Four Node Graphlet and Triad Enumeration on Distributed Platforms

Yudi Santoso[1*], Xiaozhou Liu[1], Venkatesh Srinivasan[1] and Alex Thomo[1]

[1]Department of Computer Science, University of Victoria, Victoria, BC, Canada.

*Corresponding author(s). E-mail(s): santoso@uvic.ca;
Contributing authors: superliuxz@gmail.com; srinivas@uvic.ca; thomo@uvic.ca;

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 the scale of the graph that they can process. Distributed computing is often proposed as a solution to improve the maximum scale. 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 and triads in directed graphs using a distributed platform. We propose an efficient distributed solution that significantly surpasses the existing solutions on the scale and performance. 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. Our experimental results show that our solution has a strong machine scalability close to one.

Keywords: distributed computing, graph mining, massive networks, subgraph enumeration

1 Introduction

Network or graph analytical methods have proven to be invaluable in data analysis. Analysing a big network - with thousands or more nodes and edges, however, is not a trivial task and a lot of studies have been done to improve the methods. Here we focus on methods for enumerating small subgraphs in a very large network. In real world networks, we often see small structures inside them which act as the building blocks, defining the characteristics of the networks. For example, in a social network, there are small sets of interconnected nodes, each set represents a group of friends. Therefore, finding these small structures, or subgraphs, is an important component in graph analysis.

Indeed, many problems in network/graph analysis require enumeration and/or counting of small subgraphs. Such problems can be found in various fields: in biology [1, 2] chemistry [3, 4], social study [5, 6], network analysis and classification [7], and more. Here we focus on graphlets, which are defined as small *induced* subgraphs (See Section 3 for the definition.). 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 order. For example, Milenkovic and Przulj [2] used 2, 3, 4, and 5 node graphlets to analyse Protein-Protein-Interaction (PPI) networks.

Graphlet enumeration problem is challenging when the input graph is very large, with millions of nodes and edges. In fact, until the recent work [8] (for a 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 [9]. 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 $\binom{n}{k} \propto n(n-1)\dots(n-k+1)$ 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 have been built by minimizing unnecessary checking. Nonetheless, it is generally true that the number of subgraphs, and hence the enumeration time, 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 an important role in graph analysis, including computing the clustering coefficient [10] and truss decomposition [11]. The best known (distributed) solution for triangle enumeration can process a graph of five billion nodes [12]. Four-node subgraph enumeration is already challenging. Known solutions are limited in term of the scale, or the size of the graph that can be processed in a reasonable amount of time. Counting (either exact [9, 13] or approximate [14, 15]) can do more, but we focus on enumeration, which is a more challenging problem. Using a distributed platform, where the tasks

are distributed to many machines in a cluster, is an obvious option to increase the computing power. 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 challenges. If this is not done properly, we will get a poor scalability, where scalability is measured as performance over cost [16]. A poor scalability would not justify the economical cost of the distributed platform. Distributing a computational task to many machines often entails redundant or duplicate computations. Therefore, we need to optimize our distributed solution to minimize this redundancy.

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 has been proposed in [8]. It has a run time that 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 that do one type or pattern at a time. Nonetheless, it was 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. We found that the graph partitioning scheme of Park et al. [12, 17] is suitable for our purpose.

Motivated by the fact that many real world networks are directed graphs, we also study distributed graphlet enumeration for directed graphs. Directed graphs contain more detailed pictures of the networks. For example, by using triads, which are three node directed graphlets with all three nodes connected to each other, we can analyze transitivity more accurately [18]. Directed clustering coefficient can be used as a measure of systemic risk in complex banking networks [19]. Also, triad enumeration is an important element in social network analysis [20].

For directed case, there are three possible links between a pair of nodes, and because of this the number of subgraph types becomes numerous. For the moment we restrict ourselves to triads. There are already seven types of triads that we need to consider. We proposed a single machine solution for triad enumeration in [21]. Here, we expand the study by including more datasets and analyses, and bring the solution onto distributed platforms. We show that our distributed solution for directed graphs can achieve strong machine scalability, similar to the undirected case. We will then provide a road plan on how to extend this solution to four nodes.

Our contributions are summarized as follows:

- We devise 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 of this algorithm to prove its correctness and efficiency.
- We build an implementation of our proposed solution in Spark available at https://github.com/D4GE/D4GE.

- We perform 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 succeed in processing a larger graph of a size that has never been processed before in a reasonable amount of time, enumerating quadrillions of graphlets using a modest cluster of machines [1].
- We also expand our solution to enumerate triads on a distributed platform. We test our solution on several massive directed graphs, and show its scalability through experimentation.

2 Related Work

Subgraph enumeration has received a lot of attention in the literature quite recently. There are many papers on triangle enumeration, including [22] and [23] which contains Compact-Forward algorithm - the most effective serial algorithm for triangle enumeration. On higher order subgraphs, most papers focus on the counting, such as [13, 24, 25], among others. Previous solutions for the induced subgraph enumeration, such as FanMod [26] and Rage [27], do not perform well on million-scale graphs. Cliques, or complete subgraphs, are easier to enumerate, with the latest solution in [28].

Distributed solutions have been studied extensively in the last two decades. For the triangle enumeration problem, partition-based solution was discussed in [29], where some load-balancing problem was exposed. Park et al. proposed a more efficient solution called PTE (Pre-partitioned Triangle Enumeration) [12]. They later generalized PTE to enumerate query subgraphs of various order, and called their solution PSE (Pre-partitioned Subgraph Enumeration) [17]. It uses VF2 [2] algorithm [30] 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 [31] and its descendant Fractal [32], DistGraph [33], GraphZero [34], G-Miner [35], G-thinker [36], RADS [37] 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 [8], 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 call it S4GE (Simultaneous Four-node Graphlet Enumeration). We use this algorithm, with some modification, as the serial algorithm for our distributed solution.

Triad census algorithm by Batagelj and Mrvar [39], which is employed in Pajek, had been known as the standard algorithm to enumerate triads for

[1] By modest here we mean a cluster with less than one thousand machines.
[2] This is the name given by the authors of [30], and as far as we know is not an abbreviation.

several years. Although this algorithm can perform triad enumeration in sub-quadratic time, it is not fast enough for very large graphs with millions of nodes and edges.

Chin et al. [40] developed a compact data structure that makes it easier to parallelize the computation. The idea is to encode the link information using 2-bits for each node in the adjacency list. Suppose the nodes were labeled by using 32-bit integers. The bits are shifted to the left by two, and the two lowest bits are then used for the edge direction. Thus, only 30 bits can actually be used to label the nodes. Parimalarangan et al. [41] took on this idea and combined it with the most efficient algorithms known for triangle enumeration on single machines [23]. In [21], we proposed a solution, FPTE (Four Pointers Triad Enumeration), that computes the link information on the fly, and hence frees two more bits to label the nodes. With this solution we were able to process larger graphs, compared to [40] and [41]. Moreover, it was empirically shown to be faster. Nevertheless, none of these solutions for directed graphs are for distributed platform.

To the best of our knowledge, there is no distributed solution to enumerate all the induced 4-node graphlets, or directed graphlets, that can scale to large graphs using a modest cluster of machines.

3 Preliminaries

3.1 Graphs and Graphlets

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. The number of nodes, $n = |V|$, is known as the order of the graph, and the number of edges, $m = |E|$, is known as the size of the graph. Here we assume that the graph is simple, i.e., without any 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 $d(u) = |N(u)|$.

Subgraph. A graph $H = (V_H, E_H)$ is a subgraph of $G = (V_G, E_G)$ if $V_H \subseteq V_G$, and $E_H \subseteq E_G$. In this case we write $H \subseteq G$.

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 an edge-induced subgraph is not the same as an induced subgraph, as an induced subgraph is a subgraph of chosen vertices while an edge-induced subgraph is a subgraph of chosen edges. This is illustrated by the example in Figure 1. 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 Figure 2. There are

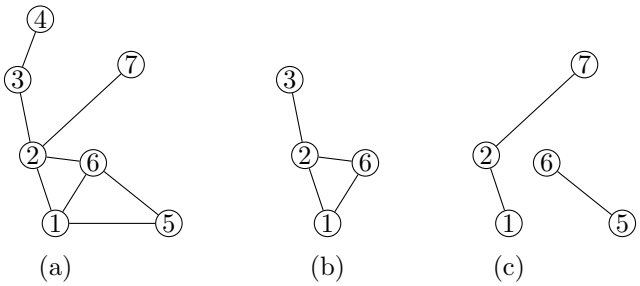

Fig. 1 (a) A graph, (b) an induced subgraph (induced by vertex set $\{1, 2, 3, 6\}$) and (c) an edge-induced subgraph (induced by edge set $\{1-2, 2-7, 5-6\}$). Note that (c) is not an induced subgraph of (a).

Fig. 2 Three node graphlets: a wedge and a triangle.

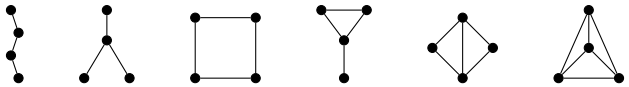

Fig. 3 Four node graphlets: a 3-path, a 3-star, a rectangle or 4-cycle, a tailed-triangle, a diamond, and a 4-clique.

six types of four node graphlets, as shown in Figure 3. They are 3-path (or 4-node-path), 3-star, rectangle (or 4-cycle), lollipop (or tailed-triangle), diamond (or 4-chordal-cycle) and 4-clique.

Cliques. A complete graph is a graph in which every pair of nodes is 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.

3.2 Graph Partitions

For distributed computation we partition the graph using a coloring scheme as in [12]. Here are the definitions used in 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 that 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 by $N^+(u)$. The out-degree of vertex u is denoted by $d^+(u) = |N^+(u)|$.

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). Note that orienting the edges does not change the edge set.

Symmetrization. Symmetrization is the process of making all edges in a directed graph bi-directional. As we will see below, symmetrization is needed for our distributed solution.

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 $k = 3$ (i.e., for triangles), 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 $S_{ij} \not\subset S_{ijk}$.

4 Graphlets Enumeration

Compact-Forward [23] is known as the most effective serial algorithm for triangle enumeration. It utilizes the DAG technique to optimize the efficiency. For a given ordered-by-degree input DAG $\overrightarrow{G}(V, \overrightarrow{E})$, Compact-Forward enumerates all triangles using $O(|\overrightarrow{E}|^{3/2})$ operations.

In our implementation, we use a variant of Compact-Forward where the orientation task is done separately before the enumeration, in what we call the graph preparation phase (Algorithm 1). The triangle enumeration part is listed in Algorithm 2. If we input a graph that has already been prepared, then lines 3 and 5 of Algorithm 2 are unnecessary because the graph preparation basically provides us with a DAG.

Note that a DAG is effective because of the symmetrical property of triangles. That is, if we have a triangle with nodes i, j and k, we can list the triangle by any one of the six possible permutations of the nodes. Suppose, $i < j < k$, then we only need to list the triangle as (i, j, k). Thus, by imposing an ordering on the nodes through DAG we can follow the order and make sure that we include all triangles without double counting.

Algorithm 1 Graph-Prep

Require: 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 (i.e., neighbours that are lower in the ordering of the nodes) from each neighbour list.

Algorithm 2 Triangle Enumeration

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

```
1:  for all vertex u ∈ V(G) do
2:      for all vertex v ∈ N(u) do
3:          if v > u then
4:              for all w ∈ N(u) ∩ N(v) do
5:                  if w > v then
6:                      EnumerateTriangle (u, v, w)
7:                  end if
8:              end for
9:          end if
10:     end for
11: end for
```

For this reason, enumerating wedges is actually more complicated than enumerating triangles. Take an arbitrary wedge, the lowest node might either be at the center or at one of the legs. This leads to two types of wedges, which need to be enumerated differently.

4.1 Four Node Graphlet Enumeration

By observing Figure 3 we noticed that a tailed triangle contains one triangle, while a diamond contains two triangles and a 4-clique contains four triangles. The other three do not contain triangles but they contain wedges. This gave us an idea to enumerate four node graphlets in a cascading manner, by first searching for three node graphlets and then for each that is found, searching for four node graphlets.

Our solution for a single machine has been described in detail in [8]. We call it S4GE (*Simultaneous 4-node Graphlet Enumeration*) [3]. For this, we extend the Triangle Enumeration to include wedges as well. The pseudo code is listed in Algorithm 3 [4]. There are two types of wedges: type 1 has the center as the lowest node, type 2 has one leg as the lowest node. Here, ExploreTriangle, ExploreWedgeType1, and ExploreWedgeType2 can be considered as placeholders for listing the triangles and wedges.

[3] It was not named in the original paper.
[4] We have a correction in this algorithm from the original version, on lines 9 and 12, to enumerate all wedges properly.

Algorithm 3 TRIANGLE AND WEDGE ENUMERATION

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

for all vertex $u \in V(G)$ **do**
  for all vertex $v \in N(u)$ **do**
    if $u < v$ **then**
      for all $u' \in N(u)$ and $v' \in N(v)$ **do**
        if $(u' > u) \wedge (v' > u)$ **then**
          if $u' = v' > v$ **then**
            EnumerateTriangle (u, v, u')
          end if
          if $((u' < v') \vee (v' = u)) \wedge (u' > v)$ **then**
            EnumerateWedgeType1 (v, u, u')
          end if
          if $(u' > v') \wedge (v' \neq u)$ **then**
            EnumerateWedgeType2 (u, v, v')
          end if
        end if
      end for
    end if
  end for
end for

For four node graphlets, we extend the EnumerateTriangle and EnumerateWedge functions to include calls to ExploreTriangle and ExploreWedge, respectively. Whenever a triangle, (u, v, w), is found, the ExploreTriangle function (Algorithm 4) is then called. This function 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) is a four-clique. A node z that is in two of the three neighbour sets gives us a diamond, while a z that is in only one of the three neighbour sets gives us a tailed triangle. For 4-cliques, because of the symmetry we only need to consider the sets of larger neighbours, i.e., $z > u, v, w$. 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 lose 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.

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 4 ExploreTriangle

Require: 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 $(u, v, w, z)_8$
end for
for all z in two sets and $z >$ opposite node **do**
    EnumerateDiamond $(.)_7$
end for
for all z in one set only **do**
    EnumerateTailedTriangle $(.)_6$
end for

Algorithm 5 ExploreWedgeType-1

Require: 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 $(u, v, z, w)_5$
end for
for all $z \in N^{>u}(u)$ only **do**
    if $z > w$ **then**
        Enumerate3Star $(u, v, w, z)_4$
    end if
end for
for all $z \in N^{>u}(v)$ only **do**
    Enumerate3Path $(w, u, v, z)_3$
end for
for all $z \in N^{>u}(w)$ only **do**
    Enumerate3Path $(v, u, w, z)_3$
end for

Algorithm 6 ExploreWedgeType-2

Require: 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 ∈ N^{>u}(v) only do
    if z > w then
        Enumerate3Star (v, u, w, z)_4
    end if
end for
for all z ∈ N^{>u}(w) only do
    if z ≠ v then
        Enumerate3Path (u, v, w, z)_3
    end if
end for
```

The runtime of S4GE 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. This bound comes from the fact that for each triangle or 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 $(N_\Delta + N_\angle) \lesssim n d_{\max}^2$, with the upper value is satisfied by a regular graph. However, for all real world networks that we have considered so far, we have $(N_\Delta + N_\angle) \ll n d_{\max}^2$. Also, $T_{3g} \ll n d_{\max}^2$ using an efficient enumeration. Therefore, in practice, our runtime is much less than the worst case bound of $O(n d_{\max}^3)$.

4.2 Previous Distributed Enumeration

Park et al. [12] 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. reduced 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) [17]. 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 sub-graphs matching G_q. PSE employs VF2 algorithm [30] as the local serial algorithm for query graph matching. We stress that VF2 can only take 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 $|E|$.

Note that PTE can only enumerate triangles, while PSE can enumerate graphlets of any size (given the appropriate serial algorithm to do the task). However, PSE is not fine-tuned for enumerating four-node graphlets using the S4GE as the serial algorithm.

5 Proposed Methods

5.1 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 from 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 assignments 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.

Algorithm 7 D4GE

Require: An undirected graph $G(V, E)$; the number of colors ρ

Construct $\overrightarrow{G}(V, \overrightarrow{E})$ by applying edge-orientation to $G(V, E)$
Symmetrise $\overrightarrow{G}(V, \overrightarrow{E})$ into $G^{\text{sym}}(V, E^{\text{sym}})$
Partition E^{sym} into directed edge sets E^*_{ij} using ρ
Generate ordered color assignments and sub-problems $\{S_{\text{Cs}} \mapsto \{K_{ijkl}\}\}$
for all S_{Cs}, $\{K_{ijkl}\}$ **do** ▷ Distributed-for
    $E_{\text{map}} \leftarrow \text{ReadEdgeSets}_{\text{CD}}\ (S_{\text{Cs}}, 4)$
    for all $K_{ijkl} \in \{C_0, C_1, ...\}$ **do**
        $\text{S4GE}_{\text{CD}}\ (E_{\text{map}}, ijkl)$
    end for
end for

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 7 and Algorithm 8. 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).

Algorithm 8 ReadEdgeSets$_{\text{CD}}$

Require: Sub-problem S_{Cs} with $Cs = \{c_0, c_1, ..., c_l\}$; order of query graph k

Initialize empty map $E_{\text{map}} \equiv \{(i, j) \mapsto E^*_{ij}\}$
for all $(i, j) \in Cs^2$ **do**
    if $i = j$ and $|\{c_0, c_1, ..., c_l\}| \neq k$ **then**
        $E_{\text{map}}[(i, i)] \leftarrow E^*_{ii}$
    else
        $E_{\text{map}}[(i, j)] \leftarrow E^*_{ij}$
    end if
end for
return E_{map}

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 Figure 4 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.

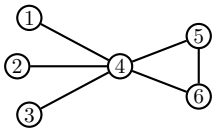

Fig. 4 A graph illustrating the need for symmetrization.

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 over 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 modified S4GE algorithm.

Algorithm 9 S4GE$_{\text{CD}}$

Require: 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 $ijkl$

$E^*_{ij} \equiv E_{\text{map}}[ij]$, $E^*_{ik} \equiv E_{\text{map}}[ik]$, $E^*_{jk} \equiv E_{\text{map}}[jk]$
for all $(u,v) \in E^*_{ij}$ **do**
  if $\eta(u) < \eta(v)$ **then**
    for $u' \in N(u) \subset E^*_{ik}$ and $v' \in N(v) \subset E^*_{jk}$ **do**
      if $(u' > u) \wedge (v' > u)$ **then**
        if $u' = v' > v$ **then**
          DExploreTriangle $(u, v, u', E_{\text{map}}, ijkl)$
        end if
        if $(u' < v') \wedge (u' > v)$ **then**
          DExploreWedge-1 $(v, u, u', E_{\text{map}}, ijkl)$
        end if
        if $u' > v'$ **then**
          DExploreWedge-2 $(u, v, v', E_{\text{map}}, ijkl)$
        end if
      end if
    end for
  end if
end for

5.2 S4GE$_{\text{CD}}$

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$_{\text{CD}}$. The pseudocode for S4GE$_{\text{CD}}$ is given in Algorithm 9, and the details of the explore-functions are given in Algorithms 10, 11, and 12 respectively.

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 consists 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

Algorithm 10 DExploreTriangle

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

$N^{>u}(u) \equiv \{z \mid z \in N(u)|_{E_{\text{map}}[il]},\ \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z \mid z \in N(v)|_{E_{\text{map}}[jl]},\ \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z \mid z \in N(w)|_{E_{\text{map}}[kl]},\ \eta(z) > \eta(u)\}$
for all $z \in N^{>u} \cap N^{>u}(v) \cap N^{>u}(w)$ with $z > w$ **do**
    EnumerateFourClique (u, v, w, z)
end for
for all z in two sets and $z >$ opposite node **do**
    EnumerateDiamond (u, v, w, z)
end for
for all z in one set only **do**
    EnumerateTailedTriangle (u, v, w, z)
end for

applied to different edge-based enumeration algorithms to suit different needs. All it requires is to modify the input to accommodate a directed sub-set of the input graph.

Algorithm 11 DExploreWedge-1

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

$N^{>u}(u) \equiv \{z \mid z \in N(u)|_{E_{\text{map}}[il]},\ \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z \mid z \in N(v)|_{E_{\text{map}}[jl]},\ \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z \mid z \in N(w)|_{E_{\text{map}}[kl]},\ \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ **do**
    EnumerateRectangle (u, v, z, w)
end for
for all $z \in N^{>u}(u)$ only **do**
    if $z > w$ **then**
        EnumerateThreeStar (u, v, w, z)
    end if
end for
for all $z \in N^{>u}(v)$ only **do**
    EnumerateThreePath (w, u, v, z)
end for
for all $z \in N^{>u}(w)$ only **do**
    EnumerateThreePath (v, u, w, z)
end for

Algorithm 12 DExploreWedge-2

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

$E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
$N^{>u}(v) \equiv \{z \mid z \in N(v)|_{E^*_{jl}},\ \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z \mid z \in N(w)|_{E^*_{kl}},\ \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(v)$ only **do**
    if $z > w$ **then**
        Enumerate3Star (v, u, w, z)
    end if
end for
for all $z \in N^{>u}(w)$ only **do**
    if $z \neq v$ **then**
        Enumerate3Path (u, v, w, z)
    end if
end for

5.3 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_{CD}, is given as Algorithm 13.

Algorithm 13 CF4_{CD}

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

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

Note that CF4 extends the idea of Compact-Forward algorithm from triangles to four-cliques, hence the name CF4. The correctness of CF4_{CD} is intuitive. CF4_{CD} does $O(m^2)$ work for a given graph.

5.4 Analysis

In this analysis, first we show that D4GE with S4GE_{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_{CD} algorithm on each individual color assignment. Since S4GE correctly enumerates all 4-node graphlets for any given graph, $\text{D4GE/S4GE}_{\text{CD}}$ 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_{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_{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:

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

If we assume the edges are distributed evenly, the expected size of $m_{\neq}^{\text{sym}}$ 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

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

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_{CD} as the localized algorithm. Since S4GE_{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 $2\,m^{\text{sym}}d_{\text{max}}^{\text{sym}}$, where $d_{\text{max}}^{\text{sym}}$ is the maximum degree of the symmetrised graph. Following the analysis to derive expression 1, D4GE does

$$
2\binom{\rho}{2}m_{=}^{\text{sym}}d_{\text{max}}^{\text{sym}}+2\left[1+\binom{\rho-1}{2}\right]m_{\neq}^{\text{sym}}d_{\text{max}}^{\text{sym}} \qquad (3)
$$

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_{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. $\text{D4GE/S4GE}_{\text{CD}}$ 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_{CD} does

$$3\,\rho\, d^{\text{sym}}_{\max}(|\Delta| + |\angle|) \tag{7}$$

amount of work.

As discussed briefly at the beginning of Section 5, PSE may discover a four-node-graphlet in more than one subproblem group. The number of duplications depend on the number of colors of the triangles and wedges. This, in turn, determines the amount of work. We denote the uni-color triangles and wedges by Δ_I and $\angle_I$, the bi-color ones by Δ_{II} and $\angle_{II}$, and the tri-color ones by

Δ_{III} and $\angle_{III}$. We can write

$$\begin{aligned} W_I^{\text{PSE}} &= (1 + a(\rho))\, 3\, d_{\max}^{\text{sym}}(|\Delta_I| + |\angle_I|) \\ W_{II}^{\text{PSE}} &= (1 + b(\rho))\, 3\, d_{\max}^{\text{sym}}(|\Delta_{II}| + |\angle_{II}|) \\ W_{III}^{\text{PSE}} &= (1 + c(\rho))\, 3\, d_{\max}^{\text{sym}}(|\Delta_{III}| + |\angle_{III}|) \end{aligned} \tag{8}$$

where $a(\rho), b(\rho), c(\rho)$ are positive functions of ρ representing the duplications in the three types. Their exact values depend on the instance of the input graph.

Expression 8 minus expression 7 yields

$$3\, d_{\max}^{\text{sym}}\ (a(\rho)\,(|\Delta_I| + |\angle_I|) + b(\rho)\,(|\Delta_{II}| + |\angle_{II}|) + c(\rho)\,(|\Delta_{III}| + |\angle_{III}|)) \tag{9}$$

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, grows with respect to ρ and, is sensitive to the number of 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.

6 Experiments

Our solution, D4GE, is implemented in Apache Spark 2.4.5 with OpenJDK 1.8.0. We experimented with it on several large real world datasets. The graph datasets were collected from http://law.di.unimi.it/datasets.php in WebGraph format, except for the **orkut** dataset, and symmetrized to get undirected graphs. The **orkut** dataset was retrieved from https://snap.stanford.edu/data/com-Orkut.html. The statistics of the symmetrized graphs are listed in Table 1.

For the smaller graphs, 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[5] and 1.5 GB of RAM per worker. For the three largest graphs, **uk02**, **enwiki18** and **indochina**, we employed a larger cluster on Compute Canada[6] using 14 compute nodes, with 48 cores and 192 GB RAM per node. This configuration effectively gives us 672 workers with 4 GB RAM per worker, which can still be considered modest.

[5] Each worker is equivalent to a physical CPU core.
[6] https://docs.computecanada.ca/wiki/Cedar

Table 1 The numbers of vertices n, edges m, wedges $|\angle|$, and triangles $|\Delta|$, and the max degree of the symmetrized graphs. The last three graphs are the largest and they require more computing power than the others.

Dataset	n	m	$\|\angle\|$	$\|\Delta\|$	$d_{\max}^{\text{sym}}$
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

We compared the performance with the performance of the SotA, the PSE, and the single machine solution S4GE. The single machine experiment was conducted using a machine with dual Xeon E5-2620 processors and 128 GB of RAM. The total number of threads in this machine is 24. We set our time budget to be six days for each run.

6.1 The impact of ρ on performance

Let us first address the impact of ρ on the overall performance of our distributed algorithm. In a previous literature, Suri and Vassilvitskii [29] 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. [12] 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$_{\text{CD}}$ on three different datasets and varying $\rho = 8, 12, 16$ and 20. The result is shown in Figure 5.

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 $\rho = 8$,

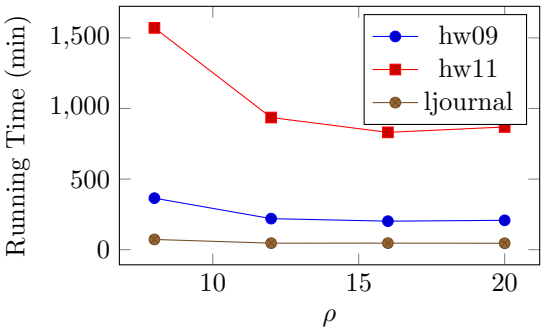

Fig. 5 The enumeration time (minutes) of D4GE/S4GE$_{\text{CD}}$ on several graphs, with varying value of ρ. Higher ρ does not add much overhead; the lines flatten out rather than sloping up perceptibly.

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 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.

6.2 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 6. 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 [12] and [17].

6.3 S4GE vs D4GE/S4GE$_{\text{CD}}$

Here we compare the running time of D4GE/S4GE$_{\text{CD}}$ to S4GE on a single machine. For this experiment, for D4GE/S4GE$_{\text{CD}}$ we used a cluster of 120 distributed workers, while for S4GE we used a machine with 24 threads. We set $\rho = 16$ for the distributed runs. Our results are shown in Table 2. For **hollywood09**, **dewiki**, **hollywood11**, and **orkut** using S4GE we abort the runs because they are over our time budget of 6 days. Note that 6 days is 8640 minutes. We notice that workload imbalance has a big impact on the S4GE runtime. Except for **amazon**, we see big speedup for all the datasets. For **amazon**, the runtime is too short and the overhead for the distributed

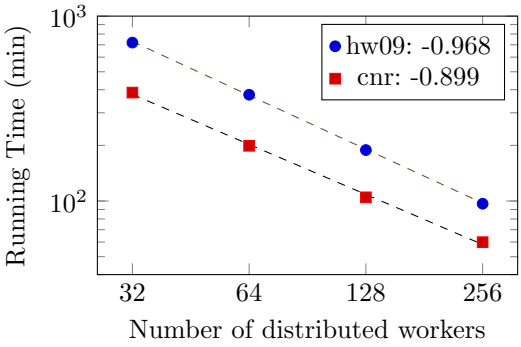

Fig. 6 Machine scalability of D4GE/S4GE on *cnr* and *hollywood09*. D4GE/S4GE show very strong scalability with slope -0.899 and -0.968, which is very close to -1, the perfect value.

computing is larger than the gain. The runtime of dewiki is longer than the others because it has higher maximum degree.

Table 2 The enumeration time (minutes) of D4GE/S4GE$_{CD}$ with $\rho = 16$, 120 workers, against S4GE (single machine) with 24 threads.

Dataset	S4GE	D4GE/S4GE$_{CD}$	Speedup
enron	1.28	0.18	7.1
cnr	2933	132	22.2
amazon	0.23	0.37	0.6
hollywood09	$>$ 6 days	204	/
dewiki	$>$ 6 days	2328	/
hollywood11	$>$ 6 days	864	/
orkut	$>$ 6 days	390	/
ljournal	1367	47	29

6.4 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 $\rho = 16$ and use the same cluster configuration for both. The enumeration times are listed in Table 3. 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 **dewiki**, we can deduce that the speedup is > 3.7 (i.e., 8640/2328). 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 expressions 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$_{\text{CD}}$ compared to PSE/S4GE.

Table 3 The enumeration time (minutes) of D4GE/S4GE$_{\text{CD}}$ against PSE/S4GE, with $\rho = 16$, 120 workers.

Dataset	PSE/S4GE	D4GE/S4GE$_{\text{CD}}$	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	$>$ 6 days	2328	/
hollywood11	9186	864	10.6
orkut	3799	390	9.7
ljournal	432	47	9.2

6.5 PSE/VF2 vs D4GE/CF4$_{\text{CD}}$

Lastly, we compare the performance of Park et al.'s PSE/VF2 implementation on 4-clique query, against our D4GE/CF4$_{\text{CD}}$. The results are shown in Table 4. Comparing our D4GE/CF4$_{\text{CD}}$ suite against one of the *state-of-the-art* subgraph 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**.

Table 4 Enumeration time (minutes) of D4GE/CF4$_{\text{CD}}$ against PSE/VF2, with $\rho = 16$

Dataset	PSE/VF2	CD$_{\text{ext}}$/CF4$_{\text{CD}}$	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

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 5. We can see that PSE partitioning scheme, when combined with S4GE algorithm, emits around 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.

Table 5 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%

Comparing the second and the third column of Table 5 on duplicate emissions, we can see that localized VF2 algorithm emits less 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 **hollywood** and **orkut** datasets.

The overall results show that D4GE/$CF4_{CD}$ 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/$CF4_{CD}$ is tuned to enumerating 4-cliques only whereas VF2 is capable of answering any k-order sub-graph query.

We also want to emphasize that the comparison here is aimed to show the performance gain of D4GE over PSE; while D4GE/$S4GE_{CD}$ can be revised to query 4-cliques, it is designed for a bigger goal - enumerating all 4-node graphlets.

6.6 The Output of D4GE/$S4GE_{CD}$

Here we summarize the results of our experiments with D4GE/$S4GE_{CD}$. We list the counts of graphlets in Tables 6 and 7.

For the largest datasets, **uk02**, **enwiki18** and **indochina**, we employed a larger cluster of 672 workers with 4 GB RAM per worker. On this cluster, we set ρ to 25, which gives us 15,250 sub-problems. The results are shown

Table 6 The outputs of D4GE/S4GE$_{CD}$ with $\rho = 16$, 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
4-cycle	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

Table 7 The outputs of D4GE/S4GE$_{CD}$ with $\rho = 16$, 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
4-cycle	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

in Table 8. 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 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 = 24 hours = 1440 minutes.

Table 8 The outputs of D4GE/S4GE$_{CD}$ with $\rho = 25$, 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
4-cycle	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

6.7 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 [8] it was shown that S4GE algorithm is bounded by $T_{3g} + (|\angle| + |\Delta|)d_{\max}^{\text{sym}}$, 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 3, **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 plot the enumeration time of eight small-medium datasets against $d_{\max}^{\text{sym}}(|\Delta|+|\angle|)$ in Figure 7. From our experiments, the enumeration time demonstrates high correlation with respect to $d_{\max}^{\text{sym}}(|\Delta|+|\angle|)$. 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.

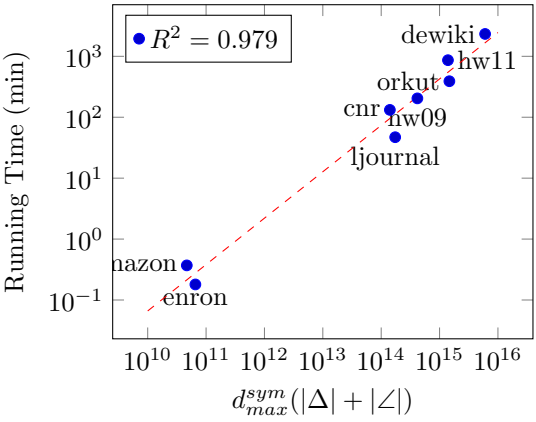

Fig. 7 Strong correlation between the enumeration time and $d_{max}^{sym}(|\Delta|+|\angle|)$ on the small-medium datasets.

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

$$N = M\,I \tag{10}$$

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.

7 Directed Case

So far we have been focusing our study on undirected graphs. However, many real world networks are directed. Let us now turn our attention to graphlets in directed graphs. In a directed graph each edge has a direction. Edges connected to a node u can be classified in two types: edges *to* u and edges *from* u. Accordingly, we have outgoing neighbours of u, $N^+(u)$, and ingoing neighbours of u, $N^-(u)$, for the neighbouring nodes of u. The out-degree of u is $d^+(u) = |N^+(u)|$, and the in-degree of u is $d^-(u) = |N^-(u)|$. Note that $N^+(u)$ and $N^-(u)$ may overlap, because for a pair of nodes u and v we may have both $u \to v$ and $u \leftarrow v$ edges. We call the connection between two nodes a link. There are three types of links between nodes u and v: from u to v, from v to u, and bidirectional. We encode the links by using two binary digits as shown in Table 9.

Table 9 Link encoding using two binary digits.

Link	Value	Binary
○ ○	0	00
○→○	1	01
○←○	2	10
○↔○	3	11

A triad is a subgraph of three nodes in a directed graph [39, 42]. When each pair of the nodes is connected we have a closely connected triad. Here, since we restrict our study to only closely connected triads, we will simply call them as triads. Other authors use the term triangles [43, 44], but we do not want to confuse them with the undirected triangles, or the directed triangles as defined in [21, 45]. There are seven types of triads, as shown in Figure 8. Enumerating triads means listing the edges (or links) as well as the nodes inside every triad. Thus, triad enumeration is more complex than triangle enumeration. Nonetheless, we have shown that it is possible to devise an efficient algorithm that, when combined with a compression framework such as WebGraph [46], is able to enumerate triads on a graph with a billion nodes and billions of edges using a single commodity machine [21].

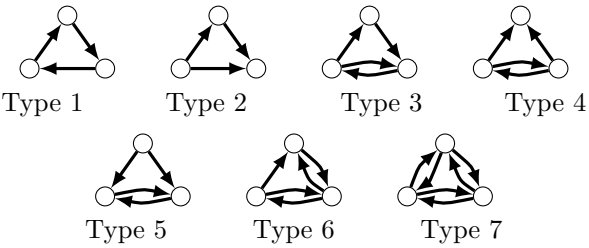

Fig. 8 Seven types of triads.

7.1 Triad Enumeration

Our serial algorithm for triad enumeration is listed in Algorithm 14. The algorithm requires both a directed graph and its transpose graph as input. The transpose of a directed graph $G = (V, E)$ is another directed graph $G^T = (V, E^T)$, where E^T is the same set of edges as E but with each edge is reversed. The key idea here is that the ingoing neighbours of node u in G are the outgoing neighbours of u in G^T, i.e., $N^-(u) \equiv N^-_G(u) = N^+_{G^T}(u)$. Therefore, we can consider only the outgoing adjacency lists from each G and G^T in the computation. That is, we find $N^+(u)$ from G and $N^-(u)$ from G^T. To find the triads, this algorithm employs four pointers, one on each of $N^+(u)$, $N^-(u)$, $N^+(v)$, and $N^-(v)$. Therefore, we call it *Four Pointers Triad Enumeration* (FPTE) algorithm.

The algorithm iterates over the first node u. This iteration can easily be parallelized. For each u, it checks both $N^+(u)$ and $N^-(u)$ to find the neighbours of u and their respective links. For each neighbour of u, v, it finds their common neighbours using four pointers (Line 7). For each common neighbour, w, it looks up the triad type based on the links among the three nodes (u, v, w). The encodings are listed in Table 10. In line 10, `enum()` is a space holder for an enumeration or listing function.

Algorithm 14 FPTE

Require: A directed graph $G = (V, E)$ and its transpose G^T
Ensure: The number of each type of triads in G, Δ_i

```
1: Δ1 ← 0, ..., Δ7 ← 0
2: for all u ∈ V do                                   ▷ Parallelize
3:     while there is next do
4:         Find next neighbour in N+(u) and/or N−(u): v.
5:         Code the link uv as e1: either 01, 10 or 11
6:         while there is next do
7:             Find next common neighbour of u and v: w.
8:             Code the links vw as e2, and wu as e3.
9:             Look up triad type i using e1, e2, e3.
10:            enum(u, v, w, e1, e2, e3)
11:            Δi ← Δi + 1
12:        end while
13:    end while
14: end for
```

Table 10 Triad types and binary encoding.

Triad	Binary Code
Type 1	010101, 101010
Type 2	010110, 011001, 100101, 101001, 100110, 011010
Type 3	010111, 011101, 110101, 101011, 101110, 111010
Type 4	011011, 110110, 101101
Type 5	100111, 111001, 011110
Type 6	011111, 110111, 111101, 101111, 111011, 111110
Type 7	111111

The 4-pointers algorithm is an expansion of the 2-pointers algorithm commonly used for set intersections in the (undirected) triangle enumeration. The flow of the 4-pointers algorithm is illustrated in Figure 9. At the start, each pointer is set to the lowest member of each corresponding set. It checks on the lowest pointer to see if there is another pointer at the same level, and if the member is a common neighbour of u and v. If so, it then computes the link type, and enumerates the triad. It then proceeds by moving the lowest pointer(s) to the next neighbour. Thus, it searches for intersection between $(N^+(u), N^-(u))$ and $(N^+(v), N^-(v))$.

As with triangle enumeration, preprocessing the input graph before the enumeration is crucial in shortening the runtime. In this case, the preprocessing needs to be done simultaneously so that any relabelling would be consistent between the graph and its transpose. To get the greatest benefit, the sorting is based on whichever has the bigger max degree. The pseudocode for this preprocessing is listed in Algorithm 15.

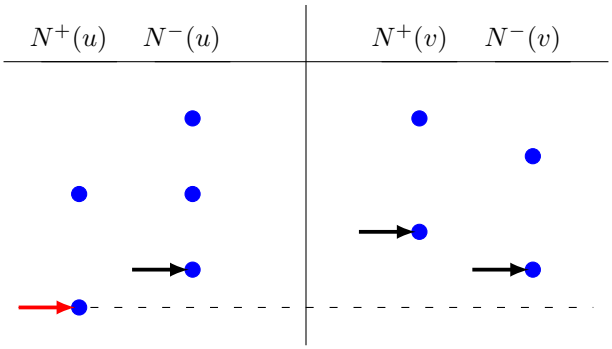

Fig. 9 Four pointers

Algorithm 15 DiGraph-Prep

Require: An directed graph $G(V, E)$ and its transpose $G^T(V, E^T)$

1: Check the maximum out-degrees of G and G^T.
2: Sort V based on the out-degrees of either G or G^T, whichever has the higher maximum out-degree, in ascending order.
3: Relabel the vertices according to their new order.
4: Build adjacency list of the sorted and relabeled vertices.
5: Cut out the smaller out-neighbours from each neighbour list.

7.2 Triad Enumeration on Distributed Platform

While Algorithm 14 (FPTE) is already parallelized (line 2), its scalability is limited to a single machine - shared memory model. In order for the FPTE to enjoy the multi-machine - discrete memory computing clusters with much higher degree of parallelism, we fit FPTE under the duplication-free partition scheme that D4GE proposed earlier. To achieve this, we modified D4GE to handle 3-node sub-graphs, and modified FPTE algorithm to work with directed edgesets.

Because D4GE partitioning scheme operates independently of the serial algorithm, there are only three minor changes required. First, the generated sub-problems and color-assignments are reduced from size 4 down to size 3, so the total number of sub-problems is now $\binom{\rho}{2} + \binom{\rho}{3}$. Second, there is no need for symmetrization. Third, because FPTE operates on both G and G^T, the partitioning are applied to both G and G^T as well, and for each single sub-problem, the directed edgesets of G and G^T are loaded into memory. The modified partitioning scheme is listed as Algorithm 16.

Algorithm 16 D3GE

Require: A directed graph $G(V, E)$ and its transpose $G^T(V, E^T)$; the number of colors ρ

1: Construct $\overrightarrow{G}(V, \overrightarrow{E})$ and $\overrightarrow{G^T}(V, \overrightarrow{E^T})$ by applying edge-orientation to $G(V, E)$ and $G^T(V, E)$
2: Partition $\overrightarrow{E}$ and $\overrightarrow{E^T}$ into directed edge sets E^*_{ij} and E^{T*}_{ij} using ρ
3: Generate ordered color assignments and sub-problems $\{S_{\text{Cs}} \mapsto \{K_{ijkl}\}\}$
4: **for all** S_{Cs}, $\{K_{ijkl}\}$ **do** ▷ Distributed-for
5: $E_{\text{map}}, E^T_{\text{map}} \leftarrow \text{ReadEdgeSets}_{\text{CD}}\ (S_{\text{Cs}}, 3)$
6: **for all** $K_{ijkl} \in \{C_0, C_1, ...\}$ **do**
7: $\text{FPTE}_{\text{CD}}\ (E_{\text{map}}, E^T_{\text{map}}, ijkl)$
8: **end for**
9: **end for**

Modification to FPTE is also minimal. The modification here follows the same fashion as migrating S4GE to S4GE_{CD}: instead of the entire graph G and its transpose G^T, modified FPTE enumerates over directed edgesets E^*_{pq} and E^{T*}_{pq} given a color-assignment ijk. Specifically, given color-assignment ijk, line 2 and 3 of FPTE (Algorithm 14) requires edgesets E^*_{ij} and E^{T*}_{ij}; and line 7 of FPTE requires edgesets E^*_{ik} and E^{T*}_{ik} for knowledge of ($N^+(u)$ and $N^-(u)$), and E^*_{jk} and E^{T*}_{jk} for knowledge of ($N^+(v)$ and $N^-(v)$). The rest of FPTE stays unmodified, as the edgesets solely supply the corresponding neighbourhood information but do not alter the behavior of the algorithm. We call this modified version of FPTE as FPTE_{CD}, and is summarized as Algorithm 17.

7.3 Experiment

For the experiments, we used a Compute Canada cluster with 4 compute nodes, each node with 32 cores and 128 GB RAM per node. This configuration effectively gives us 128 workers with 4 GB RAM per worker. We set ρ to 12, yielding us 286 sub-problems. For comparison, we also ran experiments on a single machine. The configuration of the machine is of dual Intel Xeon E5620 CPUs, for a total of 16 threads, and 64 GB RAM. The datasets are listed in Table 11. These are selected to cover the comparison against the ones already in [21], plus five additional datasets of varying sizes.

The enumeration times are listed in Table 12. The last five graphs were not listed in the FPTE paper. Even for the largest graph, **twitter**, with 42M vertices and 1.5B edges, D3GE/FPTE_{CD} is able to enumerate all seven types of triads within 8 minutes, delivering a very strong performance. With eight times the parallelism, compared against the single machine FPTE, for **cnr**, **ljournal**, **uk05**, **dewiki**, **enwiki18** and **uk02**, the speedups are less than 4 fold. While the performance is still improved, these low speedups do not meet the expectation. This is because the original FPTE, while limited on a single

Algorithm 17 FPTE_{CD}

Require: Two mappings from the colors of directed edge set to the edge set $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$ and $E^T_{\text{map}} \equiv \{(i,j) \mapsto E^{T*}_{ij}\}$; ordered color assignment $ijkl$

Ensure: The number of each type of triads in E_{map} and E^T_{map}, Δ_i.

$\Delta_1 \leftarrow 0, \ldots, \Delta_7 \leftarrow 0$
$E^*_{ij} \equiv E_{\text{map}}[ij]$, $E^*_{ik} \equiv E_{\text{map}}[ik]$, $E^*_{jk} \equiv E_{\text{map}}[jk]$
$E^{T*}_{ij} \equiv E^T_{\text{map}}[ij]$, $E^{T*}_{ik} \equiv E^T_{\text{map}}[ik]$, $E^{T*}_{jk} \equiv E^T_{\text{map}}[jk]$
for all $u \in E^*_{ij} \cup E^{T*}_{ij}$ **do**
    while there is next **do**
        $N^+(u) \equiv E^*_{ij}[u]$, $N^-(u) \equiv E^{T*}_{ij}[u]$.
        Find next neighbour in $N^+(u)$ and/or $N^-(u)$: v.
        Code the link uv as $e1$: either 01, 10 or 11
        while there is next **do**
            $N^+(u) \equiv E^*_{ik}[u]$, $N^-(u) \equiv E^{T*}_{ik}[u]$.
            $N^+(v) \equiv E^*_{jk}[v]$, $N^-(v) \equiv E^{T*}_{jk}[v]$.
            Find next common neighbour of u and v: w, in $(N^+(u), N^-(u))$ and $(N^+(v), N^-(v))$.
            Code the links vw as $e2$, and wu as $e3$.
            Look up triad type i using $e1, e2, e3$.
            $\texttt{enum}(u, v, w, e1, e2, e3)$
            $\Delta_i \leftarrow \Delta_i + 1$
        end while
    end while
end for

machine, has the advantage of the shared-memory model, which makes the computation efficient - no partitioning is required. D3GE exposes FPTE to a cluster of workers, and this bears a cost. Because of the discrete-memory model of the clusters, we have to pre-partition the input graph into **overlapping** and independent sub-graphs (sub-problems) and let the workers solve each of the sub-problems. The overlapping of the sub-graphs is necessary because in the discrete-memory model, the workers cannot access each other's memory content. In other words, if D3GE/FPTE_{CD} and FPTE are given the same number of workers/threads, D3GE/FPTE_{CD} inherently does more work per worker, due to the overlap. Additionally, the overlapping portion grows with respect to ρ, further discounting the distributed solution as compared to the shared-memory model. However we would like to stress that this problem is not particular to D3GE/FPTE_{CD}. All known partition schemes suffer from the inevitable overlap. Note that this comparison here only shows the performance improvement over the single machine, not the scalability. The true scalability of D3GE will be discussed later.

On the other hand, D3GE/FPTE_{CD} is able to achieve a 27.4 speedup on the **arabic** dataset, a 161.3 speedup on the **indochina** dataset, and a 116.7

Table 11 The numbers of vertices n, edges m, maximum degree of the original graph $d_{\max}$ and its transpose $d^T_{\max}$, and the effective maximum degrees after the preprocessing, $d^{\text{eff}}_{\max}$ and $d^{T\text{eff}}_{\max}$, of the graph datasets.

Dataset	n	m	$d_{\max}$	$d^T_{\max}$	$d^{\text{eff}}_{\max}$	$d^{T\text{eff}}_{\max}$
cnr	326K	3.2M	2,716	18,235	1,336	81
dewiki	1.5M	36M	5,032	117,908	5,032	409
ljournal	5.4M	79M	2,469	19,409	1,257	397
enwiki18	5.6M	128M	7,948	247,628	7,620	311
indochina	7.4M	194M	6,985	256,425	6,870	6,821
uk02	18.5M	298M	2,450	194,942	2,288	942
arabic	22.7M	640M	9,905	575,618	6,646	3,126
uk05	39.5M	936M	5,213	1,776,852	5,213	584
twitter	41.7M	1.5B	2,997,469	770,155	2,896	5,745

Table 12 The enumeration time (seconds) of D3GE/FPTE$_{\text{CD}}$ with $\rho = 12$ and 128 workers, against original FPTE with 16 threads on a single machine.

Dataset	FPTE	D3GE/FPTE$_{\text{CD}}$	Speedup
cnr	3.0	2.2	1.36
ljournal	81	24.4	3.3
arabic	2961	107.9	27.4
uk05	796	207.1	3.8
dewiki	30.3	14.0	2.16
enwiki18	136.4	34.5	3.95
indochina	9,228.8	57.2	161.3
uk02	191.5	92.7	2.07
twitter	51,984.0	445.6	116.7

speedup on the **twitter** dataset. However, this is not because D3GE/FPTE$_{\text{CD}}$ works much faster on these datasets, but rather it reflects the poor performance of FPTE. Upon inspecting the datasets in Table 11, we can see that these three datasets have relatively large maximum effective degrees. FPTE suffers from having to do all the work for the node with highest effective degree on a single thread. This is similar to 'the curse of last reducer problem' as pointed out by Suri and Vassilvitskii [29]. D3GE/FPTE$_{\text{CD}}$ avoids this problem through a partitioning scheme that leads to a better work-load balance.

Next, we experimented on the scalability of D3GE/FPTE$_{\text{CD}}$ by plotting the enumeration time over the **arabic** and **uk02** datasets, using varying number of distributed workers from 32 to 256. The results are shown in Figure 10. Similar to Figure 6, FPTE$_{\text{CD}}$ fitted under D3GE scales almost perfectly with respect to the degree of parallism: the slopes are -0.894 and -0.866 respectively. This means that every time the number of the distributed workers doubles, the

enumeration time is reduced by factors of $2^{0.894} = 1.858$ and $2^{0.866} = 1.822$ respectively. We re-confirm that the scalability of our proposed distributed partitioning scheme is on-par with the SotA Map-Reduce based ones [12] and [17].

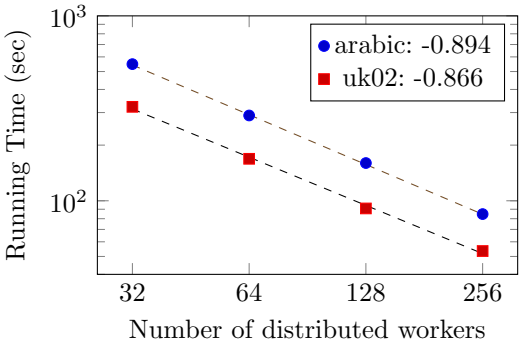

Fig. 10 Scalability of D3GE/FPTE$_{CD}$ on *uk02* and *arabic*. D3GE/FPTE$_{CD}$ again presents very strong scalability with slope -0.866 and -0.894.

7.4 Extension to Four Node

Can we generalize S4GE to the directed case? That is, can we build a solution to enumerate four node directed graphlets? First, let us see how these graphlets look like. The underlying graphs for these graphlets would be the same as the ones shown in Figure 3. It is just that now each edge is replaced by a directed link with three possible types. Also, we need to check for any isomorphisms to get the number of distinct types. Take the tailed-triangle for example. We already know that there are seven types of triads. Now, there are three possible tails, and therefore there are 21 distinct types of directed tailed triangles.

A similar approach to S4GE can be done here. That is, we can first compute the three node directed graphlets, and then proceed by looking for any four node directed graphlets that are attached to them. In this case, we would also need to compute the directed wedges as well. The biggest challenge would be the explosion of the number of types.

8 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 computations, which are unavoidable with other schemes. We show that, D4GE when combined with S4GE, requires at most $2\,m^{\text{sym}}$ 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.

We have also presented a distributed platform solution for enumerating three node directed graphlets, or triads, the D3GE. We empirically showed that it has a very good scalability and much better performance, as compared to the single machine solution, on graphs with high maximum degrees.

9 Acknowledgments

We thank the anonymous reviewers for their detailed comments that helped us improve the presentation of this work substantially.

References

[1] Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J.: Mining coherent dense subgraphs across massive biological networks for functional discovery. Bioinformatics **21**(suppl_1), 213–221 (2005)

[2] Milenković, T., Pržulj, N.: Uncovering biological network function via graphlet degree signatures. Cancer informatics **6** (2008)

[3] Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. IEEE Transactions on Knowledge and Data Engineering **17**(8), 1036–1050 (2005)

[4] Ralaivola, L., Swamidass, S.J., Saigo, H., Baldi, P.: Graph kernels for chemical informatics. Neural networks **18**(8), 1093–1110 (2005)

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

[6] Bröcheler, M., Pugliese, A., Subrahmanian, V.S.: Cosi: Cloud oriented subgraph identification in massive social networks. In: 2010 International Conference on Advances in Social Networks Analysis and Mining, pp. 248–255 (2010). IEEE

[7] Wong, S.W., Cercone, N., Jurisica, I.: Comparative network analysis via differential graphlet communities. Proteomics **15**(2-3), 608–617 (2015)

[8] Santoso, Y., Srinivasan, V., Thomo, A.: Efficient enumeration of four node graphlets at trillion-scale. In: 23rd EDBT, pp. 439–442 (2020)

[9] Pinar, A., Seshadhri, C., Vishal, V.: Escape: Efficiently counting all 5-vertex subgraphs. In: Proceedings of the 26th International Conference on

World Wide Web, pp. 1431–1440 (2017). International World Wide Web Conferences Steering Committee

[10] Newman, M.E.: The structure and function of complex networks. SIAM review **45**(2), 167–256 (2003)

[11] Wang, J., Cheng, J.: Truss decomposition in massive networks. Proceedings of the VLDB Endowment **5**(9) (2012)

[12] Park, H.-M., Myaeng, S.-H., Kang, U.: Pte: Enumerating trillion triangles on distributed systems. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1115–1124 (2016)

[13] Hočevar, T., Demšar, J.: A combinatorial approach to graphlet counting. Bioinformatics **30**(4), 559–565 (2014)

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

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

[16] McSherry, F., Isard, M., Murray, D.G.: Scalability! but at what {COST}? In: 15th Workshop on Hot Topics in Operating Systems (HotOS {XV}) (2015)

[17] Park, H.-M., Silvestri, F., Pagh, R., Chung, C.-W., Myaeng, S.-H., Kang, U.: Enumerating trillion subgraphs on distributed systems. ACM Transactions on Knowledge Discovery from Data (TKDD) **12**(6), 1–30 (2018)

[18] Batagelj, V., Zaveršnik, M.: Short cycle connectivity. Discrete Mathematics **307**(3-5), 310–318 (2007)

[19] Tabak, B.M., Takami, M., Rocha, J.M., Cajueiro, D.O., Souza, S.R.: Directed clustering coefficient as a measure of systemic risk in complex banking networks. Physica A: Statistical Mechanics and its Applications **394**, 211–216 (2014)

[20] Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications vol. 8. Cambridge University Press, ??? (1994)

[21] Santoso, Y., Srinivasan, V., Thomo, A., Chester, S.: Triad enumeration at trillion-scale using a single commodity machine. In: 22nd EDBT (2019)

[22] Schank, T., Wagner, D.: 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, pp. 606–609 (2005). https://doi.org/10.1007/11427186_54. https://doi.org/10.1007/11427186_54

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

[24] Ahmed, N.K., Neville, J., Rossi, R.A., Duffield, N.: Efficient graphlet counting for large networks. In: 2015 IEEE International Conference on Data Mining, pp. 1–10 (2015). IEEE

[25] Bressan, M., Chierichetti, F., Kumar, R., Leucci, S., Panconesi, A.: Counting graphlets: Space vs time. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp. 557–566 (2017). ACM

[26] Wernicke, S., Rasche, F.: Fanmod: a tool for fast network motif detection. Bioinformatics **22**(9), 1152–1153 (2006)

[27] Marcus, D., Shavitt, Y.: Rage–a rapid graphlet enumerator for large networks. Computer Networks **56**(2), 810–819 (2012)

[28] Danisch, M., Balalau, O., Sozio, M.: Listing k-cliques in sparse real-world graphs. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, pp. 589–598 (2018). International World Wide Web Conferences Steering Committee

[29] Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: Proceedings of the 20th International Conference on World Wide Web. WWW '11, pp. 607–614. ACM, New York, NY, USA (2011). https://doi.org/10.1145/1963405.1963491

[30] Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. IEEE transactions on pattern analysis and machine intelligence **26**(10), 1367–1372 (2004)

[31] Teixeira, C.H., Fonseca, A.J., Serafini, M., Siganos, G., Zaki, M.J., Aboulnaga, A.: Arabesque: a system for distributed graph mining. In: Proceedings of the 25th Symposium on Operating Systems Principles, pp. 425–440 (2015). ACM

[32] Dias, V., Teixeira, C.H., Guedes, D., Meira, W., Parthasarathy, S.: Fractal: A general-purpose graph pattern mining system. In: Proceedings of the 2019 International Conference on Management of Data, pp. 1357–1374

(2019)

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

[34] Mawhirter, D., Reinehr, S., Holmes, C., Liu, T., Wu, B.: Graphzero: Breaking symmetry for efficient graph mining. arXiv preprint arXiv:1911.12877 (2019)

[35] Chen, H., Liu, M., Zhao, Y., Yan, X., Yan, D., Cheng, J.: G-miner: an efficient task-oriented graph mining system. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–12 (2018)

[36] Yan, D., Guo, G., Chowdhury, M.M.R., Özsu, M.T., Ku, W.-S., Lui, J.C.: G-thinker: A distributed framework for mining subgraphs in a big graph. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1369–1380 (2020). IEEE

[37] Ren, X., Wang, J., Han, W.-S., Yu, J.X.: Fast and robust distributed subgraph enumeration. arXiv preprint arXiv:1901.07747 (2019)

[38] Zhang, H., Yu, J.X., Zhang, Y., Zhao, K., Cheng, H.: Distributed subgraph counting: a general approach. Proceedings of the VLDB Endowment **13**(12), 2493–2507 (2020)

[39] Batagelj, V., Mrvar, A.: A subquadratic triad census algorithm for large sparse networks with small maximum degree. Social networks **23**(3), 237–243 (2001)

[40] Chin Jr, G., Marquez, A., Choudhury, S., Feo, J.: Scalable triadic analysis of large-scale graphs: Multi-core vs. multi-processor vs. multi-threaded shared memory architectures. In: Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium On, pp. 163–170 (2012). IEEE

[41] Parimalarangan, S., Slota, G.M., Madduri, K.: Fast parallel graph triad census and triangle counting on shared-memory platforms. In: Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International, pp. 1500–1509 (2017). IEEE

[42] Davis, J.A., Leinhardt, S.: The structure of positive interpersonal relations in small groups. Sociological Theories in Progress **2**, 218–251 (1972)

[43] Seshadhri, C., Pinar, A., Kolda, T.G.: Fast triangle counting through wedge sampling. In: Proceedings of the SIAM Conference on Data Mining, vol. 4, p. 5 (2013)

[44] Wang, P., Qi, Y., Sun, Y., Zhang, X., Tao, J., Guan, X.: Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. Proceedings of the VLDB Endowment **11**(2), 162–175 (2017)

[45] Santoso, Y.: Triangle counting and listing in directed and undirected graphs using single machines. Master's thesis, University of Victoria (2018)

[46] Boldi, P., Vigna, S.: The WebGraph framework I: Compression techniques. In: Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595–601. ACM Press, Manhattan, USA (2004)