rho = 3$ and $k = 3$ (triangle), the sub-problems are: S_0, S_1, S_2, S_{01}, S_{02}, S_{12} and S_{012}, where, $S_i = E_{ii}$, $S_{ij} = E_{ii} \cup E_{ij} \cup E_{jj}$, and $S_{ijk} = E_{ij} \cup E_{ik} \cup E_{jk}$. Note that $S_i \subset S_{ij}$, but $S_{ij} \not\subset S_{ijk}$.

3.2 Graphlets Enumeration

Compact-Forward [15] is the *de-facto* serial algorithm for triangle enumeration. For a given order-by-degree input DAG $\overrightarrow{G}(V, \overrightarrow{E})$, Compact-Forward enumerates all triangles with $O(|\overrightarrow{E}|^{3/2})$ operations. A single machine algorithm for enumerating all six types of 4-node graphlets in a single run was introduced in [28], and we call this algorithm S4GE (Simultaneous 4-node Graphlets Enumeration)[1]. It searches four node graphlets through triangles and wedges. That is, it first searches for triangles and wedges, and for each triangle that it finds it looks around it to see if this triangle is a part of any tailed triangles, diamonds and/or 4-cliques. Similarly, through wedges it searches for 3-paths, 3-stars and rectangles. These are done by intersecting the neighbour sets of the three vertices. The details can be found in [28]. S4GE has a running time $O(T_{3g} + (|\Delta| + |\angle|)d)$, where T_{3g} is the time to enumerate wedges and triangles.

3.3 Previous Distributed Enumeration

Park et al. [22] proposed a distributed solution for triangle enumeration called PTE (Pre-partitioned Triangle Enumeration). PTE employs Compact-Forward algorithm as the local serial algorithm. On each distributed worker, PTE does $O(m^{1.5}/\rho^3)$ amount of work. Summing over all $O(\rho^3)$ sub-problems, PTE recovers $O(m^{1.5})$ amount of work overall, which is the same asymptotic behaviour as the Compact-Forward on a single machine. Note however, that because of the distribution of the subproblems there are inherently some redundant computations. Park et al. was able to reduce the total number of operations by a factor of $2 - \frac{2}{\rho}$ by employing color directions to minimize this redundancy.

Park et al. generalised PTE to support non-induced sub-graph query of an arbitrary order, called PSE (Pre-partitioned Subgraph Enumeration) [23]. PSE takes a query sub-graph $G_q(V_q, E_q)$ of order k as input where $k = |V_q|$, and enumerates all the matching sub-graphs to G_q. PSE employs VF2 algorithm [6] as the local serial algorithm for query graph matching. We stress that VF2 can only do one non-induced subgraph query at a time, in contrast to S4GE which enumerates all types of four-node induced connected subgraphs simultaneously. PSE starts by defining $\sum_{l=1}^{k} \binom{\rho}{l}$ sub-problems. For example, with $\rho = 4$ and $k = 4$, PSE first defines the following sub-problems: S_0, S_1, S_2, S_3, S_{01}, S_{02}, S_{03}, S_{12}, S_{13}, S_{23}, S_{012}, S_{013}, S_{023}, S_{123} and S_{0123}. Park et al. observed that solving the sub-problems independently introduces duplicate emissions, and that some sub-problems can be grouped together to reduce the duplication. For example, since $S_0 \subset S_{01} \subset S_{012}$, enumerating the sub-graphs from S_{012} also enumerates all the sub-graphs from S_0 and S_{01}. PSE introduced sub-problem group as the fundamental computing task on each distributed worker. For example, $\{S_{012}, S_0, S_1, S_2\}$, is a valid sub-problem group, where solving S_{012} is sufficient to solve for the entire group. Park et al. showed that PSE requires at most $\binom{\rho-1}{k-2}|E|$ amount of network read, for querying k-order sub-graphs from an input graph of size $|E|$.

4 PROPOSED METHODS

4.1 Generalized Color-Direction

PSE, while correctly enumerating all sub-graphs that match the query, discovers certain sub-graphs more than once. Consider the following: if there is a 4-node sub-graph (u, v, w, z) whose color is $(0, 0, 1, 1)$, it can be discovered from the group $\{S_{012}, S_0, S_1, S_2\}$ as well as the group $\{S_{013}, S_{01}, S_{03}\}$, since the first group S_{012} reads edge sets $E_{00} \cup E_{11} \cup E_{22} \cup E_{01} \cup E_{02} \cup E_{12}$, and the second group S_{013} reads edge sets $E_{00} \cup E_{11} \cup E_{33} \cup E_{01} \cup E_{03} \cup E_{13}$. Both contains $E_{00} \cup E_{01} \cup E_{11}$ where (u, v, w, z) of color $(0, 0, 1, 1)$ would be found.

We observe that: (1) Given a 4-node graphlet and ρ colors, there are in total ρ^4 possible color assignments of the four vertices (denoted by K_{ijkl}). (2) When a 4-node graphlet (u, v, w, z) is emitted, it is imposed that the graphlet edges are oriented following the ordering of the vertices: $(\overrightarrow{u,v})$, $(\overrightarrow{u,w})$, $(\overrightarrow{u,z})$, $(\overrightarrow{v,w})$, $(\overrightarrow{v,z})$ and $(\overrightarrow{w,z})$. Combining both observations, each color assignment K_{ijkl} can be used to represent the set of all possible 4-node graphlets (u, v, w, z) of ordered colors (i, j, k, l), where the edges can only point from color i to colors $\{j, k, l\}$, from color j to colors $\{k, l\}$, and from color k to color l. Any 4-node graphlet can have only one unique ordered color assignment. Each ordered color assignment contains all the 4-node graphlets that meets the criteria, and there is no overlap among different ordered color assignments. Hence, enumerating from all ordered color assignment enumerates all the 4-node graphlets once and once only. The ordered color assignment can be viewed as a directed version of a sub-problem. Unlike a sub-problem S_{ijkl} that requires the union of edge sets E_{ij}'s, a color assignment K_{ijkl} requires the union of directed edge sets E^*_{ij}'s. A color assignment K_{ijkl} requires knowledge of $E^*_{ij} \cup E^*_{ik} \cup E^*_{il} \cup E^*_{jk} \cup E^*_{jl} \cup E^*_{kl}$. However, simply solving all the ordered color assignments on distributed workers will incur unnecessary network read. The color assignments therefore are grouped into sub-problems to reduce the network read, with a strategy introduced below. Sub-problems become the fundamental task assigned to each distributed worker.

[1]This algorithm was not named in the original paper.

Algorithm 1 D4GE

```
Input: An undirected graph $G(V, E)$; the number of colors $\rho$
1: Construct $\overrightarrow{G}(V, \overrightarrow{E})$ by applying edge-orientation to $G(V, E)$
2: Symmetrise $\overrightarrow{G}(V, \overrightarrow{E})$ into $G^{\mathrm{sym}}(V, E^{\mathrm{sym}})$
3: Partition $E^{\mathrm{sym}}$ into directed edge sets $E^*_{ij}$ using $\rho$
4: Generate ordered color assignments and sub-problems $\{S_{\mathrm{Cs}} \mapsto \{K_{ijkl}\}\}$
5: for all $S_{\mathrm{Cs}}$, $\{K_{ijkl}\}$ do                ▷ Distributed-for
6:     $E_{\mathrm{map}} \leftarrow$ ReadEdgeSets$_{\mathrm{CD}}$ $(S_{\mathrm{Cs}}, 4)$
7:     for all $K_{ijkl} \in \{C_0, C_1, ...\}$ do
8:         S4GE$_{\mathrm{CD}}$ $(E_{\mathrm{map}}, ijkl)$
```

Algorithm 2 ReadEdgeSets$_{\mathrm{CD}}$

```
Input: Sub-problem $S_{\mathrm{Cs}}$ with $Cs = \{c_0, c_1, ..., c_l\}$; order of query graph $k$
1: Initialize empty map $E_{\mathrm{map}} \equiv \{(i, j) \mapsto E^*_{ij}\}$
2: for all $(i, j) \in Cs^2$ do
3:     if $i = j$ and $|\{c_0, c_1, ..., c_l\}| \neq k$ then
4:         $E_{\mathrm{map}}[(i, i)] \leftarrow E^*_{ii}$
5:     else
6:         $E_{\mathrm{map}}[(i, j)] \leftarrow E^*_{ij}$
   return $E_{\mathrm{map}}$
```

We call our scheme, applied to four-node graphlets, as Distributed 4-node Graphlet Enumeration (D4GE). The pseudo code of D4GE is given as Algorithm 1 and Algorithm 2. We want to stress that while PTE employs the idea of color-direction to reduce the amount of work performed, we exploit both the linearity of the DAG and the color-direction, and are able to observe that the color-assignment problem is essentially a combination problem, and the unique relationship between any subgraph to its color assignment guarantees the duplication-freeness of our algorithm. In addition, PTE explicitly lists all the ordered color-tuples in the algorithm, while we use combinations to generalize color-assignment. This works not only for $k = 4$, but also to any order k (with ρ^k color-assignments).

D4GE takes a DAG as input and symmetrises it. Symmetrization is necessary to ensure the correctness of S4GE to enumerate all the wedge-based 4-node graphlets. Consider the graph in Fig. 3 with DAG adjacency list: 1: {4}, 2: {4}, 3: {4}, 4: {5,6}, 5: {6}, 6: ∅. If we apply S4GE algorithm on this adjacency list, we will only find triangle (4, 5, 6) but will not discover tailed-triangle (1,4,5,6), (2,4,5,6) and (3,4,5,6), as vertices 1, 2, 3 are not in the adjacency list of of vertex 4. With symmetrization, the adjacency list is now 1: {4}, 2: {4}, 3: {4}, 4: {1,2,3,5,6}, 5: {4,6}, 6: {4, 5}, and S4GE can now successfully enumerate the three tailed-triangles, as vertices 1, 2 and 3 are added into the neighbourhood of vertex 4.

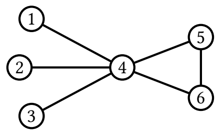

Figure 3: A graph illustrating the need for symmetrization.

Now, we introduce the grouping strategy to form sub-problems from ordered color assignments. Consider color assignments K_{0001} and K_{0002}. By definition K_{0001} requires knowledge of $E^*_{00} \cup E^*_{01}$ and K_{0002} requires knowledge of $E^*_{00} \cup E^*_{02}$. If these two color assignments are computed on two different workers, the partitioned edge-set E^*_{00} is then loaded twice. To address this, color assignments K_{pqrs} are grouped into sub-problems. We use S_{ijkl} to denote a sub-problem. The color assignments are grouped by the following rule: K_{pqrs} belongs to sub-problem S_{ijkl} if the *sorted* and *reduced* form of $\{p, q, r, s\}$ is $\{i, j, k, l\}$, where *sorted* means $\{p, q, r, s\}$ is sorted in ascending order, and *reduced* means removing the duplicated colors from the sequence $\{p, q, r, s\}$. For example, the sorted and reduced form of $\{2, 0, 1, 0\}$ is $\{0, 1, 2\}$. Therefore K_{2010} belongs to S_{012}.

It is not hard to see that sub-problem S_{ijkl} contains $4! = 24$ color assignments - precisely the number of permutations of the sequence $\{i, j, k, l\}$. To fully cover all the color assignments, D4GE generates $\binom{\rho}{2}$ number of S_{ij}, $\binom{\rho}{3}$ number of S_{ijk} and $\binom{\rho}{4}$ number of S_{ijkl}. Sub-problem S_{ijkl} contains all the ordered color assignments K_{pqrs} where $p, q, r, s \in \{i, j, k, l\}$; sub-problem S_{ijk} contains all the ordered color assignments K_{pqrs} where $p, q, r, s \in \{i, j, k\}$; sub-problem S_{ij} contains all the ordered color assignments K_{pqrs} where $p, q, r, s \in \{i, j\}$. In the special case of ordered color assignments K_{iiii} where all four colors are the same, we omitted S_i and instead attach K_{iiii} to sub-problem S_{ij} where $i+1 = j\%\rho$. Each sub-problem is computed independently on a distributed worker.

For all ordered color assignments under sub-problem S_{ijkl}, there are only two possible relative orders of two arbitrary colors p and q: p precedes q or the reverse, meaning for any p and q from $\{i, j, k, l\}$, $E^*_{pq} \cup E^*_{qp} = E_{pq}$ is needed. Hence overall, to fully enumerate S_{ijkl}, $E_{ij} \cup E_{ik} \cup E_{il} \cup E_{jk} \cup E_{jl} \cup E_{kl}$ needs to be read. Sub-problem S_{ijk} can be treated as $S_{iijk} \cup S_{ijjk} \cup S_{ijkk}$ to reflect that it requires $E_{ii} \cup E_{ij} \cup E_{ik} \cup E_{jj} \cup E_{jk} \cup E_{kk}$, and sub-problem S_{ij} can be treated as $S_{iiij} \cup S_{iijj} \cup S_{ijjj}$ to reflect that it requires $E_{ii} \cup E_{ij} \cup E_{jj}$. Each of the sub-problems and the associated ordered color assignments are sent to a distributed worker; the worker iterates all the ordered color assignments. For each colored assignment the worker reads the directed edge sets from distributed storage, and enumerates the 4-node graphlets by applying the a modified S4GE algorithm.

4.2 S4GE$_{\mathrm{CD}}$

S4GE is modified accordingly so that it is able to enumerate all 4-node graphlets for an ordered color assignment $ijkl$, and we call this modified version S4GE$_{\mathrm{CD}}$. The pseudocode for S4GE$_{\mathrm{CD}}$ is given in Algorithm 3, with the details of the explore-functions are given in Algorithms 4, 5, and 6 respectively.

Algorithm 3 S4GE$_{\text{CD}}$

Input: A mapping from the colors of directed edge set to the edge set $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; ordered color assignment $ijkl$

```
1: $E^*_{ij} \equiv E_{\text{map}}[ij]$, $E^*_{ik} \equiv E_{\text{map}}[ik]$, $E^*_{jk} \equiv E_{\text{map}}[jk]$
2: for all $(u,v) \in E^*_{ij}$ do
3:     if $\eta(u) < \eta(v)$ then
4:         for $u' \in N(u) \subset E^*_{ik}$ and $v' \in N(v) \subset E^*_{jk}$ do
5:             if $(u' > u) \wedge (v' > u)$ then
6:                 if $u' = v' > v$ then
7:                     ExploreTriangle $(u, v, u', E_{\text{map}}, ijkl)$
8:                 if $(u' < v') \wedge (u' > v)$ then
9:                     ExploreWedge-1 $(v, u, u', E_{\text{map}}, ijkl)$
10:                if $u' > v'$ then
11:                    ExploreWedge-2 $(u, v, v', E_{\text{map}}, ijkl)$
```

Algorithm 4 ExploreTriangle

Input: Given triangle (v,u,w); $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; ordered color assignment $ijkl$.

```
1: $N^{>u}(u) \equiv \{z | z \in N(u)|_{E_{\text{map}}[il]}, \eta(z) > \eta(u)\}$
2: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E_{\text{map}}[jl]}, \eta(z) > \eta(u)\}$
3: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E_{\text{map}}[kl]}, \eta(z) > \eta(u)\}$
4: for all $z \in N^{>u}(u \cap N^{>u}(v) \cap N^{>u}(w)$ with $z > w$ do
5:     Enumerate4Clique $(u, v, w, z)$
6: for all $z$ in two sets and $z >$ opposite node do
7:     EnumerateDiamond $(u, v, w, z)$
8: for all $z$ in one set only do
9:     EnumerateTailedTriangle $(u, v, w, z)$
```

Instead of enumerating on a complete graph, S4GE$_{\text{CD}}$ now enumerates on a sub-graph denoted by the color assignment $ijkl$. The sub-graph is consisted of a mapping between the ordered color 2-tuples (i,j) and the corresponding directed edge-sets E^*_{ij}. For an ordered color assignment, there are $\binom{4}{2} = 6$ such 2-tuples: (i,j), (i,k), (i,l), (j,k), (j,l) and (k,l). (i,j), (i,k), (j,k) and the corresponding edge-sets are used to discover the wedge or triangle, and (i,l), (j,l), (k,l) and the corresponding edge-sets are used to discover the graphlet after the base wedge or triangle have been discovered. S4GE$_{\text{CD}}$ inherits the correctness from S4GE since the actual intersection logic is untouched, whereas S4GE$_{\text{CD}}$ solely focuses on a particular edge-induced sub-set of the input graph, with all the edges pointing from color i to j, k, l, from j to k, l and from k to l.

The modification of S4GE shows the expandability of D4GE partitioning scheme. Since the intersection is not modified, the partitioning scheme can be applied to different edge-based enumeration algorithm to suit different needs. All it requires is to modify the input to accommodate a directed sub-set of the input graph.

4.3 Compact-Forward for 4-clique listing

Since PSE with VF2 only supports one query graph per run, for the purpose of comparison we build an algorithm to enumerate 4-cliques. Furthermore, we fit it to be used with the color direction scheme of D4GE. This algorithm, called CF4$_{\text{CD}}$, is given as Alg. 7. Note that CF4 extends the idea of Compact-Forward algorithm from triangles to four-cliques, hence the name CF4. The correctness of CF4$_{\text{CD}}$ is intuitive. CF4$_{\text{CD}}$ does $O(m^2)$ work for a given graph.

Algorithm 5 ExploreWedge-1

Input: Given wedge (v,u,w); $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; ordered color assignment $ijkl$.

```
1: $N^{>u}(u) \equiv \{z | z \in N(u)|_{E_{\text{map}}[il]}, \eta(z) > \eta(u)\}$
2: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E_{\text{map}}[jl]}, \eta(z) > \eta(u)\}$
3: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E_{\text{map}}[kl]}, \eta(z) > \eta(u)\}$
4: for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ do
5:     EnumerateRectangle $(u, v, z, w)$
6: for all $z \in N^{>u}(u)$ only do
7:     if $z > w$ then
8:         Enumerate3Star $(u, v, w, z)$
9: for all $z \in N^{>u}(v)$ only do
10:    Enumerate3Path $(w, u, v, z)$
11: for all $z \in N^{>u}(w)$ only do
12:    Enumerate3Path $(v, u, w, z)$
```

Algorithm 6 ExploreWedge-2

Input: Given wedge (v,u,w); $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; ordered color assignment $ijkl$.

```
1: $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
2: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
3: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
4: for all $z \in N^{>u}(v)$ only do
5:     if $z > w$ then
6:         Enumerate3Star $(v, u, w, z)$
7: for all $z \in N^{>u}(w)$ only do
8:     if $z \neq v$ then
9:         Enumerate3Path $(u, v, w, z)$
```

Algorithm 7 CF4$_{\text{CD}}$

Input: An edge-oriented edge set E^*_{ij}, E^*_{ik}, E^*_{jk}, E^*_{il}, E^*_{jl}, E^*_{kl}

```
1: for all $(\overrightarrow{u,v}) \in E^*_{ij}$ do
2:     for all $w \in \{N^+(u)|_{E^*_{ik}} \cap N^+(v)|_{E^*_{jk}}\}$ do
3:         for all $z \in \{N^+(u)|_{E^*_{il}} \cap N^+(v)|_{E^*_{jl}} \cap N^+(w)|_{E^*_{kl}}\}$ do
4:             Enumerate $(u, v, w, z)$
```

4.4 Analysis

In this analysis , first we show that D4GE with S4GE$_{\text{CD}}$ correctly enumerates all the 4-node graphlets. D4GE works by generating all possible colored assignments of all 4-node graphlets. Any 4-node graphlet must be found from one and only one of the colored assignments. D4GE then applies S4GE$_{\text{CD}}$ algorithm on each individual color assignment. Since S4GE correctly enumerates all 4-node graphlets for any given graph, D4GE/S4GE$_{\text{CD}}$ correctly enumerates all 4-node graphlets for all color assignments of $G^{\text{sym}}(V, E^{\text{sym}})$.

Second, we show that D4GE with S4GE$_{\text{CD}}$ is expected to require no more than $2\,m^{\text{sym}}$ amount of network read in addition to PSE, where m^{sym} is the number of edges in E^{sym}. For D4GE with S4GE$_{\text{CD}}$, the edge set E_{ii} is requested $(\rho-1)$ times (by sub-problems S_{ik}), and $\binom{\rho-1}{2}$ times (by sub-problems S_{ikl}), hence the amount of network read is $\sum_{i=0}^{\rho-1}|E_{ii}|\binom{\rho}{2}$. The E_{ij} with $i \neq j$ is requested once by sub-problems S_{ij}, $\binom{\rho-2}{1}$ times by sub-problems S_{ijk}, and $\binom{\rho-2}{2}$ times by sub-problems S_{ijkl}. Thus, the amount of network read is $\sum_{i=0}^{\rho-1}\sum_{j=i+1}^{\rho-1}|E_{ij}|\left[1+\binom{\rho-1}{2}\right]$. Combining both cases:

$$\begin{aligned}
&\sum_{i=0}^{\rho-1}|E_{ii}|\binom{\rho}{2}+\sum_{i=0}^{\rho-1}\sum_{j=i+1}^{\rho-1}|E_{ij}|\left[1+\binom{\rho-1}{2}\right]\\
&=\binom{\rho}{2}\left[\sum_{i=0}^{\rho-1}|E_{ii}|+\sum_{i=0}^{\rho-1}\sum_{j=i+1}^{\rho-1}|E_{ij}|\right]-(\rho-2)\sum_{i=0}^{\rho-1}\sum_{j=i+1}^{\rho-1}|E_{ij}|\\
&\equiv\binom{\rho}{2}m^{\text{sym}}-(\rho-2)m^{\text{sym}}_{\neq}
\end{aligned}\tag{1}$$

If we assume the edges are distributed evenly, the expected size of $m^{\text{sym}}_{\neq}$ is $\frac{\rho^2-\rho}{\rho^2}=1-\frac{1}{\rho}$ of m^{sym}. Recall that, for $k=4$, PSE requires $\binom{\rho-1}{2}\,m^{\text{sym}}$ amount of network read. Thus the difference to PSE is

$$\left[\binom{\rho}{2}-\rho+3-\frac{2}{\rho}\right]m^{\text{sym}}-\binom{\rho-1}{2}m^{\text{sym}}=\left(2-\frac{2}{\rho}\right)m^{\text{sym}}\tag{2}$$

which is less than $2\,m^{\text{sym}}$.

Last, we show D4GE reduces the amount of work compared to PSE. For this comparison we are using S4GE$_{\text{CD}}$ as the localized algorithm. Since S4GE$_{\text{CD}}$ enumerates all 4-node graphlets by discovering the base triangle and wedges first, we separate the work calculation into two parts: one being the amount of work to discover all the base triangle and wedges, the other to discover the fourth vertex.

Let us consider the first part. We can see that $\sum_{(u,v)\in E^{\text{sym}}}(d^{\text{sym}}(u)+d^{\text{sym}}(v))$ is the amount of work to intersect all pairs of edges for a symmetrised graph. This sum is bounded by and can be estimated by $2\,m^{\text{sym}}d^{\text{sym}}_{\max}$, where $d^{\text{sym}}_{\max}$ is the maximum degree of the symmetrised graph. Following the analysis to derive expression 1, D4GE does

$$2\binom{\rho}{2}m^{\text{sym}}_{=}d^{\text{sym}}_{\max}+2[1+\binom{\rho-1}{2}]m^{\text{sym}}_{\neq}d^{\text{sym}}_{\max}\tag{3}$$

amount of work for discovering all the base triangles and wedges. For PSE, each E^*_{ii} is read $\binom{\rho-1}{2}$ times; each E^*_{ij} with $i\neq j$ is read $\binom{\rho-2}{1}+\binom{\rho-2}{2}=\binom{\rho-1}{2}$ times. So the total amount of work done by PSE to list all base triangles and wedges is

$$2\binom{\rho-1}{2}m^{\text{sym}}_{=}d^{\text{sym}}_{\max}+2\binom{\rho-1}{2}m^{\text{sym}}_{\neq}d^{\text{sym}}_{\max}.\tag{4}$$

Subtracting Expression 3 by Expression 4 yields

$$\begin{aligned}
&2\binom{\rho-1}{1}m^{\text{sym}}_{=}d^{\text{sym}}_{\max}+2m^{\text{sym}}_{\neq}d^{\text{sym}}_{\max}\\
&=2(\rho-2)\,m^{\text{sym}}_{=}d^{\text{sym}}_{\max}+2m^{\text{sym}}d^{\text{sym}}_{\max}
\end{aligned}\tag{5}$$

recall that if we assume edges are distributed evenly, the expected value of $m^{\text{sym}}_{=}$ is $\frac{1}{\rho}m^{\text{sym}}$. Thus expression 5 can be simplified to

$$\begin{aligned}
&2(\rho-2)\,m^{\text{sym}}_{=}d^{\text{sym}}_{\max}+2m^{\text{sym}}d^{\text{sym}}_{\max}\\
&=2\frac{\rho-2}{\rho}m^{\text{sym}}d^{\text{sym}}_{\max}+2m^{\text{sym}}d^{\text{sym}}_{\max}\\
&=\left(4-\frac{4}{\rho}\right)m^{\text{sym}}d^{\text{sym}}_{\max}
\end{aligned}\tag{6}$$

Now consider the second part - the work required to locate the fourth vertex after listing all the base shapes. Given a particular base triangle or wedge (u,v,w), the amount of work by S4GE$_{\text{CD}}$ to locate the 4th vertex z through intersection is $d^{\text{sym}}(u)+d^{\text{sym}}(v)+d^{\text{sym}}(w)$. Similarly, this expression is upper bounded by $3\,d^{\text{sym}}_{\max}$. Also for each graph dataset, the numbers of triangles and wedges are fixed. Denote the uni-color triangles and wedges as Δ_I and $\angle_I$, the bi-color ones as Δ_{II} and $\angle_{II}$, and the tri-color ones as Δ_{III} and $\angle_{III}$. For PSE with S4GE, each Δ_I and $\angle_I$ is emitted $\binom{\rho-1}{2}$ times from E_{ii}; each Δ_{II} and $\angle_{II}$ is emitted $\binom{\rho-2}{1}$ times from $E_{ii}\cup E_{ij}$; each Δ_{III} and $\angle_{III}$ is emitted $1+\binom{\rho-3}{1}$ from $E_{ij}\cup E_{jk}\cup E_{ik}$. Hence the total work for locating the 4th vertex z, using PSE partitioning scheme and S4GE serial algorithm is:

$$\begin{aligned}
&3\binom{\rho-1}{2}d^{\text{sym}}_{\max}(|\Delta_I|+|\angle_I|)+3\binom{\rho-2}{1}d^{\text{sym}}_{\max}(|\Delta_{II}|+|\angle_{II}|)\\
&+3\left(1+\binom{\rho-3}{1}\right)d^{\text{sym}}_{\max}(|\Delta_{III}|+|\angle_{III}|)
\end{aligned}\tag{7}$$

D4GE/S4GE$_{\text{CD}}$ enumerates each triangle and wedges ρ times. This is required because given a triangle or wedge of color (i,j,k), the 4th vertex can have ρ different colors. All ρ colors are necessary to ensure that all graphlets would be enumerated. Thus overall, D4GE with S4GE$_{\text{CD}}$ does

$$3\,\rho\,d^{\text{sym}}_{\max}(|\Delta|+|\angle|)\tag{8}$$

amount of work. Expression 7 minus expression 8 yields

$$\begin{aligned}
&3\left(\binom{\rho-1}{2}-\rho\right)d^{\text{sym}}_{\max}(|\Delta_I|+|\angle_I|)\\
&-6\,d^{\text{sym}}_{\max}(|\Delta_{II}|+|\angle_{II}|+|\Delta_{III}|+|\angle_{III}|)
\end{aligned}\tag{9}$$

which is the amount of extra work PSE/S4GE does compared to D4GE/S4GE$_{\text{CD}}$ for enumerating all the 4-node graphlets, after all the wedges and triangles are discovered.

Now consider expressions 6 and 9. Expression 6 shows that the extra work performed by D4GE with S4GE$_{\text{CD}}$ to discover all base triangles and wedges is sensitive to the size of the symmetrised graph, *ie.*, the number of edges and degrees; expression 9 shows that the extra work performed by PSE with S4GE to list the 4th vertex, has a quadratic growth with respect to ρ, and is sensitive to the number of uni-color triangles and wedges. Note that for real-world graphs, the number of wedges plus triangles is often a magnitude greater than the number of the edges, and for a reasonable-sized cluster, ρ is often set to a large value. As a result, D4GE with S4GE$_{\text{CD}}$ can often achieve greater performance improvement. This will be confirmed in the experiments below.

Table 1: The numbers of vertices n, edges m, wedges $|\angle|$, and triangles $|\Delta|$, and the max degree of the symmetrized graphs.

Dataset	n	m	$\|\angle\|$	$\|\Delta\|$	d^{sym}_{max}
enron	69K	510K	40M	1M	1.6K
cnr	326K	5.6M	7.8B	21M	18K
amazon	735K	7M	38M	4.5M	1.1K
hollywood09	1.1M	114M	33B	4.9B	11K
dewiki	1.5M	33M	51B	89M	118K
hollywood11	2.2M	229M	100B	7.1B	13K
orkut	3M	234M	44B	628M	33K
ljournal	5.4M	100M	8.7B	441M	19K
uk02	18.5M	529M	188B	4.5B	195K
enwiki18	5.6M	235M	297B	378M	248K
indochina	7.4M	304M	392B	61B	256K

5 EXPERIMENTS

Unless specifically stated otherwise, the experiments were conducted using 30 Intel E5430 quad-core machines with 6 GB of RAM each. This gives equivalently 120 distributed workers[2] and 1.5 GB of RAM per worker. D4GE is implemented in Apache Spark 2.4.5 with OpenJDK 1.8.0. The graph datasets were collected from http://law.di.unimi.it/datasets.php in WebGraph format, and symmetrized. The statistics of the symmetrized graphs are listed in Table 1.

5.1 The impact of ρ on performance

We first address ρ's impact on the overall performance of our distributed algorithm. In a previous literature, Suri and Vassilvitskii [30] regarded ρ as a trade off between the network read and the input size of each distributed worker: a larger value of ρ increases the amount of network read, but also decreases the input size as each task becomes smaller. Park et al [22] on the other hand adjusted ρ accordingly to the input graph size, to fully utilize the amount of available memory for each worker.

We show that while ρ affects the amount of network read, a large ρ value in practice can help with balancing the workload distribution, even when the number of sub-problems over-saturates the number of workers. Also, with a large enough ρ, the input size of each task shall never exceed the allocated memory for each worker. We experimented with D4GE/S4GE$_{CD}$ on three different datasets and varying $\rho = 8, 12, 16$ and 20. The result is shown in Fig. 4.

When $\rho = 8$ there are $\binom{8}{2} + \binom{8}{3} + \binom{8}{4} = 154$ sub-problems, hence $\rho = 8$ is the minimum value to saturate our cluster of 120 workers. Any ρ greater than 8 will over-saturate the cluster. Theoretically, we should not see any improvement after $\rho = 8$, but in fact we do. This is because of better load balancing. From $\rho = 8$ to 12, we observe improvement, consistently on various datasets. This shows that $\rho = 12$, with almost 5 times more sub-problems than $\rho = 8$, gives us a better workload distribution. However, the improvement diminishes and the performance would eventually decrease as ρ gets higher. The overhead of network read and Apache Spark framework itself could

[2]Each worker is equivalent to a physical CPU core.

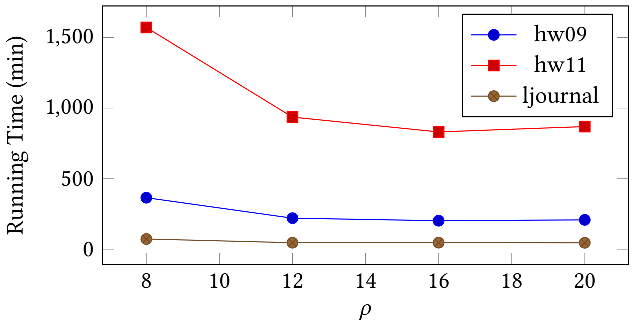

Figure 4: The enumeration time (minutes) of D4GE/S4GE$_{CD}$ on several graphs, with varying value of ρ. Higher ρ do not add much overhead; the lines flatten out rather than sloping up perceptibly.

dwarf the computation when ρ is too large. We would like to note that the network read in our experiment is through internal traffic - i.e., traffic between distributed workers and distributed storage. Internal traffic is often free even on a commercial platform, and the internal network connection can be order of magnitude faster than an external one. Even though a large ρ value introduces more network read, the performance penalty from the network read is negligible.

5.2 Machine scalability

We investigate the machine scalability of D4GE/S4GE by measuring the running time on **hollywood09** and **cnr** dataset while varying the number of distributed workers from 32 to 256. The results are presented in Figure 5. D4GE/S4GE shows strong scalability: with slope -0.968 and -0.899 respectively, which are very close to the perfect value -1. It means that the running time decreases by $2^{-0.968} = 1.956$ and $2^{0.899} = 1.865$ times, respectively, when the number of machines is doubled. We emphasize that this is on-par with the SotA Map-Reduce based algorithms [22] and [23].

5.3 PSE/S4GE vs D4GE/S4GE$_{CD}$

Next, we modified PSE and replaced VF2 with S4GE in the PSE. We then compared D4GE/S4GE$_{CD}$ against PSE/S4GE. For this experiment we set $\rho = 16$. The enumeration times are listed in Table 2. We found that D4GE/S4GE$_{CD}$ is more efficient for all of the tested graphs, and D4GE/S4GE$_{CD}$ is able to achieve up to 11x speedup, which is on the **cnr** dataset.

A significant speedup is achieved on **cnr**, **hollywood09**, **hollywood11**, **orkut** and **ljournal**. For these datasets, the number of wedges plus triangles are much greater than the number of edges, as can be seen in Table 1. According to expression 6 and 9, PSE's performance is penalized by the number of triangles from type-1 sub-problems and wedges, whereas D4GE's performance is penalized no more than the number of edges from the symmetrised graph. This gives advantage to D4GE/S4GE$_{CD}$ compared to PSE/S4GE.

For the largest datasets, **uk02**, **enwiki18** and **indochina**, we employed a larger cluster on Compute Canada[3] using 14 compute nodes, with 48 cores and 192 GB RAM per node. This configuration

[3]https://docs.computecanada.ca/wiki/Cedar

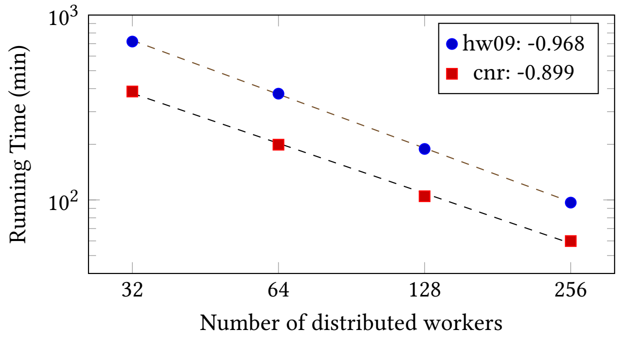

Figure 5: Machine scalability of D4GE/S4GE on *cnr* and *hollywood09*. D4GE/S4GE show very strong scalability with slope -0.899 and -0.968 which is very close to -1, the perfect value.

Table 2: The enumeration time (minutes) of D4GE/S4GE$_{CD}$ against PSE/S4GE, with ρ = 16, 120 workers.

Dataset	PSE/S4GE	D4GE/S4GE$_{CD}$	Speedup
enron	0.55	0.18	3.1
cnr	1446	132	11.0
amazon	0.37	0.37	1.0
hollywood09	2190	204	10.7
dewiki	> 7days	2328	/
hollywood11	9186	864	10.6
orkut	3799	390	9.7
ljournal	432	47	9.2

Table 3: The outputs of D4GE/S4GE$_{CD}$ with ρ = 16, on a cluster of 120 workers.

Graphlets	enron	cnr	amazon	hw09
3-path	2.51B	6.12B	372M	21.4T
3-star	8.04B	41.4T	610M	16.7T
square	21.6M	37.9B	2.69M	168B
tailed-triangle	583M	79.4B	92.3M	8.87T
diamond	46.1M	43.0B	13.1M	635B
4-clique	5M	160M	4.19M	1.39T
Running Time (min)	0.18	132	0.37	204

effectively gives us 672 workers with 4 GB RAM per worker, which can still be considered modest. On this cluster, we set ρ to 25, which gives us 15,250 sub-problems. The results are shown in Table 5. D4GE/S4GE$_{CD}$ was able to complete **uk02** in about 30 hours, **enwiki18** in 82 hours and **indochina** in 124 hours, enumerating more than 2, 7.5 and 10 quadrillion graphlets in total. We emphasize that, to the best of our knowledge, there is no existing algorithm that can enumerate all the 4-node graphlets in a dataset of this scale

Table 4: The outputs of D4GE/S4GE$_{CD}$ with ρ = 16, on a cluster of 120 workers.

Graphlets	dewiki	hw11	orkut	ljournal
3-path	10.4T	104T	18.6T	1.81T
3-star	661T	92.8T	97.8T	8.85T
square	13.1B	643B	70.1B	8.55B
tailed-triangle	993B	26.8T	1.51T	190B
diamond	11.9B	1.88T	47.8B	27B
4-clique	158M	728B	3.22B	16.1B
Running Time (min)	2328	864	390	47

Table 5: The outputs of D4GE/S4GE$_{CD}$ with ρ = 25, on a cluster of 672 workers.

Graphlets	uk02	enwiki18	indochina
3-path	1.9T	66.2T	7.6T
3-star	1.97Q	7.4Q	10.01Q
square	238B	76B	617B
tailed-triangle	6.1T	5.1T	9.3T
diamond	1.8T	61.7B	3.3T
4-clique	157B	876M	99.3T
Running Time (min)	1800	4885	7416

in a feasible amount of time. We estimate that, for each, PSE/S4GE would take more than 7 days to run using the same cluster, which is impractical. Note that 1 day = 1440 minutes.

We emphasize that the overall speedup of D4GE against PSE is also because D4GE guarantees no duplication during the enumeration. We obtained Park et al's PSE+VF2 implementation from https://datalab.snu.ac.kr/pegasusn/download.php, version 3.0.1. We modify the source code to count the number of duplicated emissions. We list the percentages of duplicate emissions from PSE/S4GE, PSE/CF, and PSE/VF2. For PSE/S4GE, we list the median percentage of the duplications for all six types of 4-node graphlets, and we query 4-clique against VF2. The results are presented in Table 7. We can see that PSE partitioning scheme, when combined with S4GE algorithm, emits aroud 300% of duplicates. The percentages are around 40% for PSE/CF4, and lower for PSE/VF2. From this table, we might deduce that PSE was indeed designed to work together with VF2, but not suited for S4GE.

5.4 PSE/VF2 vs D4GE/CF4$_{CD}$

Lastly, we compare Park et al's PSE/VF2 implementation with 4-clique query, against our D4GE/CF4$_{CD}$. The results are shown in Table 6. Comparing our D4GE/CF4$_{CD}$ suite against one of the *state-of-the-art* sub-graph enumeration algorithm, up to 5.2 fold speedup is observed on small graph such as **amazon**, and > 20 fold speedup on large graph such as **hollywood09**.

Comparing the second and the third column of Table 7 on duplicate emissions, we can see that localized VF2 algorithm emits less

Table 6: Enumeration time (minutes) of D4GE/CF4$_{\mathrm{CD}}$ against PSE/VF2, with $\rho = 16$

Dataset	PSE/VF2	CD$_{\mathrm{ext}}$/CF4$_{\mathrm{CD}}$	Speedup
enron	0.7	0.15	4.7
cnr	1.1	0.33	3.3
amazon	1.3	0.25	5.2
hollywood09	324	16	20.3
dewiki	4.5	1.5	3.0
hollywood11	288	31	9.3
orkut	16	5.0	3.2
ljournal	11	2.5	4.4

Table 7: Duplicated emissions from PSE partitioning scheme with different local algorithms.

Dataset	S4GE	CF4	VF2(K$_4$)
enron	255%	36%	19%
cnr	245%	32%	14%
amazon	270%	36%	1.2%
hollywood09	379%	41%	29%
dewiki	/	39%	25%
hollywood11	305%	41%	29%
orkut	246%	41%	29%
ljournal	309%	41%	26%

duplicated 4-cliques than CF4, when both are using the same PSE partitioning scheme. Yet, still up to 29% of duplicates are emitted by VF2, from both of **hollywood**s and **orkut** datasets.

The overall results show that D4GE/CF4$_{\mathrm{CD}}$ has better performance than PSE/VF2 for enumerating 4-cliques. However, we also acknowledge that, PSE/VF2 might suffer from its generality in this particular comparison. D4GE/CF4$_{\mathrm{CD}}$ is tuned to enumerating 4-cliques only whereas VF2 is capable of answering any k-order sub-graph query.

We also want to emphasis that the comparison here is aimed to show the performance gain of D4GE over PSE; while D4GE/S4GE$_{\mathrm{CD}}$ can be revised to query 4-cliques, it is designed for a bigger goal - enumerating all 4-node graphlets.

5.5 Discussion

It is common in the literature that performance or scalability is measured against the size of the input graph, either by the number of vertices or more commonly the number of edges. We would like to point out that in the context of 4-node graphlet enumeration, using the S4GE algorithm, the number of vertices or edges should not be the primary consideration when it comes to the amount of computation. In [28] it was shown that S4GE algorithm is bounded by $T_{3g} + (|\angle| + |\Delta|)d^{\mathrm{sym}}_{\max}$, where T_{3g} is the time to enumerate all the wedges and triangles. As a consequence, a *small* graph such as **dewiki** can have a much longer runtime than graphs of larger size, such as **ljournal**. As can be seen in Table 2, **ljournal** which is three times larger than the **dewiki** in size has a runtime that is only 2% of the **dewiki**'s. Notice that **dewiki** has a much larger number of graphlets, in particular the 3-stars. We plots the enumeration time of eight small-medium datasets against $d^{\mathrm{sym}}_{\max}(|\Delta| + |\angle|)$ in Figure 6. From our experiments, the enumeration time demonstrates high correlation with respect to $d^{\mathrm{sym}}_{\max}(|\Delta| + |\angle|)$. This shows that the total number of graphlets is the important metric to measure the performance of an enumeration algorithm. The correlation also shows that the D4GE performs as expected, i.e. it does not distort the single machine solution expectation.

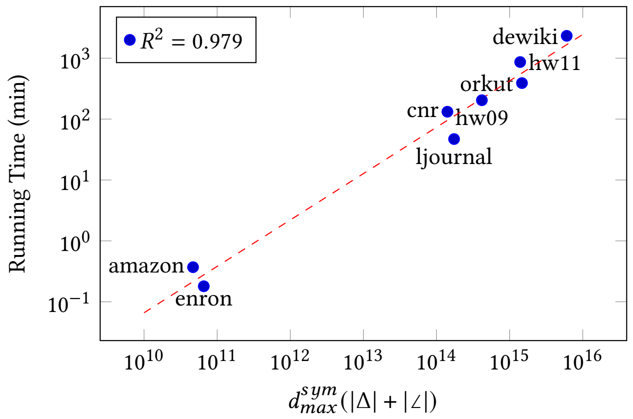

Figure 6: Strong correlation between the enumeration time and $d^{sym}_{max}(|\Delta| + |\angle|)$ on the small-medium datasets.

We have stated from the beginning that graphlets are induced connected subgraphs. Some papers in the literature focus on enumerating non-induced subgraphs instead. We should point out that we can get the non-induced subgraphs from the induced subgraphs, or vice versa, since they are correlated. However, getting the non-induced from the induced is easier than the other way around. Given a set of four vertices, the induced subgraph for this vertices (there is only one) contains all the non-induced subgraphs. On the other hand, we need to find the largest non-induced subgraph to get the induced subgraph. The counts are related by

$$N = MI \tag{10}$$

where N is the vector for the non-induced counts (3-path, 3-star, square, tailed triangle, diamond, 4-clique, in this order), and I is for the induced one. The matrix M is

$$M = \begin{bmatrix} 1 & 0 & 4 & 2 & 6 & 12 \\ 0 & 1 & 0 & 1 & 2 & 4 \\ 0 & 0 & 1 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 & 4 & 12 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Another question that the readers might ask is why enumerate all types of 4-node graphlets in a single run? Why not just one type at a time, like many other solutions? Our answer is that we can turn off any pattern that we do not want in the S4GE, and do some optimization for each. However, if we need all types of the graphlets, for a complete analysis, it will be more efficient to do all at once rather than do them one by one. Notice that due to its design, the running time of S4GE is less than the sum of the times for enumerating the six graphlet types individually.

6 PROBABILISTIC GRAPHLETS

In this section we turn our attention to Probabilistic Graphs. We follow the following definitions.

Probabilistic graph. A probabilistic graph is defined as $\tilde{G} = (G, P)$, where $G = (V, E)$ is a graph, and P is a function that assigns each edge $e \in E$ with a probability $p_e \in (0, 1]$.

In contrast to a deterministic graph G where each edge is guaranteed to exist, in a probabilistic graph $\tilde{G}$ each edge only exists with some probability.

Probabilistic graphlet. The probability of a graphlet g in a probabilistic graph $\tilde{G} = (G, P)$ to occur is equal to the product of the probabilities of all edges in the graphlet. That is, $p_g = \prod_{e \in g} p_e$.

The problem than we want to solve is as follows: Given a probabilistic graph $\tilde{G} = (G, P)$, enumerate all graphlets that have a probability to occur more than a certain value (threshold), γ. Notice that answering this question would also answer the question of *how many* that has probability greater than the threshold, i.e. the counts for each type of graphlet.

6.1 Probabilistic Enumeration Algorithm

The naive solution for probabilistic graphlets enumeration is to post-filter-out the graphlets whose probabilities are less than the threshold. However, this approach does not take advantage of the probabilistic nature at all. In fact, with this approach any γ will yield the same enumeration time, while in principle a large γ should require much less computing power as it has less number of graphlets to enumerate.

We found that S4GE is suitable to answer this question. We leverage the fact that S4GE is an intersection based algorithm, and that it discovers 4-node graphlets gradually: from edges to wedges/triangles, and then to 4-node graphlets. This gives us a hint that we can apply probability-based pruning at each stage of the discovery pipeline. First and foremost, only edges whose probabilities $p_e \geq \gamma$ need to be considered. Upon the discovery for wedges/triangles, only those with probability greater than γ need to be processed further. Lastly, the surviving wedges/triangles are used to discover 4-node graphlets. The final probability of the 4-node graphlets is checked against γ before emitted.

We highlight the additional changes made to Algorithm 1, 3, 4, 5 and 6. On line 3 of Algorithm 1, when partitioning E^{sym} into edge-sets, γ is now taken into account, such that only the edges with probability $p_e \geq \gamma$ are partitioned into E^*_{ij}. This not only disqualifies certain edges, but also reduces the sizes of the directed edge-sets, hence reduces the network read at the beginning of the enumeration phase. For Algorithm 3, 4, 5 and 6, we insert a probability check whenever a new triangle/wedge/graphlet is discovered. For Algorithm 3, whenever the third candidate vertex is discovered, we calculate the probability by $P((u, v)) * P((u, v')) * P((v, v'))$ for a triangle, $P((u, v)) * P((u, u'))$ for a Type-1 wedge and $P((u, v)) * P((v, v'))$ for a Type-2 wedge. The calculated probability, if greater than or equal to γ, is then passed into the respective explore functions for candidacy check of the 4th vertex, where the probability of the graphlet is calculated and checked against γ before the emission.

6.2 Experiments

For our experiments, probabilistic graph datasets are generated from deterministic datasets used in Section 5 above. We chose five undirected graphs that require moderate to high workload: **ljournal**, **hollywood09**, **orkut**, **hollywood11** and **dewiki**. We used a random number generator that follows uniform distribution to assign a probability $p \in (0, 1]$ to each edge of the graphs, and ran our probabilistic implementation of D4GE/S4GE$_{\text{CD}}$ on them.

We did several runs with varying threshold, γ. For each graph, we increase the probability threshold, from 0.1 to 0.9, at a step of 0.1. For a reference, the results of deterministic enumerations are also included, which can be viewed as $\gamma = 0$. The results are displayed in Figure 8 for all types of 4-node graphlets. The running times are shown in Figure 7.

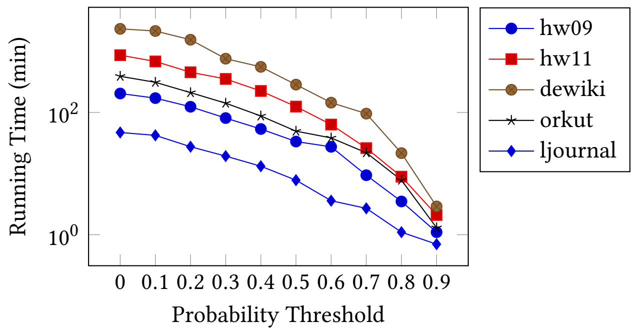

Figure 7: Enumeration time at different probability threshold.

6.3 Discussion

From Figure 7 we can observe the decreasing trend in enumeration time with the increasing probability threshold. This trend can be observed in all five sample graphs. If we look at the **dewiki** curve, for example, with the probability threshold increased from 0.1 to 0.5, the enumeration is greatly reduced, from above 2.3K minutes to 285 minutes, a reduction by a factor of 8. The running time keeps decreasing as the threshold increases. As γ approaches to 1, the running time approaches 0, as expected, because there will be no edge that passes the threshold.

The number of emitted graphlets from different input graphs are listed from Figure 8a to Figure 8e respectively. Again we can observe the decreasing trend in the enumerated graphlets of all six types, across all the inputs. In particular, the 4-clique is the one mostly impacted by the increasing probability threshold. This is expected as 4-clique has the highest number of edges, and the probability of a 4-clique is the product of the probabilities of its six edges. Since each edge has a probability less than or equal to 1, the likely-hood for the product of six to be smaller than a threshold is always greater than the product of five (tailed-triangle and diamond), four (square) and three (3-path and 3-star). Nonetheless, they all should converge to zero when the threshold is close to one. Also, keep in mind that the

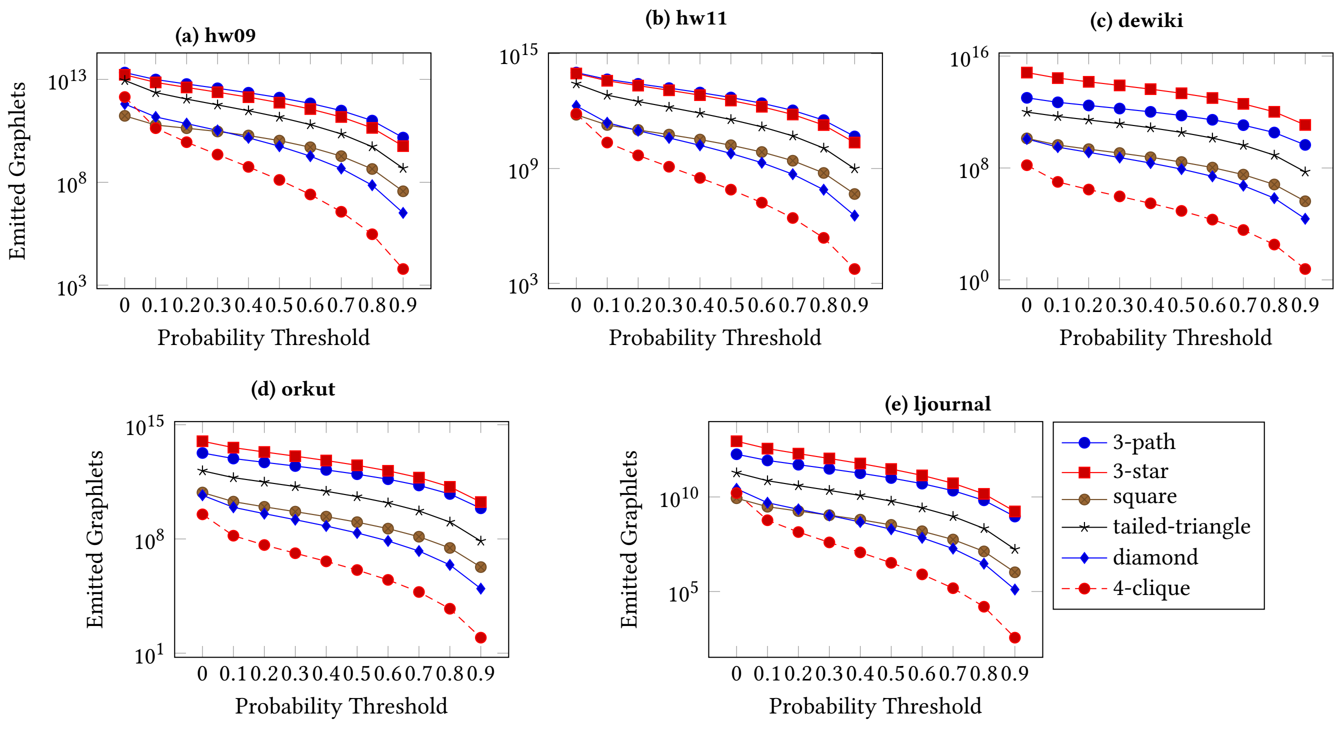

Figure 8: Probability graphlets enumeration results.

probabilistic graphs that we use in this experiment have uniform probabilistic distribution for the edges.

Overall, the results imply that while enumerating all six types of 4-node graphlets can be computational challenging, if the use case allows for omitting edges that meet certain criteria, the computational requirements can be reduced by a large factor.

In a probabilistic graph, each edge has a chance to exist, or not. Given an input graph $G(V, E)$ with m edges, in principle, there could be 2^m probable graphs, G_i, each with probability p_i. Notice that once a probable graph occurs it can be treated as a deterministic graph, and that each G_i is a disjoint possibility. Then, we have $\sum_{i=1}^{2^m} p_i = 1$. In general, a specific graphlet can be found inside many probable graphs. The probability of a graphlet can also be interpreted as the probability for any of the probable graphs that contain the graphlet to occur. Let g be a graphlet. Then, according to the Law of Total Probability we have $\Pr(g) = \sum_{i=1}^{2^m} \Pr(g \cap G_i) = \sum_{i=1}^{2^m} \Pr(g|G_i)\Pr(G_i)$. Now, given that G_i is deterministic, $\Pr(g|G_i) = 1$ if G_i contains g or 0 if G_i does not contain g. So

$$\Pr(g) = \sum_{i|g \in G_i} \Pr(G_i) \tag{12}$$

From the enumeration we get the cumulative counts of the graphlets, as shown in Figure 8. We can then compute the differential counts in each interval of the threshold, from 0 to 0.1, from 0.1 to 0.2 and so on, from which we can estimate the mean counts of the graphlets in the given probabilistic graph by

$$N_{\text{avg}} = \sum_j 1 \cdot \Pr(g_j) \approx \sum_\alpha n_\alpha P_\alpha \tag{13}$$

where n_α is the count of graphlets with probabilities within the interval α and P_α is the middle value of the interval. The estimates are listed below. The columns are for 3-path, 3-star, rectangle, tailed-triangle, diamond, and 4-clique, respectively. We can see that, for example, the average 4-clique for **dewiki** is estimated to be 16.5M, just above 10% of the count in the deterministic graph. Keep in mind that here we employ uniform probability distribution for the edges. In general, the percentages would depend on the probability distribution of the edges.

hw09:	N_{avg}	3.93T	2.83T	30.1B	1.18T	80.9B	140B
	Pct	18.4%	17.0%	17.9%	13.3%	12.7%	10.1%
hw11:	N_{avg}	18.1T	15.3T	93.1B	3.51T	211B	73.9B
	Pct	17.5%	16.5%	14.5%	13.1%	11.2%	10.1%
dewiki:	N_{avg}	1.89T	105T	1.89B	171B	1.51B	16.5M
	Pct	18.3%	16.0%	14.5%	17.3%	12.6%	10.4%
orkut:	N_{avg}	3.30T	15.7T	9.44B	232B	5.63B	332M
	Pct	17.7%	16.0%	13.5%	15.4%	11.8%	10.3%
ljournal:	N_{avg}	0.33T	1.42T	1.37B	29.8B	3.25B	1.64B
	Pct	18.2%	16.1%	16.1%	15.7%	12.0%	10.2%

7 CONCLUSION

In this paper, we have presented D4GE scheme that brings 4-node graphlets enumeration into the distributed platform. This scheme is able to suppress duplicate emissions which are unavoidable with other schemes. We show that, D4GE when combined with S4GE, requires at most 2 m^{sym} more network read compared to PSE/S4GE, and is able to reduce the amount of work relative to PSE/S4GE for real world graphs, where the numbers of wedges and triangles are order of magnitudes greater than the number of edges. We experimented D4GE with S4GE_{CD} and CF4_{CD} localized algorithms, and both combinations out-perform the *state-of-the-art* competitors by up to 11x. Last but not least, D4GE/S4GE_{CD} is the only known algorithm capable of simultaneously enumerating all 4-node graphlets on dataset with almost 20 million nodes and a half billion edges.

Finally, we have shown that our algorithm can be generalized to enumerate graphlets in probabilistic graphs. With careful pruning, the running time can be reduced depending on the probabilistic threshold. With our solution, we were able to estimate the mean counts of the graphlets 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] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting graphlets: Space vs time. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 557–566.

[3] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1651–1663.

[4] Matthias Bröcheler, Andrea Pugliese, and Venkatramanan S Subrahmanian. 2010. COSI: Cloud oriented subgraph identification in massive social networks. In *2010 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 248–255.

[5] Hongzhi Chen, Miao Liu, Yunjian Zhao, Xiao Yan, Da Yan, and James Cheng. 2018. G-miner: an efficient task-oriented graph mining system. In *Proceedings of the Thirteenth EuroSys Conference*. 1–12.

[6] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26, 10 (2004), 1367–1372.

[7] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 589–598.

[8] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. 2005. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering* 17, 8 (2005), 1036–1050.

[9] Vinicius Dias, Carlos HC Teixeira, Dorgival Guedes, Wagner Meira, and Srinivasan Parthasarathy. 2019. Fractal: A general-purpose graph pattern mining system. In *Proceedings of the 2019 International Conference on Management of Data*. 1357–1374.

[10] Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. 2020. Nucleus Decomposition in Probabilistic Graphs: Hardness and Algorithms. *arXiv preprint arXiv:2006.01958* (2020).

[11] Katherine Faust. 2010. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks* 32, 3 (2010), 221–233.

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

[13] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* 21, suppl_1 (2005), i213–i221.

[14] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science* 407, 1-3 (2008), 458–473.

[15] 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

[16] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. 2019. Linc: a motif counting algorithm for uncertain graphs. *Proceedings of the VLDB Endowment* 13, 2 (2019), 155–168.

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

[18] Daniel Mawhirter, Sam Reinehr, Connor Holmes, Tongping Liu, and Bo Wu. 2019. GraphZero: Breaking Symmetry for Efficient Graph Mining. *arXiv preprint arXiv:1911.12877* (2019).

[19] Frank McSherry, Michael Isard, and Derek G Murray. 2015. Scalability! But at what {COST}?. In *15th Workshop on Hot Topics in Operating Systems (HotOS {XV})*.

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

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

[22] Ha-Myung Park, Sung-Hyon Myaeng, and U Kang. 2016. Pte: Enumerating trillion triangles on distributed systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1115–1124.

[23] Ha-Myung Park, Francesco Silvestri, Rasmus Pagh, Chin-Wan Chung, Sung-Hyon Myaeng, and U Kang. 2018. Enumerating trillion subgraphs on distributed systems. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 6 (2018), 1–30.

[24] 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.

[25] 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.

[26] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. 2005. Graph kernels for chemical informatics. *Neural networks* 18, 8 (2005), 1093–1110.

[27] Xuguang Ren, Junhu Wang, Wook-Shin Han, and Jeffrey Xu Yu. 2019. Fast and robust distributed subgraph enumeration. *arXiv preprint arXiv:1901.07747* (2019).

[28] Yudi Santoso, Venkatesh Srinivasan, and Alex Thomo. 2020. Efficient Enumeration of Four Node Graphlets at Trillion-Scale.. In *23rd EDBT*. 439–442.

[29] Thomas Schank and Dorothea Wagner. 2005. Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. In *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*. 606–609. https://doi.org/10.1007/11427186_54

[30] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting Triangles and the Curse of the Last Reducer. In *Proceedings of the 20th International Conference on World Wide Web (WWW '11)*. ACM, New York, NY, USA, 607–614. https://doi.org/10.1145/1963405.1963491

[31] Nilothpal Talukder and Mohammed J Zaki. 2016. A distributed approach for graph mining in massive networks. *Data Mining and Knowledge Discovery* 30, 5 (2016), 1024–1052.

[32] 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.

[33] Andrei Todor, Alin Dobra, and Tamer Kahveci. 2015. Counting motifs in probabilistic biological networks. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*. 116–125.

[34] Jia Wang and James Cheng. 2012. Truss Decomposition in Massive Networks. *Proceedings of the VLDB Endowment* 5, 9 (2012).

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

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

[37] Da Yan, Guimu Guo, Md Mashiur Rahman Chowdhury, M Tamer Özsu, Wei-Shinn Ku, and John CS Lui. 2020. G-thinker: A Distributed Framework for Mining Subgraphs in a Big Graph. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1369–1380.

[38] Hao Zhang, Jeffrey Xu Yu, Yikai Zhang, Kangfei Zhao, and Hong Cheng. 2020. Distributed subgraph counting: a general approach. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2493–2507.