Parallel Pattern Enumeration in Large Graphs

Abir Farouzi[1,3], Xiantian Zhou[2], Ladjel Bellatreche[1], Mimoun Malki[3], and Carlos Ordonez[2]

[1] LIAS/ISAE-ENSMA, France
[2] University of Houston, USA
[3] Ecole Nationale Supérieure en Informatique de Sidi Bel Abbès, Algeria

Abstract. Graphlet enumeration is a fundamental task in graph exploration and analysis. It has many real-life applications including biology and Chemistry. In this paper, we presents a novel approach to extracting these patterns with queries, in a distributed fashion. Our solution underlies an efficient partitioning strategy based on vertex coloring, that guarantees perfect load balancing and an exact graphlet enumeration. To the best of our knowledge, this is the first paper to provide a short and abstract solution with queries to enumerate both 3-vertex and 4-vertex patterns at a billion scales in a reasonable time.

1 Introduction

In graph analytics, finding small structures is crucial to studying the relationships between a set of individuals/objects. These structures are called graphlets or patterns, which are defined as small induced subgraphs. The problem of graphlet enumeration is involved in many fields. In biology for example, [13] studied the interaction and the function of proteins in the entire proteome, which is based on protein-protein interaction (PPI) natworks. For that, [13] developed a graph-based technique that condenses a protein's nearby topology within a PPI network by using a vector of graphlet degrees known as the protein's signature. It then determines the similarity between the signatures of all pairs of proteins. Another relevant example would be in chemistry, where [6] have studied the chemical compounds classification by developing a method based on two steps: (1) sub-structure discovery process which includes the graphlet enumeration, and (2) the classification process allowing an intelligent chemical compounds classification using the graphlets. Other applications can be found in [4,5].

On the other hand, it is well-known that the graphlets beyond three vertices are complicated to enumerate since *the number of instances* can rapidly grow with $O(n^{\alpha})$, where α is the order of the graphlet. Indeed, we enumerate 2 structures for the 3-vertex graphlets, 6 structures for the 4-vertex graphlets, 21 for 5-vertex graphlets, and so on. In practice, it is typical for both the list of graphlets and the time required for their enumeration to increase quickly as the size of the graph increases, resulting in a computationally expensive task [11].

In this context, we presented in a previous work [7] a distributed solution with queries, to enumerate efficiently all the embedded triangles in large graphs.

The triangles constitute a special case of graphlets, where $\alpha = 3$. However, it is not as hard as larger graphlets, particularly those of 4 vertices. Indeed, enumerating 4-vertex graphlets is more challenging, since it requires checking and outputting 6 instances with different structures. Our focus in this work is to present an efficient parallel solution to enumerate all the 4-vertex graphlets. Our solution underlines an efficient partitioning strategy based on vertex-coloring and inspired by our previous work [7] on triangles. Furthermore, 4-vertex graphlet enumeration requires more join operations compared to the triangles. Thus, in the present work, we developed a staged enumeration strategy, where the simple graphlets of 3 and 4 order are used to reveal larger or complex subgraphs dynamically. This reduce memory usage and accelerate the running time. To the best of our knowledge, this is the first paper that presents an integrated solution to discover both 3-vertex and 4-vertex graphlets (graph patterns) with queries.

Our contributions are :

(1) We provide a distributed algorithm for staged graphlet enumeration. Our approach can be implemented on any parallel system including DBMS.
(2) We propose a vertex cut partitioning strategy based on coloring that guarantees a perfect load balancing, and a local enumeration.
(3) We present the computational model optimization for an efficient and optimized execution of our solution.
(4) We study the partitioning strategy, the result correctness, and the isomorphism between the resulting graphlets.
(6) We finally present an experimental validation of our findings and compare our results with a competing graphlet enumeration solution.

Our paper is organized as follows. Preliminaries are described in Section 2. In Section 3, we explain our data partitioning strategy, the computational model optimization, and our solution for graphlet enumeration. In Section 4, we study the partitioning strategy, the graph isomorphism, and the load balancing. We present an experimental validation in Section 5. Section 6 explains closely related work and Section 7 concludes the paper with general remarks.

2 Preliminaries

2.1 Graph

Let $G = (V, E)$ be an undirected unweighted graph with $n = |V|$ (V is the set of vertices) and $m = |E|$ (E is the set of edges). For our algorithm, we read the input graph G as an edge list $E(u, v)$, where an edge goes from u to v. We define $H = (V_H, E_H)$ as a subgraph of $G = (V, E)$ if $V_H \subseteq V$ and $E_H \subseteq E$, and as an induced subgraph of $G = (V, E)$ if $\forall u, v \in V_H$ and $(u, v) \in E_H$ then $(u, v) \in E$.

2.2 Graphlets (Patterns in Connected Subgraphs)

A graphlet H is a connected induced subgraph of G. We denote two types of 3-vertex graphlets: (1) Wedges (W) that are paths of two edges, and (2) Triangles

(TR) which are cycles with three connected edges. Moreover, there are six types of 4-vertex graphets: (3) 3-Path (P), (4) 3-Star (S), (5) Rectangle (R), (6) Tailed Triangle (T), (7) Diamond (D), (8) 4-vertex Clique (C) (see Fig. 1).

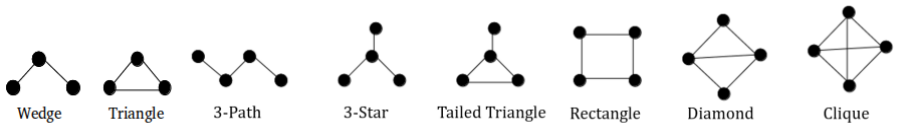


Fig. 1: 3 and 4-vertex graphlets.

2.3 Computational model

In order to run our algorithm, we use the $k-machine$ model (a.k.a *Big Data* model), introduced by [9]. It consists of a set of $k \geq 2$ machines built on a shared-nothing architecture. Each machine can communicate directly with the other machines via message passing (no shared memory) while running an instance of the distributed algorithm. Since there is no shared memory, an efficient data partitioning strategy is mandatory. It aims to minimize the data communication between machine during the distributed algorithm execution. Note that the data communication is a time and space consuming task.

3 Our Approach for Pattern Enumeration

3.1 Data partitioning

The key idea underlying our partitioning approach is to perform a vertex-cut partitioning based on coloring, so that each vertex and its incident edges are sent to the same machine (see Fig. 2). Our partitioning strategy aims to partition the vertex set V into c subsets of $O(n/c)$ vertex each, where c is the number of color subsets ($c \geq 2$). Then, each edge between two subsets of colors is sent to one random machine from the $k-machine$ model called the proxy machine. Finally, each local machine collects its required edges according to its quadruplet of colors (hard-coded in the algorithm) to perform locally the graphlet enumeration.

In practice, we create table $V\_s(u, u\_color)$ to store each vertex color. Indeed, each entry in the table V_s is a couple of a vertex and its color chosen uniformly and independently at random from the c colors. Then, the table $E\_s\_proxy(u, v, u\_color, v\_color)$ is created to send edges to proxies. It holds the end-vertices color of each edge. Finally, the small table $quadruplet(machine, c_1, c_2, c_3, c_4)$ is created and replicated on each machine of the $k-machine$. This small table is used by each local machine to collect its required edges and stores them in the table $E\_s\_local(machine, u, v, u\_color, v\_color)$. The edges stored

in E_s_local table depends on the type of graphlets to generate. Hence, we need to recreate this table according to the required edges to form the graphlets. Notice that each entry in the table *quadruplet* consists of one of the possible permutation of the c colors. Suppose we have two colors: black (b) and white (w). We can generate (b,b,b,b), (b,b,b,w), $\cdots$ etc. We can generate c^4 quadruplets using these c colors.

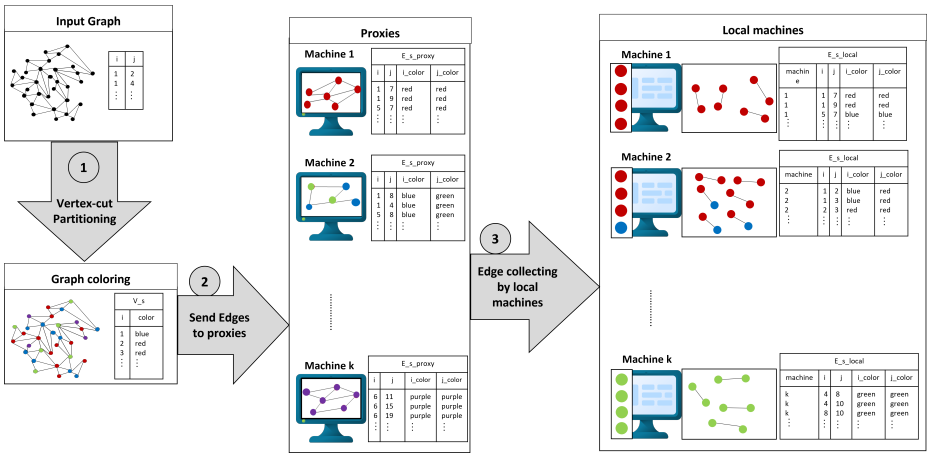


Fig. 2: Data partitioning for graphlet enumeration.

3.2 $k-machine$ distributed model optimization

The ideal execution of our algorithm requires one machine for each color quadruplet. This means we need a $k-machine$ model of size:

$$k = c^4 \tag{1}$$

However, with larger number of c, the size of the $k-machine$ model can grow rapidly. Hence, we propose $k-machine$ model optimization without impacting our algorithm functions. For that, we need to allow each machine to manage more than one quadruplet. Generally, we can use the following equation:

$$k = c^l \quad where \quad 1 \leq l \leq 4 \tag{2}$$

Here, each machine needs to manage c^{4-l} quadruplets of colors. For example, with $c = 2$ and $k = 8$, each machine manages 2 color quadruplets.

3.3 Graphlet enumeration

Enumerating graphlets of order $O(n^\alpha)$ results in many instance. Suppose we have a graph with n vertices, enumerating all its embedded graphlets requires

studying and checking $C_n^\alpha = \frac{n!}{\alpha!*(n-\alpha)!}$ possible combinations. Hence, we need to generate 6 instances of 4-vertex graphlets. Moreover, each graphlet of order 4 outputs $4! = 24$ permutations. These permutations represent repetitions, and their elimination is a hard task in most cases. Thus, only some configurations of the graphlets must be considered, and a redundancy elimination process need to be integrated into our algorithm. Fig. 4 summarizes the graphlets configuration to consider for each 4-vertex graphlet type. Note that each graphlet is made of 4 vertices $\{u, v, w, z\}$ where $u < v < w < z$.

In order to accelerate the graphlet enumeration, we allow an enumeration by stage (see Fig. 3). Thus, we classify the 4-vertex graphlets into two categories:

(1) Intuitive graphlets: it consists of three graphlets: 3-Path, 3-Star, and rectangle; that we can reuse to generate the other graphlets.
(2) Derived graphlets: represented by the complex graphlets that can be derived from the intuitive graphlets: tailed triangle, diamond, and clique.

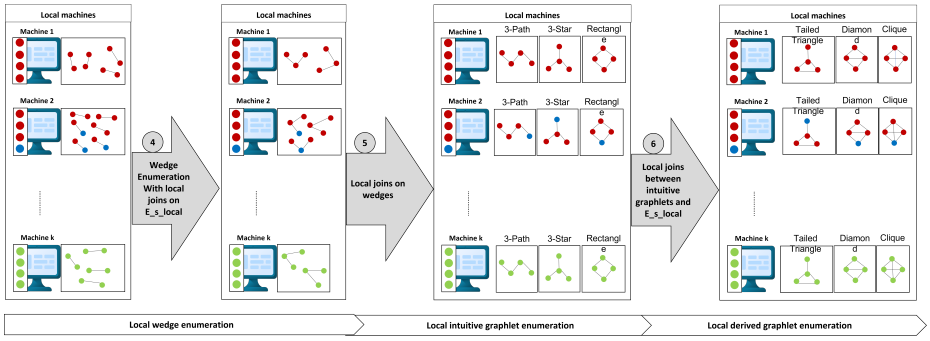


Fig. 3: Graphlet enumeration by stage.

Wedge enumeration The main idea behind our approach is to enumerate graphlets by stage. Thus, we enumerate wedges to output intuitive graphlets, that will be used to list derived graphlets. Wedges are graphlets consisting of 3 vertices and 2 edges. We compute them using one self join on the table E_s_local. We enumerate four types of wedges according to color quadruplets. Precisely, the wedges of type 1 ($Wedge\_T1$) correspond to the first three colors ($\{c_1, c_2, c_3\}$) in the table *quadruplet*. This wedge is involved in the enumeration of all the intuitive graphlets. Then, we enumerate wedges of type 2 ($Wedge\_T2$) with ($\{c_2, c_3, c_4\}$) colors to output paths, and wedges of type 3 ($Wedge\_T3$) with ($\{c_1, c_2, c_4\}$) colors to list 3-stars. Finally, we output wedges of type 4 ($Wedge\_T4$) with ($\{c_1, c_3, c_4\}$) colors to enumerate rectangles. For example and without loss of generality, on machine M_1 with (c_1, c_2, c_3, c_4) as quadruplet of colors, we generate $\{Wedge\_T1, Wedge\_T2, Wedge\_T3, Wedge\_T4\}$

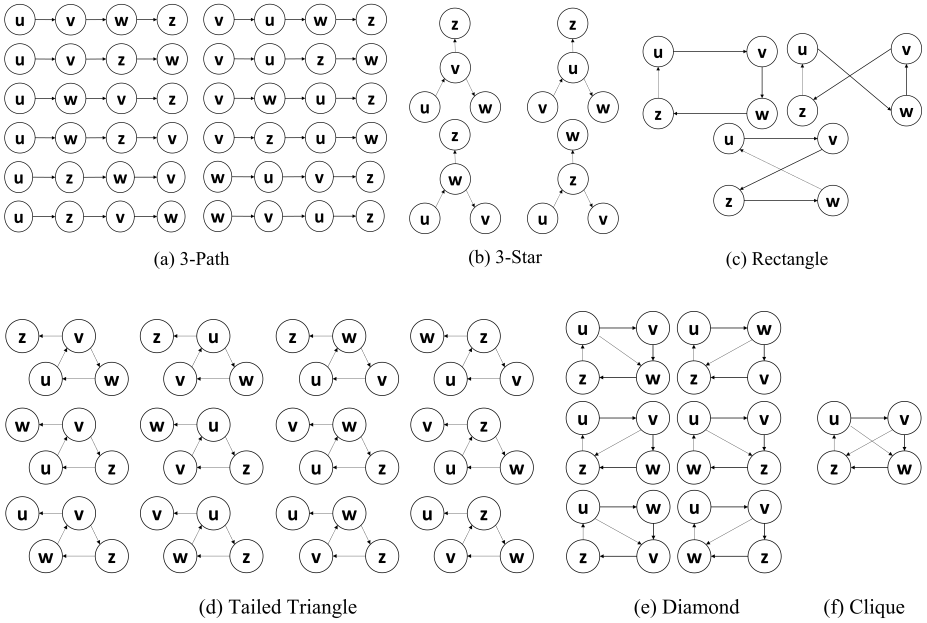


Fig. 4: Graphlets configuration to output.

whose vertex colors are in $\{(c_1, c_2, c_3), (c_2, c_3, c_4), (c_1, c_2, c_4), (c_1, c_3, c_4)\}$ respectively. We create the following tables to store the generated wedges for each machine.

$Wedge\_T1(machine, u, v, w, u\_color, v\_color, w\_color)$
$Wedge\_T2(machine, u, v, w, u\_color, v\_color, w\_color)$
$Wedge\_T3(machine, u, v, w, u\_color, v\_color, w\_color)$
$Wedge\_T4(machine, u, v, w, u\_color, v\_color, w\_color)$

Furthermore, triangles can easily be extracted using the wedges of type 1. Indeed, triangles are a specific case of wedges with an edge connecting each couple of vertices of the wedge (3-vertex clique). The triangle enumeration problem is largely discussed in our previous paper [7].

Intuitive 4-vertex graphlet enumeration Wedges are the building blocks of intuitive graphlets, wherein we generate the required wedges to enumerate each variant of these graphlets. Subsequently, we carry out joins between them. Fig. 5 summarizes this process.

3-Path consist of two wedges (u, v, w) and (v, w, z). It mainly depends on its end-vertices order, in such a way that $u < z$. Hence, we enumerate $\binom{4}{2} = 12$ permutations (by symmetry elimination). For that, we create table $Path(machine, u, v, w, z)$ that holds the 3-Paths resulting of a local join between the table $Wedge\_T1$ and the table $Wedges\_T2$. The query $Q(P)$ summarizes this computation:

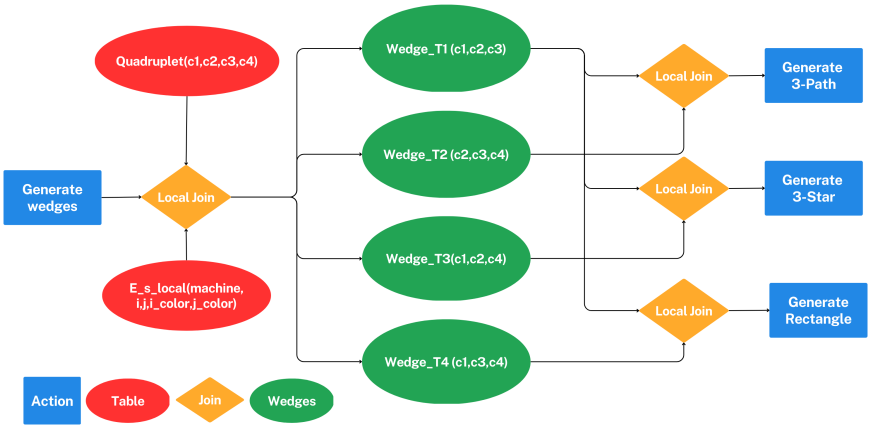


Fig. 5: Intuitive 4-vertex graphlet enumeration process.

```
Q(P):INSERT INTO Path SELECT T1.machine as machine, T1.u as u, T1.v as v,
    T1.w as w, T2.w as z
        FROM Wedge_T1 T1 JOIN Wedge_T2 T2
            ON T1.machine=T2.machine AND T1.v=T2.u AND T1.w=T2.v
        WHERE T1.u<T2.w;
```

3-Star is formed by two wedges (u, v, w) and (u, v, z) connected with the common edge (u, v) of the wedges. The generation of this graphlet depends on the outer vertices of the star. Hence, we need to enumerate $\binom{4}{3} = 4$ permutations. To save the results, we create table $Star(machine, u, v, w, z)$. Then, we join the table $Wedge\_T1$ with table $Wedge\_T3$ considering $u < w < z$ to remove all the repetitions. The query $Q(S)$ bellow summarises the 3-Star graphlet enumeration:

```
Q(S):INSERT INTO Star SELECT T1.machine as machine, T1.u as u, T1.v as v,
    T1.w as w, T3.w as z
        FROM Wedge_T1 T1 Join Wedge_T3 T3
            ON T1.machine=T3.machine AND T1.u=T3.u AND T1.v=T3.v
        WHERE T1.u<T1.w AND T1.w<T3.w;
```

Rectangle is represented by two opposite wedges $\{(u, v, w), (u, z, w)\}$ with $v! = z$. For this graphlet, we can have 4 configurations for the cyclic symmetry and 2 configurations for each clockwise counter-clockwise symmetry. As a result, we need to generate one configuration of each of them, so we generate $\frac{24}{4\times2} = 3$ permutations. Hence, we create the table $Rectangle(machine, u, v, w, z)$ and we perform a local join between the table $Wedge\_T1$ and the table $Wedge\_T4$ for each case ($u < v < w < z$, $u < v < z < w$ and $u < w < v < z$) and we union the results together, as presented in the query $Q(R)$:

```
Q(R):INSERT INTO Rectangle SELECT T1.machine as machine, T1.u as u,
     T1.v as v, T1.w as w, T4.v as z
        FROM Wedge_T1 T1 join Wedge_T4 T4
```

```
            ON T1.machine=T4.machine AND T1.u=T4.w AND T1.w=T4.u
        WHERE T1.u<T1.v AND T1.v<T1.w AND T1.w<T4.v
    UNION SELECT T1.machine as machine, T1.u as u, T1.v as v, T1.w as w,
    T4.v as z
        FROM Wedge_T1 T1 join Wedge_T4 T4
            ON T1.machine=T4.machine AND T1.u=T4.w AND T1.w=T4.u
        WHERE T1.u<T1.v AND T1.v<T4.v AND T4.v<T1.w
    UNION SELECT T1.machine as machine, T1.u as u, T1.v as v, T1.w as w,
    T4.v as z
        FROM Wedge_T1 T1 join Wedge_T4 T4
            ON T1.machine=T4.machine AND T1.u=T4.w AND T1.w=T4.u
        WHERE T1.u<T1.w AND T1.w<T1.v AND T1.v<T4.v;
```

Derived 4-vertex graphlet enumeration The graphlets of this class are generated using the intuitive graphlets class.

Tailed Triangle can be seen as a 3-Star (u, v, w, z) with v at the center, and an edge (u, w), (w, z) or (u, z) connecting two of its endpoints. Its enumeration relies on the vertex at the center and the vertex at the end of its tail, for that we consider $\binom{4}{2} = 12$ permutations. To list them, we create table $Tailed(type, machine, u, v, w, z)$, where *type* distinguish whether the connecting edge is between $\{u, w\}$, $\{u, z\}$ or $\{w, z\}$. Than we recreate table E_s_local to have, on each machine all the edges that endpoints are of colors (c_1, c_3), (c_1, c_4), or (c_3, c_4) of each color quadruplet. Finally, we perform a local join between the table *Star* and the table E_s_local, as mentioned in the query $Q(T)$:

```
/*1=T(u,v,w),2=T(u,v,z),3=T(v,w,z), T=Triangle*/
Q(T):INSERT INTO Tailed SELECT 1, E1.machine as machine, u, v, w, z
        FROM Star E1 Join E_s_local E2
            ON E1.machine=E2.machine AND E1.u=E2.i AND E1.w=E2.j
    UNION SELECT 2, E1.machine as machine, u, v, w, z
        FROM Star E1 Join E_s_local E2
            ON E1.machine=E2.machine AND E1.u=E2.i AND E1.z=E2.j
    UNION SELECT 3, E1.machine as machine, u, v, w, z
        FROM Star E1 Join E_s_local E2
            ON E1.machine=E2.machine AND E1.w=E2.i AND E1.z=E2.j;
```

Diamond can be recognized as a rectangle with an edge on one of its diagonals. Since only 3 configurations are needed for the rectangles, each rectangle with one of it diagonal can be output as diamond. So, we need to output $3 \times 2 = 6$ permutations for this graphlet. To enumerate them, we create table $Diamond(machine, u, v, w, z)$. Then, we recreate the table E_s_local to have edges whose end-vertices are of colors (c_1, c_3) and (c_2, c_4) of each color quadruplet. We finally join locally the table *Rectangle* with the table E_s_local according to the query $Q(D)$:

```
Q(D):INSERT INTO Diamond SELECT E1.machine as machine, u, v, w, z
        FROM Rectangle E1 Join E_s_local E2
            ON E1.machine=E2.machine AND E1.u=E2.i AND E1.w=E2.j
    UNION SELECT E1.machine as machine, u, v, w, z
        FROM Rectangle E1 Join E_s_local E2
            ON E1.machine=E2.machine AND E1.v=E2.i AND E1.z=E2.j;
```

Clique is a complete subgraph of four vertices with an edge between each couple of vertices, hence only one configuration should be output, which is made of the lexicographical order between the vertices. This graphlet is a rectangle with its both diagonals. Hence, we create table $Clique(machine, u, v, w, z)$ to hold all the 4-vertex cliques. For that, we recreate the table E_s_local to have the edges whose end-vertices are of colors (c_1, c_3) ans (c_2, c_4) of each color quadruplet. Then, we join locally the table *Rectangle* with the table E_s_local twice, as presented in the following query $(Q(C))$:

```
Q(C):INSERT INTO Clique SELECT E1.machine as machine, u, v, w, z
        FROM Rectangle E1 JOIN E_s_local E2
            ON E1.machine=E2.machine AND E1.u=E2.i AND E1.w=E2.j
        JOIN E_s_local E3
            ON E1.machine=E3.machine AND E1.v=E3.i AND E1.z=E3.j
        WHERE E1.u<E1.v AND E1.v<E1.w AND E1.w<E1.z;
```

4 Graphlet enumeration theoretical analysis

4.1 Partitioning strategy effectiveness

Our partitioning strategy aims to send an induced subgraph to each machine of the $k - machine$ model. It performs a vertex-cut partitioning using a coloring method. First, we partition the vertex set V into c color subsets. After that, each machine receives c^{4-l} quadruplets of colors, which is used to collect its required edges to form an induced subgraph of size $O(m/k)$.

Results correctness To study the correctness of our results, we define the following lemmas:

Lemma 1. *Consistency: all the graphlets are output once.*

Proof. The enumeration of each of the four vertex graphlets is local without any data communication between machines. Hence, we need to be sure that each graphlet is enumerated once.

1. Wedges: wedges of type 1 are output once on the model following the equation 2 with $l \leq 3$. Without loss of generality, suppose we have two quadruplet $q_1 = \{c_1, c_1, c_1, c_1\}$ and $q_2 = \{c_1, c_1, c_1, c_2\}$ on machine M_1. Notice that the first color triplet in q_1 and q_2 is the same. This color triplet is used to output the wedges of type 1 involved in all the intuitive graphlets. In our quadruplet assignment, we ensure that each machine from the $k - machine$ model acquires the same first triplet of color, hence each machine will exclusively output the wedges corresponding to its triplet of color.
2. Intuitive graphlets: Suppose on machine M, we have the quadruplet (c_1, c_2, c_3, c_4). The wedges with colors (c_1, c_2, c_3), (c_2, c_3, c_4), (c_1, c_2, c_4) and (c_1, c_3, c_4) are held by the tables $Wedge\_T1$, $Wedge\_T2$, $Wedge\_T3$ and $Wedge\_T4$ resp. on machine M without local repetitions. When we generate 3-Path (3-Star or rectangle) graphlets on M, we compute $Wedge\_T1 \bowtie_{\substack{T1.u = T2.v \\ \& T1.w = T2.v}}$

$Wedge\_T2$ ($Wedge\_T1 \bowtie_{\substack{T1.u = T3.v \\ \& T1.w = T3.v}} Wedge\_T3$ or $Wedge\_T1 \bowtie_{\substack{T1.u = T4.v \\ \& T1.w = T4.v}} Wedge\_T4$). Each wedge is unique on M, thus the generated paths (stars or rectangles) are unique on M because they consist of unique wedges on M. Furthermore, depending on the cluster size, each machine defines $4 - l$ quadruplet of colors that are uniquely and specifically assigned to it. Hence, the order of the colors of each quadruplet is unique on each machine (there is no repetition). As a result, the generated intuitive graphlets are output once, since their enumeration depends on the colors and the order of their corresponding quadruplet on each machine.

3. Derived graphlet: we proved above that each intuitive graphlet is output once, and since the output of the derived graphlets is based on the intuitive ones, then each derived graphlet is output once.

Lemma 2. *Completeness: There is no missing graphlet in the output.*

Proof. In our partitioning strategy, each vertex choose independently and uniformly a color from the c colors, in the same time, we create color quadruplets consisting of the different permutations of the c colors. As a result, each edge with its colored end-vertices involved in one or more 3-vertex or 4-vertex graphlets, its end-vertices colors are in one or more color quadruplets. Without loss of generality, suppose the vertex u chose color c_1 and the vertex v chose the color c_2. The edge (u, v) will be sent to the machine M defining (c_1, c_2) in one of its quadruplets. So, if the edge (u, v) is a part of a wedge or derived graphlet, it will be on M and according to the quadruplets on M, the wedge defining the intuitive graphlet or the derived graphlet involving (u, v) will be output. Finally, there will be no missing wedges or graphlets.

Graph isomorphism We explained previously that we only consider some permutations when outputting the graphlets. In fact, when we output $P = (u, v, w, z)$ as a 3-path graphlet, we need to eliminate the opposite direction $P' = (z, w, v, u)$, since P et P' are isomorphic. Hence, only the permutation defining $u < z$ is considered. In addition, we need to consider four permutations for 3-star graphelets, depending on the vertex at the center. For $S = (u, v, w, z)$ with v at the center, we need to consider the graphlet where $u < w < z$. All the other graphets with v at the center, are isomorphic to S. The rectangles $R = (u, v, w, z)$ are output in three permutations ($u < v < w < z$, $u < v < z < w$, $u < w < v < z$). All the remaining configurations are isomorphic to R, since the rectangle is a cycle of length 4. The tailed triangles $T = (u, v, w, z)$ accept twelve permutations. For each configuration of 3-star, we can generate three tailed triangles, considering $u < w < z$. The other configurations are the same as T. The diamonds $D = (u, v, w, z)$ need six permutations to be considered. We output two permutations for each rectangle. Hence, all the 18 remaining graphlets are isomorphic to D. Finally, for the cliques $C = (u, v, w, z)$ only one permutation is needed. All the other configurations are isomorphic to C.

4.2 Complexity and load balancing

Complexity Four graphlet enumeration is more difficult than the triangle enumeration. The complexity of triangle enumeration is $O(n^3)$, whereas it is $O(n^4)$ for four vertex graphlets. Our algorithm is bounded by $O(N_{\angle}^2)$ for the intuitive graphlets, where $N_{\angle}$ is the number of wedges. On the other hand, since the derived graphlets depends on the intuitive graphlets, their complexity is bounded by $O(max\{m\times S, m\times R\}$, with $S(R)$ is the number of 3-star (rectangle) graphlets.

Load balancing We mentioned previously that the vertex set V is partitioned into c subsets of $O(n/c)$ vertex each. Each machine then receives an induced sub-graph $G_x = (V_x, E_x)$ of G. The number of edges among the sub-graphs G_x is relatively balanced with high probability. Indeed, each vertex chose a color in a uniform manner, so it has $\frac{1}{c}$ to choose one of the c colors. Then, the edge between a couple of vertices has $\frac{1}{c} \times \frac{1}{c} = \frac{1}{c^2}$ possibility. This balances the load between the proxies. Each machine then collects the required edges to enumerate the graphlets, so it holds $(\frac{1}{c})^4 = \frac{1}{c^4}$ of each type of graphlets. As a result, each machine processes essentially the same number of graphlets, which leads to balance the workload.

5 Experimental Study

5.1 Hardware and software setup and data set

Hardware: Experiments are conducted on a modest cluster with 9 machines. Each machine has 4 cores CPU running at 2.2 Ghz on average, 4 GB of main memory, 500 GB of storage, 32 KB L1 cache, 1 MB L2 cache and Linux Ubuntu server 18.04 as operating system. The machines are connected on 1GB network cards with $128MB/s$ as bandwidth. Each machine managed two quadruplets.

Software: We used the columnar DBMS Vertica to execute our queries (the code is available at `https://github.com/lias-laboratory/sqlgraphlet`), since it is 10× faster than the row DBMSs for graph problems. However, any other parallel system that provides partitioning control can be used, including systems like SparkSQL and TigerGraph. Moreover, we compared our algorithm against D4GE [11]; a tool based on Spark for sub-graph enumeration.

Data sets: We used four real data sets from Stanford data set collection (`https://snap.stanford.edu`), summarized in Table 1.

5.2 Graphlet enumeration

Our experiments are presented in two sets : (1) the first concerns the evaluation of our approach results, including its load balancing and its speedup, and (2) the second set aims to compare our approach with D4GE[11] which is a distributed graphlets enumeration solution based on Spark. Each experiment is repeated three times and the average time measurement is reported.

Table 1: Data sets : order (n), size (m), triangle count (Δ) and maximum degree (d_max).

Data sets	n	m	Δ	d_max
Facebook	4,039	88k	1,612k	1,045
Pennsylvania	1,088k	1,541k	67k	9
Amazon	334k	925k	75k	549
DBLP	317k	1,049k	13k	343

Graphlet enumeration evaluation

Graphlet counting: Table 2 summarizes the wedges and the 4-vertex graphlets counting for each graph data set. Our partitioning strategy ensures to have each vertex with its incident edges on the same machine according to its color quadruplet. Thus, each graphlet is enumerated once on one machine if it exists.

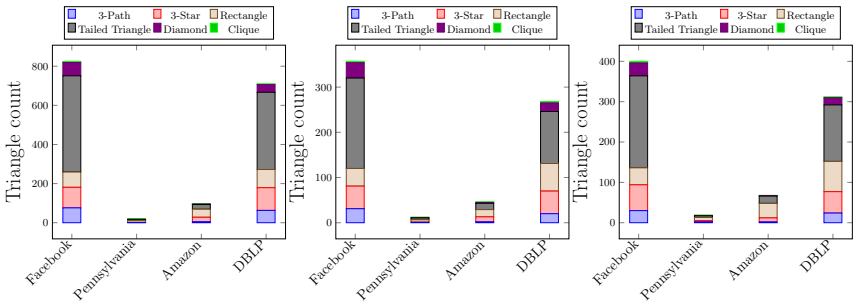
Table 2: Graphlet counting output.

Data sets	P	S	R	T	D	C
Facebook	1,055,326,189	727,318,426	144,023,053	703,783,680	138,773,046	30,004,668
Pennsylvania	7,384,597	1,707,904	157,802	318,190	5,795	21
Amazon	80,983,900	142,823,893	3,125,323	24,485,894	2,702,808	275,961
DBLP	675,637,762	431,568,151	55,107,655	316,232,255	54,904,261	16,713,192

P: 3-Path, S: 3-Star, R: Rectangle, T: Tailed Triangle, D: Diamond, C: Clique.

Our solution speed up: Fig. 6 depicts the local execution time for each graphlet algorithm on the graph data sets on a cluster of 4 machines, 8 machines and 9 machines. Notice that the partitioning time is negligible and it happens once at the beginning, so we didn't include it. From Fig. 6, we deduce that the most time consuming graphlet is the tailed triangle, because we classify this graphlet in three categories depending on the position of the triangle (at the beginning, in the middle or at the end), then we union the results together. Moreover, we can notice that the execution time of the queries improves as the size of the cluster increase. Hence, two conclusions can be drawn: (1) we can get up to 2× speedup with larger number of colors, and (2) we obtain better running time on models having less quadruplets managed by each machine. Furthermore, we notice that the derived graphlets require negligible execution time, since they are based on the intuitive graphlets. This opens doors for our stepwise enumeration strategy to list quickly larger graphlets based on smaller ones.

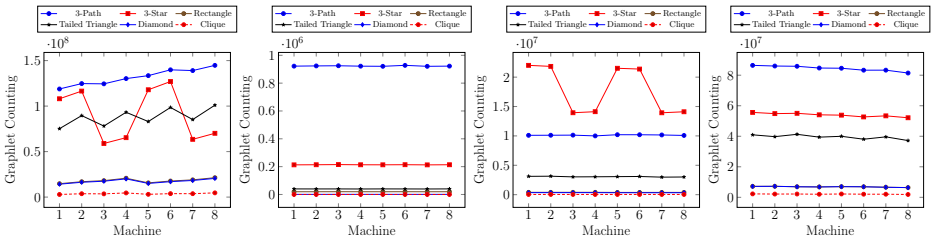
Balanced workload: To evaluate the load balancing of our partitioning strategy, we present Fig. 7, that presents line charts for the output of the graphlets on each machine, using a cluster with 8 machines. The workload balancing ensured by the partitioning strategy results in outputting almost the same count



(a) 4 Machines (b) 8 Machines (c) 9 Machines

Fig. 6: Local running time for graphlet enumeration (sec) with $k = \{4, 8, 9\}$ and $c = \{2, 2, 3\}$ resp.

of graphlets on each machine. The slight variation in counting is due to the randomization in the vertex coloring step. Precisely, the 3-Star query produced a variation between the machines because of the graph structure. Facebook and amazon data sets are particularly skewed graphs, that's why we can enumerate more 3-Star on only some machines.



(a) Facebook (b) Pennsylvania (c) Amazon (d) DBLP

Fig. 7: Load balancing on a cluster with 8 machines ($k = 8$ and $c = 2$).

Comparison with D4GE The distributed Spark solution D4GE[11] requires compressed input graphs, hence the use of graph compression tool like webGraph library [3] is mandatory. As a result, we compressed all the graph data sets using webGraph library before performing the experiments. This compression time is included in our evaluation as the pre-treatment time.

Table 3 provides a comparison between our approach and D4GE on a cluster of 8 machines using 2 colors. Despite its use of compressed data, D4GE shows more running time with sparse graphs as Pennsylvania. This data set is the

largest in our chosen data sets, but also the most sparse. On the other hand, our solution showed less efficient with dense graphs compared to D4GE. Note that unlike D4GE that only provide the graphlets count, we list the results. Our ultimate goal is to provide a tool that works by stage and save the results of each step. This has two impacts: (1) speed up the enumeration process with larger graphlets, and (2) provide a base to develop a recursive strategy to enumerate larger graph structure including the maximum cliques. To sum up, our solution gives an acceptable running time. It scales well, it does not require any graph compression or graph preparation beforehand, and it lists clearly and entirely all the four vertex graphlets in the input graph.

Table 3: Comparison of our solution against D4GE (sec).

Data sets	D4GE			Our solution		
	Pre-treatment	Enumeration	Total	Partitioning	Enumeration	Total
Facebook	8	20	28	1	250	251
Pennsylvania	25	18	43	4	10	14
Amazon	18	16	34	1	26	27
DBLP	16	28	44	2	177	179

6 Related Work

Graph Analytics is becoming a first-class challenge in database research [10]. Subgraph enumeration is among the fundamental problems that have received a lot of attention recently. It is based on recursive queries and transitive closure; two main graph problems that are largely and deeply studied in [14,8]. Triangle enumeration and counting is the simplest sub-graph enumeration problem and has been discussed in database perspective in many works such [2,7,1]. These works presented different partitioning strategies to minimize data exchange and to perform triangle enumeration locally. Moreover, many solutions for 4-vertex graphlet enumeration have been developed outside DBMS. [11] is a distributed solution based on Spark to enumerate all the triad and the four vertex graphlets in large compressed graphs. [11] was inspired by [15], who presented an efficient partitioning strategy based on coloring, that balances the workload to enumerate subgraphs but with repetitions. Other works for the induced subgraph enumeration, such as FanMod [16] and Rage [12] have emerged. However, they do not perform well on million-scale graphs and are less efficient. On the other hand, we are, to the best of our knowledge, the first to present a distributed solution for four vertex graphlet enumeration with queries, that can be reprogrammed using a programming language such as Python and MPI.

7 Conclusion

We present a novel distributed approach to solve 3-vertex and 4-vertex graphlets enumeration problem. Our current solution is programmed using SQL, but we can implement it with a programming language as Python, or a parallel system like Spark. Moreover, we experimentally proved that our partitioning strategy provides a perfect load balancing, and our solution scales well with the graph size. This study is promising and it can be efficiently extended to larger graphlets.

In future work, we will study the impact of the number of colors on query time. We will also study larger graphlet enumeration (of order α), compressing the graph using triangles as super-vertices. We plan to extend our algorithms to multicore CPUs and GPUs. As a longer term goal, we aim to solve clique enumeration, a significantly harder problem.

References

1. Ahmed, A., Enns, K., Thomo, A.: Triangle enumeration for billion-scale graphs in RDBMS. In: Proc. of AINA (2021)
2. Al-Amin, S., Ordonez, C., Bellatreche, L.: Big data analytics: Exploring graphs with optimized SQL queries. In: Proc. of DEXA (2018)
3. Boldi, P., Vigna, S.: The webgraph framework i: Compression techniques. In: Proc. of WWW (2004)
4. Bröcheler, M., Pugliese, A., Subrahmanian, V.: Cosi: Cloud oriented subgraph identification in massive social networks. In: Proc. of IEEE ASONAM (2010)
5. Charbey, R., Prieur, C.: Stars, holes, or paths across your facebook friends: A graphlet-based characterization of many networks. Netw. Sci. **7**(4), 476–497 (2019)
6. Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. IEEE TKDE **17**(8), 1036–1050 (2005)
7. Farouzi, A., Bellatreche, L., Ordonez, C., Pandurangan, G., Malki, M.: A scalable randomized algorithm for triangle enumeration on graphs based on SQL queries. In: Proc. of DaWaK (2020)
8. Jachiet, L., Genevès, P., Gesbert, N., Layaida, N.: On the optimization of recursive relational queries: Application to graph queries. In: Proc. of ACM SIGMOD (2020)
9. Klauck, H., Nanongkai, D., Pandurangan, G., Robinson, P.: Distributed computation of large-scale graph problems. In: Proc. of ACM-SIAM SODA (2015)
10. Lan, M., Wu, X., Theodoratos, D.: Answering graph pattern queries using compact materialized views. In: Proc. of DOLAP (2022)
11. Liu, X., Santoso, Y., Srinivasan, V., Thomo, A.: Distributed enumeration of four node graphlets at quadrillion-scale. In: Proc. of SSDBM (2021)
12. Marcus, D., Shavitt, Y.: Rage – a rapid graphlet enumerator for large networks. Computer Networks **56**(2), 810–819 (2012)
13. Milenković, T., Przulj, N.: Uncovering biological network function via graphlet degree signatures. Cancer Inform **6**, CIN–S680 (2008)
14. Ordonez, C., Cabrera, W., Gurram, A.: Comparing columnar, row and array dbmss to process recursive queries on graphs. Inf. Syst. **63**, 66–79 (2017)
15. Park, H., Silvestri, F., Pagh, R., Chung, C., Myaeng, S., Kang, U.: Enumerating trillion subgraphs on distributed systems. ACM TKDD **12**(6), 71:1–71:30 (2018)

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