Practical Survey on MapReduce Subgraph Enumeration Algorithms

Xiaozhou Liu, Yudi Santoso, Venkatesh Srinivasan, Alex Thomo

Abstract Subgraph enumeration is a basic task in many graph analyses. Therefore, it is necessary to get this task done within a reasonable amount of time. However, this objective is challenging when the input graph is very large, with millions of nodes and edges. Known solutions are limited in terms of scalability. Distributed computing is often proposed as a solution to improve scalability. However, it has to be done carefully to reduce the overhead cost and to really benefit from the distributed solution. In this work we provide a comprehensive overview of several Map-Reduce subgraph enumeration algorithms which currently represent the state of the art. We identify and describe the main conceptual approaches, giving insight on their advantages and limitations, and provide a summary of their similarities and differences.

1 Introduction

Finding occurrences of particular subgraphs in a network/graph is a tool for analyzing and understanding such network. This analytical tool has many applications in various fields, such as in biology [10, 14] chemistry [8, 21], social study [9, 4], network classification [25], and more. Moreover, some applications require the enumeration of all subgraphs up to a certain degree. For example, Milenkovic and

Xiaozhou Liu
University of Victoria, Victoria, BC, Canada, e-mail: `xiaozhou@uvic.ca`

Yudi Santoso
University of Victoria, Victoria, BC, Canada,e-mail: `santoso@uvic.ca`

Venkatesh Srinivasan
University of Victoria, Victoria, BC, Canada,e-mail: `srinivas@uvic.ca`

Alex Thomo
University of Victoria, Victoria, BC, Canada,e-mail: `thomo@uvic.ca`

Przulj [14] used 2, 3, 4, and 5 node induced connected subgraphs (or graphlets) to analyse Protein-Protein-Interaction (PPI) networks.

Subgraph enumeration problem is challenging when the size of the input graph is large. This can be understood by analysing the complexity. The computational complexity grows exponentially by the order of the subgraph to enumerate. Suppose we want to find subgraphs of k nodes in a graph of n nodes, then there are $\binom{n}{k} \propto n(n-1)\ldots(n-k+1)$ possible combinations to examine. If $n \gg k$, this is approximately n^k. On the other hand, an order of magnitude increase in n would increase the complexity by 10^k. In practice, not all combinations need to be checked. Nonetheless, it is generally true that the number of subgraphs grows rapidly with the size of the graph, and for enumeration the algorithm needs to touch each subgraph.

Using a distributed platform is an obvious option to increase the computing power, where we can add more compute nodes to get more done within a fixed time budget. Known distributed solutions [5, 24, 16, 18] had limited scalability, and were only designed to enumerate *triangle* subgraphs. The work of Park et al. improved upon these previous solutions and was able to enumerate triangles from graphs with billions edges [17]. The same authors, furthermore, generalized their work to support arbitrary subgraph query [19]. As argued in [19], their proposed method is superior to join-based methods of [1] and [11], because those can generate a large amount of intermediate data during the shuffle step. Liu et al. [13] identified the redundant computational work in Park et al.'s proposal [19], and devised a duplication-free distributed algorithm, enumerating quadrillions of 4-node graphlets.

We consider in this survey only works that solve the problem of induced subgraphs. For example, if we discover a clique of 4 nodes, we only enumerate the clique, but not any other 4-node subgraphs inside it. Small induced connected subgraphs, such as 4-cliques, are known as graphlets.

1.1 Contributions

We provide a practical review of several distributed algorithms for subgraph enumeration, focusing on the state-of-the-art works of [16, 18, 17, 19, 13]. We give a structured retrospective reflection on the algorithms for computing exact subgraph occurrences using the *MapReduce* paradigm [1]. We also identify and describe the main conceptual ideas, giving insight on their main advantages and possible limitations. We provide a complete overview table that classifies the key characteristics of the algorithms, highlighting their main similarities and differences.

[1] Here we use the term MapReduce in a broad sense. The algorithms in this work can be implemented in newer frameworks such as Apache Spark as well.

2 Preliminaries

In the following we give some important notions that we use in this paper.

Subgraph. A graph $H(V_H, E_H)$ is called a *subgraph* of $G(V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$.

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 $(u, v) \in E_H$ if and only if $(u, v) \in E_G$.

Graphlet. A small induced connected subgraph. There are two types of 3-node graphlets: wedge and triangle as shown in Fig. 1, and six types of 4-node graphlets, shown in Fig. 2: 3-path, 3-star, rectangle, tailed-triangle, diamond and 4-clique.

Fig. 1: 3-node graphlets: a wedge and a triangle.

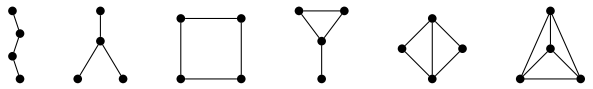

Fig. 2: 4-node graphlets: a 3-path, a 3-star, a rectangle or 4-cycle, a tailed-triangle, a diamond, and a 4-clique.

Edge-orientation. Assigning directions to edges in an undirected graph $G(V, E)$. Given a function η which determines a total ordering of the nodes in V, an undirected edge $(u, v) = (v, u)$ is orientated by η, such that if $\eta(u) < \eta(v)$, only (u, v) is listed but not (v, u). Edge-orientation transforms the undirected graph into a Directed Acyclic Graph (DAG).

Coloring. Applying a color function to each edge such that the entire graph can be partitioned into many subgraphs. First, define a function θ which maps vertex $u \in V$ to $\theta(u)$. Vertex u is said to have *color* $\theta(u)$. Edge $(u, v) \in E$ is said to have color $(i = \theta(u), j = \theta(v))$.

Edgeset. After coloring is applied to $G(V, E)$, edges of the same color are grouped together to form a subgraph, denoted by E_{ij}. E_{ij} contains all edges $(u, v) \in E$ with color (i, j).

Directed Edgeset. Edgeset of a directed graph, denoted by E^*_{ij}, representing all the edges of $\overrightarrow{E}$ of $\overrightarrow{G}(V, \overrightarrow{E})$, that originates from vertices of color i and points to vertices of color j. For $i \neq j$, $E^*_{ij} \cup E^*_{ji} = E_{ij}$; for $i = j$, $E^*_{ij} = E_{ij}$.

Sub-problem. Refers to the union of (directed) edgesets of particular colors where the distributed workers can discover all the subgraphs/graphlets in that union independently. It is the fundamental enumeration task assigned to each distributed worker.

Symmetrization. Refers to deriving graph $G^{\text{sym}}(V, E^{\text{sym}})$ from the original graph $\overrightarrow{G}(V, \overrightarrow{E})$, where $E^{\text{sym}} = \overrightarrow{E} \cup \{\overrightarrow{(v,u)} | \forall \overrightarrow{(u,v)} \in \overrightarrow{E}\}$. In other words, G^{sym} possesses the original edge as well as the reversed edge in $\overrightarrow{G}$.

MapReduce. A *de-facto* programming scheme for distributed processing [7]. The paradigm orchestrates the processing in three operations: Map, Shuffle and Reduce. Each operation facilitates its own duty, drastically reducing the complexity of implementation.

3 Serial Graphlet Enumerations

The distributed algorithms of interests in this work are composed using MapReduce paradigm, meaning all of them consist of a localized serial enumeration algorithm with a partition scheme.

All of the distributed algorithms, except one, are employing some well-known serialized triangle/subgraph enumeration such as CompactForward [12] and VF2 [6], the introduction of which are omitted here. In this section we introduce a single machine 4-node graphlet enumeration algorithm that was proposed recently, called *Simultaneous 4-node Graphlets Enumeration* (S4GE) [23].

3.1 4-node Graphlet Enumeration

Algorithm 1 S4GE

```
Require: An oriented-by-degree, symmetrized graph $\overrightarrow{G}(V, \overrightarrow{E})$
1: for all $\overrightarrow{(u,v)} \in \overrightarrow{E}$ do
2:     for all $u' \in N(u)$ and $v' \in N(v)$ do
3:         if $(u' > u) \wedge (v' \geq u)$ then
4:             if $u' = v' > v$ then
5:                 EXPLORETRIANGLE $(u, v, u')$
6:             if $((u' < v') \vee (v' = u)) \wedge (u' > v)$ then
7:                 EXPLOREWEDGETYPE1 $(v, u, u')$
8:             if $(u' > v') \wedge (v' \neq u)$ then
9:                 EXPLOREWEDGETYPE2 $(u, v, v')$
```

Algorithm 2 EXPLORE TRIANGLE

```
Require: Triangle $(u, v, w)_3$ with $u < v < w$; symmetrized neighbourhood $N(u)$, $N(v)$ and $N(w)$.
1: $N^{>u}(u) \equiv \{z | z \in N(u), \eta(z) > \eta(u)\}$; $N^{>u}(v) \equiv \{z | z \in N(v), \eta(z) > \eta(u)\}$; $N^{>u}(w) \equiv \{z | z \in N(w), \eta(z) > \eta(u)\}$
2: for all $z \in N^{>u}(u) \cap N^{>u}(v) \cap N^{>u}(w)$ with $z > w$ do
3:     ENUMERATE4CLIQUE $(u, v, w, z)$
4: for all $z$ in two sets and $z >$ opposite vertex do
5:     ENUMERATEDIAMOND $(u, v, w, z)$
6: for all $z$ in one set only do
7:     ENUMERATETAILEDTRIANGLE $(u, v, w, z)$
```

The algorithm first discovers triangles and wedges in a similar fashion as CompactForward [12], then it proceeds to search for 4-node graphlets after the discovery of triangles and wedges. That is, it first discovers triangles and wedges, and for each triangle that it locates, it checks if this triangle is a part of any tailed triangles, diamonds and 4-cliques. Similarly, through wedges it checks for 3-paths, 3-stars and rectangles. The checks are done by intersecting the neighbourhoods of the three vertices of the triangles and wedges. The details can be found in [23]. S4GE has a running time $O(T_{3g} + (|\Delta| + |\angle|)d_{max}^{sym})$, where T_{3g} is the time to enumerate wedges and triangles, and d_{max}^{sym} is the maximum degree of the symmetrized input graph. The pseudo code of S4GE for discovering all the triangles and wedges is listed as Algorithm 1; the pseudo code of S4GE for discovering all the six types of 4-node graphlets are listed as Algorithms 2, 3, and 4.

Algorithm 3 EXPLORE WEDGE TYPE-1

```
Require: Wedge $(v,u,w)_1$ with $u < v < w$; symmetrized neighbourhood $N(u)$, $N(v)$ and $N(w)$.
1: $N^{>u}(u) \equiv \{z|z \in N(u), \eta(z) > \eta(u)\}$; $N^{>u}(v) \equiv \{z|z \in N(v), \eta(z) > \eta(u)\}$; $N^{>u}(w) \equiv \{z|z \in N(w), \eta(z) > \eta(u)\}$
2: for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ do
3:     ENUMERATERECTANGLE $(u,v,z,w)$
4: for all $z \in N^{>u}(u)$ only do
5:     if $z > w$ then
6:         ENUMERATE3STAR $(u,v,w,z)$
7: for all $z \in N^{>u}(v)$ only do
8:     ENUMERATE3PATH $(w,u,v,z)$
9: for all $z \in N^{>u}(w)$ only do
10:    ENUMERATE3PATH $(v,u,w,z)$
```

Algorithm 4 EXPLORE WEDGE TYPE-2

```
Require: Wedge $(u,v,w)_2$ with $u < v$ and $u < w$; symmetrized neighbourhood $N(v)$ and $N(w)$.
1: $N^{>u}(v) \equiv \{z|z \in N(v), \eta(z) > \eta(u)\}$; $N^{>u}(w) \equiv \{z|z \in N(w), \eta(z) > \eta(u)\}$
2: for all $z \in N^{>u}(v)$ only do
3:     if $z > w$ then
4:         ENUMERATE3STAR $(v,u,w,z)$
5: for all $z \in N^{>u}(w)$ only do
6:     if $z \neq v$ then
7:         ENUMERATE3PATH $(u,v,w,z)$
```

4 Distributed Subgraph Enumeration

Park et al. [17] proposed three variants of distributed algorithms for triangle enumeration, and PTE_{Base} is the base version. PTE_{Base} partitions the undirected input graph into edgesets which are stored on a distributed file system. It defines a set of overlapping sub-problems such that each sub-problem can be solved independently. Each sub-problem is assigned to a distributed worker, and the worker proceeds to read the edgesets that are necessary to solve the sub-problem. PTE_{CD} improves on PTE_{Base} by using *color-direction* technique, which enable it to minimize the redundant computations. Finally, PTE_{SC} improves on PTE_{CD} by minimizing network traffic through a careful scheduling. However, with modern network technology and internal network connections networking cost has been of a less concern. Later on, Park et al. generalized PTE to support the enumeration of subgraphs of any order, and called their solution PSE [19].

4.1 *PTE*$_{\text{Base}}$

PTE$_{\text{Base}}$ defines three types of sub-problems: type-1 sub-problems, S_i, that emits triangles with all three vertices of the same colors; type-2 sub-problems, S_{ij}, that emitts triangles of two different colors; and type-3 sub-problems, S_{ijk}, that emitts triangles with three different colors. The triangles emitted from type-1 sub-problems are called type-1 triangles, and similarly for type-2 and type 3 triangles.

PTE$_{\text{Base}}$ partitions the input graph into edgesets, E_{ij} for $(i,j) \in \{0,1,\ldots,\rho-1\}^2$. For $i = j$ there are $\binom{\rho}{1}$ of such edgesets E_{ii}, and $\binom{\rho}{2}$ E_{ij} for $i \neq j$. The Modulo operation of ρ is used to color an edge. Each edge (u,v) belongs to the edgeset $E_{(u\%\rho)(v\%\rho)}$.

PTE$_{\text{Base}}$ computes $\binom{\rho}{2}$ type-2 sub-problems S_{ij}, and $\binom{\rho}{3}$ type-3 sub-problems, S_{ijk}. Type-1 sub-problems S_i can be embedded into type-2 sub-problem S_{ij}, hence do not need to be computed separately. For example, with $\rho = 4$, the sub-problems are: 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} and S_{123}. However, since solving S_{01} requires $E_{00} \cup E_{01} \cup E_{11}$, S_0 can also be solved along the process, as S_0 only requires E_{00}. Notice that S_0 can be embedded not only in S_{01}, but also in S_{02} and S_{03}. To avoid duplicated embedding, type-1 triangles of color i is only emitted when $i+1 = j\%\rho$ for a type-2 sub-problem S_{ij}.

PTE$_{\text{Base}}$ employs CompactForward [12] algorithm as the local serial algorithm. Because PTE$_{\text{Base}}$ partitions the input graph into $O(\rho^2)$ colored edgesets, each sub-problem is expected to process a subgraph of size $O(m/\rho^2)$, where m is the number of edges in the graph, and each distributed worker is expected to do $O((m/\rho^2)^{1.5}) = O(m^{1.5}/\rho^3)$ amount of work. Summing over all $O(\rho^3)$ sub-problems, PTE$_{\text{Base}}$ does $O(m^{1.5})$ amount of work overall, which is the same amount of work as applying CompactForward [12] on a single machine. The network read for PTE$_{\text{Base}}$ is $O(\rho m)$, since each of the $O(\rho^3)$ sub-problems needs to read $O(m/\rho^2)$ amount of data.

The pseudo code of PTE$_{\text{Base}}$ is given as Algorithm 5:

Algorithm 5 PTE$_{\text{Base}}$

```
Require: An undirected graph $G(V,E)$; the number of colors $\rho$
1: Construct $\overrightarrow{G}(V,\overrightarrow{E})$ by applying edge-orientation to $G(V,E)$
2: Partition $\overrightarrow{E}$ into edge sets $E_{ij}$ using $\rho$ colors
3: Generate all sub-problems $\{S\}$ for the given $\rho$
4: for all sub-problem $S_{\text{Cs}} \in \{S\}$ do                ▷ Distributed-for
5:     $E' \leftarrow E_{ij}$ from distributed storage.
6:     COMPACTFORWARD ($E'$)
```

4.2 *PTE*$_{\text{CD}}$

Park et al. [17] improve on PTE$_{\text{Base}}$ by introducing PTE$_{\text{CD}}$, *color-direction*. Without loss of generality, consider a type-3 sub-problem S_{ijk}. If we only care about enumerating all the triangles (u,v,w) with color (i,j,k), PTE$_{\text{Base}}$ does extra work when intersecting $\overrightarrow{N}(u)$ and $\overrightarrow{N}(v)$. CompactForward algorithm used in PTE$_{\text{Base}}$ considers all possible neighbouring vertices of u, from both E^*_{ij} and E^*_{ik}, and all possible neighbouring vertices of v, from both E^*_{ji} and E^*_{jk}. This is unnecessary since for any triangle (u,v,w) with color (i,j,k), given vertices u and v with color i and j respectively, we know that w must have color k, hence only need to intersect E^*_{ik} and E^*_{jk}.

PTE$_{\text{CD}}$ exploits this fact, and lists all six possible color-direction cases for any type-3 sub-problem: for a type-3 sub-problem S_{ijk}, an arbitrary edge $\overrightarrow{(u,v)}$ of CompactForward algorithm can have colors of $\{(i,j),(j,i),(i,k),(k,i),(j,k),(k,j)\}$, and PTE$_{\text{CD}}$ only needs to intersect $\overrightarrow{N}(u)$ and $\overrightarrow{N}(v)$ from edge sets $\{(E^*_{ik},E^*_{jk})$, (E^*_{jk},E^*_{ik}), (E^*_{ij},E^*_{kj}), (E^*_{kj},E^*_{ij}), (E^*_{ji},E^*_{ki}), $(E^*_{ki},E^*_{ji})\}$ respectively. Similarly, PTE$_{\text{CD}}$ lists all six cases for any type-2 sub-problem as well.

Park et al. prove that by adapting PTE$_{\text{CD}}$, the total number of operations is reduced by a factor of $2-\frac{2}{\rho}$ compared to PTE$_{\text{Base}}$.

The pseudo code of PTE$_{\text{CD}}$ is given as Algorithm 6 and Algorithm 7:

Algorithm 6 PTE$_{\text{CD}}$

```
Require: An undirected graph $G(V,E)$; the number of colors $\rho$
 1: Construct $\overrightarrow{G}(V,\overrightarrow{E})$ by applying edge-orientation to $G(V,E)$
 2: Partition $\overrightarrow{E}$ into directed edge sets $E^*_{ij}$ using $\rho$
 3: Generate all sub-problems $\{S\}$ given $\rho$
 4: for all sub-problem $\in \{S\}$ do                                   ▷ Distributed-for
 5:     if sub-problem of type $S_{ij}$ then
 6:         Read $E^*_{ii}$, $E^*_{ij}$, $E^*_{ji}$ and $E^*_{jj}$
 7:         if $i+1 == j\%\rho$ then
 8:             CompactForward_CD ($E^*_{ii}$,$E^*_{ii}$,$E^*_{ii}$)                      ▷ Type-1
 9:         else
10:             for all (p,q,r) $\in \{(i,i,j),(i,j,i),(j,i,i),(j,j,i),(j,i,j),(i,j,j)\}$ do
11:                 CompactForward_CD ($E^*_{pq}$,$E^*_{pr}$,$E^*_{qr}$)              ▷ Type-2
12:     if sub-problem of type $S_{ijk}$ then
13:         Read $E^*_{ij}$, $E^*_{ji}$, $E^*_{ik}$, $E^*_{ki}$, $E^*_{jk}$ and $E^*_{kj}$
14:         for all (p,q,r) $\in \{(i,j,k),(i,k,j),(j,i,k),(j,k,i),(k,i,j),(k,j,i)\}$ do
15:             CompactForward_CD ($E^*_{pq}$,$E^*_{pr}$,$E^*_{qr}$)                  ▷ Type-3
```

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

```
Require: An edge-oriented edge set $E^*_{ij}$, $E^*_{ik}$ and $E^*_{jk}$
 1: for all $\overrightarrow{(u,v)} \in E^*_{ij}$ do
 2:     for all $w \in \{\overrightarrow{N}(u) | \overrightarrow{N}(u) \in E^*_{ik}\} \cap \{\overrightarrow{N}(v) | \overrightarrow{N}(v) \in E^*_{jk}\}$ do
 3:         Enumerate $(u,v,w)$
```

4.3 PSE

Park et al. generalise PTE$_{\text{Base}}$ to support non-induced subgraph query of an arbitrary order, called PSE (Pre-partitioned subgraph Enumeration) [19]. PSE takes a query subgraph $G_q(V_q,E_q)$ of order k as input where $k=|V_q|$, and enumerates all the matching subgraphs to G_q. PSE employs VF2 algorithm [6] as the local serial algorithm for query graph matching.

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}.

PSE then makes the observations that solving all the sub-problems independently introduces duplicated emissions. Some sub-problems can be grouped together to reduce duplications. This can be shown by the following: continuing with the above example, S_{012}=$E_{00} \cup E_{11} \cup E_{22} \cup E_{01} \cup E_{02} \cup E_{12}$, S_{01}=$E_{00} \cup E_{01} \cup E_{11}$ and S_0=E_{00}. It is clear that $S_0 \subset S_{01} \subset S_{012}$. In other words, enumerating the subgraphs from S_{012} also enumerates all the subgraphs from S_0 and S_{01}.

To avoid duplicated emissions, PSE introduces *sub-problem groups* and the *dominant sub-problem* of the sub-problem group. In the above example, S_0, S_{01} and S_{012} forms a sub-problem group; and since enumerating S_{012} is sufficient to enumerate all sub-problems of the group, S_{012} is the dominant sub-problem of the sub-problem group. In PSE, each sub-problem group becomes the fundamental computing task on each distributed worker; solving the dominant sub-problem of each sub-problem group is equivalent to solving all the sub-problems under this group.

PSE carefully groups the sub-problems to ensure a balanced workload distribution, such that all the sub-problem groups have similar number of sub-problems assigned. The process of generating such grouping can be described as:

1. Assign sub-problems $S_{c_0,c_1,\ldots,c_l}$ where $|\{c_0,c_1,\ldots,c_l\}| = k$ and $|\{c_0,c_1,\ldots,c_l\}| = k-1$ to different groups. It is clear that these sub-problems can never dominate each other, hence belong to their own group. These sub-problems are the prospective dominant sub-problems of their groups.
2. Assign the rest of the sub-problems to the groups created in Step 1. The assignment is prioritized towards the group with the least number of sub-problems in that group, while ensuring that the subproblem is covered by the dominat sub-problem within that group. Ties are broken randomly.
3. Repeat Step 2 until all sub-problems are assigned.

The pseudo code of PSE is given in Algorithm 8:

Algorithm 8 PSE

```
Require: An undirected graph $G(V,E)$; a query graph $G_q(V_q,E_q)$; the number of colors $\rho$
1: Construct $\overrightarrow{G}(V,\overrightarrow{E})$ by applying edge-orientation to $G(V,E)$
2: Partition $\overrightarrow{E}$ into edgesets $E_{ij}$ using $\rho$
3: Generate sub-problem groups $\{SGs\}$
4: for all sub-problem group $SG \in \{SGs\}$ do                    ▷ Distributed-for
5:     Select dominant sub-problem $S_d$ from $SG$
6:     $E'$=READEDGESETS($d$, $|V_q|$)
7:     $R$ = VF2($E'$, $G_q(V_q,E_q)$)
8:     $R'$ = DE-DUPLICATE(R)                                       ▷ PSE emits duplicates
9:     Enumerate(R')
```

Park et al. showed that PSE requires at most $\binom{\rho-1}{k-2}|E|$ amount of network read, for k-order subgraph query with input graph E.

PSE by Park et al., when published, was widely considered as the SotA of the subgraph enumeration. However the scheme discovers duplicated subgraphs. To correct the enumerations, PSE post-filters the duplicated subgraphs from the final emission. This motivates Liu et al. [13] to derive a partitioning scheme that guarantees duplication-free from the onset.

4.4 Distributed 4-node Graphlet Enumeration

Liu et al. introduced a partitioning scheme for S4GE algorithm, enabling it to be deployed in a distributed setting [13]. The partitioning scheme is named as D4GE, short for Distributed 4-node Graphlet Enumeration.

First, it is obvious that each enumerated subgraph can be mapped to only one colored tuple - for example, when enumerating 4-node graphlets, any discovered graphlet $(u,v,w,z)_4$ has a particular 4-tuple (i,j,k,l) representation, where $i=\theta(u)$, $j=\theta(v)$, $k=\theta(w)$, $l=\theta(z)$ and θ is the coloring function. The colored tuples are called the **color-assignments** of the corresponding graphlet, denoted by K_{ijkl}: the graphlet $(u,v,w,z)_4$ with color (i,j,k,l) has color-assignment K_{ijkl}.

Next, linearity is asserted over the 4-node graphlets query, such that $(u,v,w,z)_4$ is emitted if and only if $\eta(u)<\eta(v)<\eta(w)<\eta(z)$, where η is the edge-orientation function. The linearity on the emitted 4-node graphlets also implies ordering on the color-assignment of the graphlets: for an emitted graphlet $(u,v,w,z)_4$ with $\eta(u)<\eta(v)<\eta(w)<\eta(z)$, its color-assignment K_{ijkl} must satisfy $i\prec j\prec k\prec l$, where $i\prec j$ means i precedes j. On the other hand, the ordering of the color-assignments also imposes the linearity of the underlying graphlets: K_{ijkl} can only represent the graphlets $(u,v,w,z)_4$ with $\eta(u)<\eta(v)<\eta(w)<\eta(z)$. Both the graphlets and the color-assignments follow the same ordering in colors. Given the number of color ρ for 4-node graphlet enumeration, there are ρ^4 color-assignments.

While previous work employed the idea of color-direction to reduce the amount of work (PTE_{CD}), D4GE differentiates itself by the fact that D4GE exploits both the linearity of the DAG and the color-assignment, along with the fact that the color-assignment problem is essentially a combination problem. The unique relationship between any subgraph and its color-assignment guarantees the duplication-free nature of D4GE. In addition, previous works explicitly list all the ordered color-tuples in the algorithm, while D4GE utilizes combinations to generalize the color-assignments. This works not only for $k=4$, but also to any order k (with ρ^k color-assignments).

The pseudo code of D4GE is given as Algorithms 9:

Algorithm 9 D4GE

```
Require: An undirected graph $G(V,E)$; the number of colors $\rho$
1: Construct $\overrightarrow{G}(V,\overrightarrow{E})$ by applying edge-orientation to $G(V,E)$
2: Symmetrise $\overrightarrow{G}(V,\overrightarrow{E})$ into $G^{\mathrm{sym}}(V,E^{\mathrm{sym}})$
3: Partition $E^{\mathrm{sym}}$ into directed edgeset $E^*_{ij}$.
4: Generate sub-problems $S_{ij}\cup S_{ijk}\cup S_{ijkl}$.
5: for all $S_{\mathrm{Cs}}\in S_{ij}\cup S_{ijk}\cup S_{ijkl}$ do                ▷ Distributed-for
6:     $K_s\leftarrow$ Grouped color-assignments under $S_{\mathrm{Cs}}$
7:     $E_{\mathrm{map}}\leftarrow$ Read directed edgesets from distributed storage.
8:     for all $K_{ijkl}\in K_s$ do
9:         S4GE$_{\mathrm{CD}}$ $(E_{\mathrm{map}}, ijkl)$
```

D4GE groups the color-assignments into sub-problems (line 6 of Algorithm 9). 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 edgeset E^*_{00} is then loaded twice. To address this, color-assignments K_{pqrs} are grouped into sub-problems. Inherently S_{ijkl} is used 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 sorting $\{p,q,r,s\}$ in ascending order, and *reduced* means removing the duplicated colors from the sequence $\{p,q,r,s\}$. In the example of K_{2010}, the sorted and reduced form of $\{2,0,1,0\}$ is $\{0,1,2\}$. Therefore K_{2010} belongs to S_{012}.

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} (omitted S_i) where all four colors are the same, we attach K_{iiii} to sub-problem S_{ij} where $i+1 = j\%\rho$. Each sub-problem is computed independently on a distributed worker.

Liu et al. in [13] proved that with the same serialized local algorithm S4GE, D4GE requires no more than $2m^{\text{sym}}$ amount network read than PSE partitioning scheme. Under the same condition, Liu et al. showed that PSE does
$3\left(\binom{\rho-1}{2} - \rho\right) d^{\text{sym}}_{\max}(|\Delta_I| + |\angle_I|) - 6 d^{\text{sym}}_{\max}(|\Delta_{II}| + |\angle_{II}| + |\Delta_{III}| + |\angle_{III}|)$ amount of extra work when enumerating all six types of 4-node graphlets. Liu et al. argued 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. Thus D4GE with S4GE$_{\text{CD}}$ can often achieve greater performance improvement. In the Experiment chapter, Liu et al. showed that D4GE/S4GE$_{\text{CD}}$ combo achieved a convincing 10-fold speedup compared against PSE/S4GE with over eight different datasets, with almost perfect scalability up-to 256 distributed workers. Lastly, Liu et al. proved D4GE/S4GE$_{\text{CD}}$ combo's capability by enumerating the *indochina* dataset ([2], [3]), emitting more than ten quadrillion (10×10^{15}) 4-node graphlets, which is the first of its kind.

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

S4GE is modified accordingly so that it is able to enumerate all 4-node graphlets for an ordered color-assignment K_{ijkl}. This modified version is called S4GE$_{\text{CD}}$.

Instead of enumerating on a complete graph, S4GE$_{\text{CD}}$ now enumerates on a subgraph denoted by the color-assignment K_{ijkl}. The subgraph consists of a mapping between the ordered color 2-tuples (i,j) and the corresponding directed edgesets 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 edgesets are used to discover the wedge or triangle, and (i,l), (j,l), (k,l) and the corresponding edgesets 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 pseudocode for S4GE$_{\text{CD}}$ is given in Algorithm 10, with the details of the explore-functions are given in Algorithms 11, 12 and 13 respectively.

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

Algorithm 11 ExploreTriangle$_{\text{CD}}$

```
Require: Given triangle $(v,u,w)$; $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; color-assignment $ijkl$.
 1: $E^*_{il} \equiv E_{\text{map}}[il]$, $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
 2: $N^{>u}(u) \equiv \{z | z \in N(u)|_{E^*_{il}}, \eta(z) > \eta(u)\}$
 3: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
 4: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
 5: Rest follows Algorithm 2 line 2.
```

Algorithm 12 ExploreWedge-1$_{\text{CD}}$

```
Require: Given wedge $(v,u,w)$; $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; color-assignment $ijkl$.
 1: $E^*_{il} \equiv E_{\text{map}}[il]$, $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
 2: $N^{>u}(u) \equiv \{z | z \in N(u)|_{E^*_{il}}, \eta(z) > \eta(u)\}$
 3: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
 4: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
 5: Rest follows Algorithm 3 line 2.
```

Algorithm 13 ExploreWedge-2$_{\text{CD}}$

```
Require: Given wedge $(v,u,w)$; $E_{\text{map}} \equiv \{(i,j) \mapsto E^*_{ij}\}$; ordered color-assignment $ijkl$.
 1: $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
 2: $N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
 3: $N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
 4: Rest follows Algorithm 4 line 2.
```

5 Summary

In this section we present to the audience a tabular overview of the characteristics of the distributed algorithms studied in this paper. Some of the key characteristics are listed in Table 1.

Table 1: Summary of the four mentioned subgraph enumeration algorithms. CW12 (Clueweb12), SD (SubDomain) and IC (Indochina) are referring to the largest graphs that have ever been processed using the respective algorithms.

Characteristics	PTE_{Base}	PTE_{CD}	PSE	D4GE
Runtime	$O(m^{3/2})$	$O(m^{3/2})$	Query-dependent	$O((\lvert\Delta\rvert + \lvert\angle\rvert)d_{max}^{\text{sym}})$
NetworkRead	$O(\rho m)$	$O(\rho m)$	$\binom{\rho-1}{k-2} * m$	$[\binom{\rho}{2} - \rho + 3 - \frac{2}{\rho}]m^{\text{sym}}$
Capability	3×10^{12}(CW12)	3×10^{12}(CW12)	0.27×10^{15}(SD)	2.8×10^{15}(IC)
Application	Triangle	Triangle	QueryAnswering	4-node Graphlet
SotA	N	Y	Y	Y

All four distributed algorithms are within the same runtime class of their serialized counterpart, namingly, the runtime of CompactForward for PTE_{Base} and PTE_{CD}, VF2 for PSE and S4GE for D4GE. In the case of PSE, because the runtime complexity is dependent on the user-query, no exact expression is given by the authors. While not proved directly, the authors of D4GE presented a high correlation between the enumeration time against and the augmented runtime complexity. Overall the results are welcomed as none of the candidates incurs additional complexity to the enumeration tasks that are already challenging.

The network read of all four algorithms increases with respect to the number of colors ρ. This can be understood by the fact that all of the algorithms share the same definition of sub-problems, and sub-problems have overlaps, and the amount of overlaps can be parameterized by ρ. Without loss of generality, consider tri-color sub-problems S_{ijk}. Fixing the colors of i and j, there are precisely ρ number of variations in third color k, meaning that if ρ increases, the number of tri-color sub-problems also rises.

Next we summarize the maximum number of emitted subgraphs/graphlets and the respective input graphs of the candidate algorithms. This is used to indicate the capability of the candidate algorithms in their unique application. Both PTE_{Base} and PTE_{CD} are able to enumerate all triangles of the largest web-crawl graph *Clueweb12*[2], emitting more than three trillions of triangles within 10^3 minutes with 120 workers. PSE, with the 4-clique query, is able to discover 0.27 quadrillions of such subgraphs within 10^4 minutes with 120 workers, from the *SubDomain*[3] dataset. D4GE enumerates an astronomical 3-quadrillions 4-node graphlets from the *Indochina*[4] dataset, requiring 672 workers and 7×10^3 minutes of runtime.

[2] http://www.lemurproject.org/clueweb12/webgraph.php

[3] http://webdatacommons.org/hyperlinkgraph/

[4] http://law.di.unimi.it/webdata/indochina-2004/

We can conclude that all candidate algorithms in this work are extremely capable and performant in their respective applications. PTE_{Base}, although out-performed by its successor PTE_{CD}, inspires the invention of PSE. D4GE, while requires more computing power than the other candidates, is addressing an unprecedentedly challenging enumeration problem - that it enumerates all six-types of 4-node graphlets, and the graphlets are induced subgraphs, which are harder to enumerate [13].

6 Conclusion

Over the past decade, subgraph enumeration has been exposed under increased focuses, especially since the introduction of networks motifs [15] as an important tool for network analysis, as well as graphlets [20] which are now established measures for network alignment [22].

In this survey we explored several existing MapReduce-based methods to solve the subgraph enumeration problem of three different focuses: triangles, k-order subgraphs and 4-node graphlets. We presented the audiences with their main conceptual approaches, gave insight on their advantages and limitations, and provided a summary of their similarities and differences.

The aim of this work was to describe some of the state-of-the-art MapReduce algorithms, offering a thorough under-the-hood review. We restricted our focus in the MapReduce community because of the tried-and-true nature of MapReduce; the algorithms under this framework are often the most intuitive, practical and easy to implement.

Last but not least, we provided more than two dozens of references allowing further exploration of any aspects that might be of particular interest to the audience.

References

1. Foto N Afrati, Dimitris Fotakis, and Jeffrey D Ullman. Enumerating subgraph instances using map-reduce. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 62–73. IEEE, 2013.
2. Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.
3. Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.
4. Matthias Bröcheler, Andrea Pugliese, and Venkatramanan S Subrahmanian. Cosi: Cloud oriented subgraph identification in massive social networks. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, pages 248–255. IEEE, 2010.
5. Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009.

6. Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.
7. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters, 2008.
8. Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.
9. Katherine Faust. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks*, 32(3):221–233, 2010.
10. Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl_1):i213–i221, 2005.
11. Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. Scalable subgraph enumeration in mapreduce. *Proceedings of the VLDB Endowment*, 8(10):974–985, 2015.
12. Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.
13. Xiaozhou Liu, Yudi Santoso, Alex Thomo, and Venkatesh Srinivasan. Distributed enumeration of four node graphlets at quadrillion-scale. *SSDBM 2021: 33rd International Conference on Scientific and Statistical Database Management*, pages 85–96, 2021.
14. Tijana Milenković and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6, 2008.
15. Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
16. Ha-Myung Park and Chin-Wan Chung. An efficient mapreduce algorithm for counting triangles in a very large graph. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 539–548, 2013.
17. Ha-Myung Park, Sung-Hyon Myaeng, and U Kang. Pte: Enumerating trillion triangles on distributed systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1115–1124. ACM, 2016.
18. Ha-Myung Park, Francesco Silvestri, U. Kang, and Rasmus Pagh. Mapreduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1739–1748, 2014.
19. Ha-Myung Park, Francesco Silvestri, Rasmus Pagh, Chin-Wan Chung, Sung-Hyon Myaeng, and U Kang. Enumerating trillion subgraphs on distributed systems. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(6):1–30, 2018.
20. Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.
21. Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural networks*, 18(8):1093–1110, 2005.
22. Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: Concepts, algorithms, and applications to network motifs and graphlets. *ACM computing surveys*, 54(2):1–36, 2021.
23. Yudi Santoso, Venkatesh Srinivasan, and Alex Thomo. Efficient enumeration of four node graphlets at trillion-scale. In *23rd EDBT*, pages 439–442, 2020.
24. Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 607–614, New York, NY, USA, 2011. ACM.
25. Serene WH Wong, Nick Cercone, and Igor Jurisica. Comparative network analysis via differential graphlet communities. *Proteomics*, 15(2-3):608–617, 2015.