Distributed Enumeration of Four Node Graphlets at Quadrillion-Scale

by

Xiaozhou Liu

B.Sc. (Physics with Computer Science), University of Victoria, 2019

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Distributed Enumeration of Four Node Graphlets at Quadrillion-Scale

by

Xiaozhou Liu

B.Sc. (Physics with Computer Science), University of Victoria, 2019

Supervisory Committee

Dr. A. Thomo, Supervisor

(Department of Computer Science)

Dr. V. Srinivasan, Co-supervisor

(Department of Computer Science)

Supervisory Committee

Dr. A. Thomo, Supervisor

(Department of Computer Science)

Dr. V. Srinivasan, Co-supervisor

(Department of Computer Science)

ABSTRACT

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

Table of Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family, for being there all along the journey.

Dr. Thomo, for giving me the opportunity to pursue this degree, and for mentoring and encouraging me throughout my study.

Dr. Srinivasan, for sharing his knowledge, mentoring, and patience.

Yudi Santoso, for collaboration on parts of this project, and inspiration.

I never said that.

Lao Tzu

Chapter 1

Introduction

Mathematically, a network is a structure used to model certain relations between objects. The objects are often represented by *vertices* and relations are represented by *edges*. Two vertices are connected by an edge if they are related. The connecting edges can be *directed* if the represented relations are directional, or *undirected* if the relations hold both ways.

This mathematical abstraction can be applied to many real-life examples: a set of computers that are inter-connected by-wire or wirelessly form a computer network; cities and the roads in-between form a city network; molecules with physical interactions among them form a molecular network; people in a community form a social network; and many more.

A 4-node graphlet is an *induced sub-graph* of four vertices which are connected by edges. Given a graph, there are many 4-node graphlets inside it. Some graph problems require enumeration of these graphlets in their solution.

Distributed computing refers to the method of employing more than one computer to solve a computing task that is too massive for a single machine. The employed computers are usually linked with local network, therefore function as a homogeneous unit at a high level.

In this thesis, we study the computation problem of 4-node graphlet enumeration in a distributed setting.

1.1 Motivation

Finding occurrences of particular sub-graphs in a network/graph is a tool for analyzing and understanding such networks/graphs. Such networks/graphs have many applications in various fields: in biology [12, 15] chemistry [9, 25], social study [10, 6], network analysis and classification [31], and more. Here we focus on graphlets, which are defined as small *induced* sub-graphs. Furthermore, we are only interested in connected graphlets, hence throughout this paper graphlet is defined as a small induced connected sub-graph. Some applications require the enumeration of all graphlets up to a certain degree. For example, Milenkovic and Przulj [15] used 2, 3, 4, and 5 node graphlets to analyse Protein-Protein-Interaction (PPI) networks.

Graphlet enumeration problem is challenging when the size of the input graph is large. In fact, until the recent work [26] (for single machine), it was believed that sub-graphs beyond three nodes are difficult to enumerate, and that an enumeration algorithm, which has to touch each sub-graph, cannot terminate in a reasonable time [23]. The computational complexity grows exponentially on the order of sub-graphs that we want to enumerate. This can be understood combinatorially. Suppose we want to find sub-graphs of k nodes in a graph of n nodes, then there are $\binom{n}{k} = n(n-1)\dots(n-k+1)$ possible combinations that we need to check. If $n \gg k$, this is approximately n^k. With a graph of a million nodes, each increment of the sub-graph order, k, would cost a million times more in the computation. On the other hand, an order of magnitude increase in n would increase the complexity by 10^k. Of course, in practice, not all combinations need to be checked. Efficient algorithms were built by minimizing unnecessary checking. Nonetheless, it is generally true that the number

of sub-graphs grows rapidly with the size of the graph.

From this perspective, triangle is a special case. It is small enough to enumerate and yet has important role in graph analysis including computing the clustering coefficient [17] and truss decomposition [30]. The best known solution for triangle enumeration can process a graph of five billion nodes [19]. Four-node sub-graph enumeration is already challenging. Known solutions are limited in term of scalability, or the size of the graph that can be processed in a reasonable amount of time. Counting (either exact [11, 23] or approximate [24, 5]) can do more, but we focus on enumeration which is a more challenging problem.

Induced sub-graphs are more difficult to enumerate than the *non-induced* ones, because for induced sub-graphs we need to check all possible connections as well as non-connections among the nodes. Recently, an efficient algorithm for four-node graphlets enumeration had been proposed in [26]. It has a run time which is much better than $O(nd^{k-1})$, where d is the maximum degree. Moreover, it enumerates all types of four-node graphlets in a single run. This is different from other solutions which do one pattern at a time. Nonetheless, it is designed for a single machine, hence restricting its scalability.

Using a distributed platform is an obvious option to increase the computing power, where we can add more compute nodes to get more done within a given time budget. Known distributed solutions have employed *MapReduce* paradigm [7, 29, 18, 21], yet these algorithms had limited scalability, and were only designed to enumerate *triangle* sub-graphs. The work of Park et al. improved upon existing solutions and was able to enumerate triangles from graphs with billions edges [19]; the same authors generalized their existing work to remove the triangle constraint, supporting arbitrary sub-graph query [22]. However, despite considered as the *state-of-the-art* (SotA) distributed enumeration algorithm, Park et al.'s work [22] only enumerates

non-induced sub-graphs, which is far-less computational demanding than induced; and even for non-induced sub-graphs, their algorithm can barely process billion-edge graphs up to 4-node sub-graph queries. In addition, bringing a single machine solution to a distributed platform has its own challenge. If it is not done properly, we will get a poor scalability which would not justify the economical cost of the distributed platform [14].

This motivates us to search for methods to bring Santoso et al.'s algorithm [26] onto a distributed platform, such that the solution is optimal for this particular problem.

1.2 Contributions

Our contributions are summarized as follows:

1. We devised an efficient distributed algorithm for enumerating *all* induced 4-node graphlets on a single run. This includes optimizations that are particular to this problem. We provide detailed analyses on this algorithm to prove its correctness and efficiency.

2. We modified the single machine algorithm for 4-node graphlet enumeration so that it can be deployed in a distributed setting.

3. We implemented of our proposed solution in Apache Spark.

4. We did extensive experimentation with it on several massive graphs, and we compare our code with the SotA. The results show that our solution has better performance compared to the SotA. We succeeded in processing a larger graph of a size that had never been processed before in a reasonable amount of time, enumerating quadrillions of graphlets using a modest cluster of machines.

5. We also built a simpler algorithm that solely enumerates 4-cliques, and tuned it to be used in the distributed settings.

6. We investigate how to further improve the algorithm suite.

The results of this work are published in 33rd International Conference on Scientific and Statistical Database Management[1]: dblp.org/rec/conf/ssdbm/LiuS0T21.

1.3 Organization

This thesis is organized as follows:

- **Chapter 2 Preliminaries** presents the notation and definitions on graphs used in this work, as well as other existing sub-graph enumeration algorithms and solutions directly or indirectly related to this work.
- **Chapter 3 Algorithms and Implementations** details our proposed algorithm supported with analysis and implementations.
- **Chapter 4 Experiments** presents our experimental settings and results.
- **Chapter 5 Discussions** offers discussions on our results.
- **Chapter 6 Conclusions and Future Work** concludes the thesis and offers insights on how to improve the algorithm further.

[1] https://ssdbm.org/2021/accepted-papers

Chapter 2

Preliminaries

This chapter lays out the necessary definitions and backgrounds used in the thesis, and introduces a family of algorithms related to sub-graph enumeration in both serial and distributed settings. The backgrounds and definitions are presented in Section 2.1. The algorithms in Section 2.3 and 2.4 have directly influenced this work, whereas other related algorithms are listed in Section 2.2.

2.1 Definitions of Graph and Partitions

Graph. A *graph* is a structure amounting to a set of objects in which some pairs of the objects are in some sense 'related'. The set of objects are called **vertices**, denoted by V; the set of relations are called **edges**, denoted by E. The graph is denoted by $G(V, E)$. The number of vertices of the graph is denoted by $n = |V|$; the number of edges of the graph is denoted by $m = |E|$.

Neighbour. Vertex v is a *neighbour* of vertex u if u and v are connected by an edge. The set of all neighbours of vertex u is denoted by $N(u)$.

Degree. The *degree* of vertex u is the number of neighbours of vertex u, denoted by $d(u) = |N(u)|$.

Undirected Graph. In this work, unless specified otherwise, edges of a graph are non-directional by default. Such graphs are *undirected*. The undirected edge connecting vertex u and vertex v is denoted by (u, v).

Symbols	Definitions
$G(V, E)$	Undirected graph.
$\overrightarrow{G}(V, \overrightarrow{E})$	Directed graph.
$G^{sym}(V, E^{sym})$	Symmetrized graph.
V	The set of vertices.
E	The set of edges.
n	The number of vertices. $n = \|V\|$.
m	The number of edges. $m = \|E\|$.
u, v, w, z	Reserved for vertices.
$N(u)$	The neighbours of vertex u.
$N^+(u)$	The out-neighbours of vertex u.
$d(u)$	The degree of vertex u.
$d^+(u)$	The out-degree of vertex u.
(u, v)	An undirected edge between u and v.
$\overrightarrow{(u, v)}$	A directed edge from u to v.
$(u, v, w)_3$	A 3-node graphlet of vertices u, v and w.
$(u, v, w, z)_4$	A 4-node graphlet of vertices u, v, w and z.
η	The edge-orienting function.
ρ	The number of colors.
θ	The coloring function.
i, j, k, l	Reserved for colors.
(i, j)	The color of an edge.
(i, j, k)	The color of a 3-node graphlet.
(i, j, k, l)	The color of a 4-node graphlet.
E_{ij}	Edge-set of color i and j.
E^*_{ij}	Directed edge-set of color i and j.
S_{ijk}	Sub-problem of color i, j and k.

Table 2.1: Notations used in this thesis.

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

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

Directed Graph. A graph whose edges are directional is called a *directed graph*, denoted by $\overrightarrow{G}(V, \overrightarrow{E})$. The directed edge originates from vertex u to vertex v is denoted by $\overrightarrow{(u, v)}$. The out-going neighbours of vertex u is denoted by $N^+(u)$; the out-degree of vertex u is denoted by $d^+(u)$.

Directed Acyclic Graph. A *directed acyclic graph* (DAG) is a special case of a directed graph where none of its sub-graphs is a *cycle*. A *cycle* is a directed sub-graph of the original graph where the traversal from any vertex of the directed sub-graph eventually ends at the starting vertex.

Clique. A *clique* is a graph where all the vertices are connected. It is also known as a *complete graph*.

Graphlet. A graphlet is an induced sub-graph. There are two types of 3-node graphlets: wedge and triangle as shown in Fig. 2.1, and six types of 4-node graphlets, shown in Fig. 2.2: 3-path, 3-star, rectangle, tailed-triangle, diamond and 4-clique.

Figure 2.1: Three node graphlets: a wedge and a triangle.

Edge-orientation. Edge-orientation refers to assigning directions to edges in an undirected graph $G(V, E)$. First, define a function η which determines a total ordering of the nodes in V. An undirected edge $(u, v) = (v, u)$ is orientated by η,

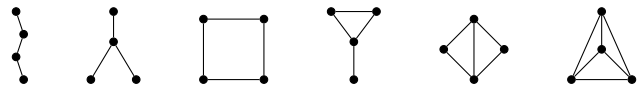


Figure 2.2: Four node graphlets: a 3-path, a 3-star, a rectangle or 4-cycle, a tailed-triangle, a diamond, and a 4-clique.

such that if $\eta(u) < \eta(v)$, we list only (u, v) but not (v, u). This oriented edge is then denoted by $\overrightarrow{(u, v)}$. In this work, we use the degrees of the vertices to define the total ordering η, that is, if $d(u) < d(v)$ then $\eta(u) < \eta(v)$. Vertices labels are used to break any ties. We call this fashion of edge-orientation **orientation-by-degree**.

Coloring. Coloring refers to a technique of applying a function to each edge such that the entire graph can be partitioned into many sub-graphs. First, define a function θ which maps vertex $u \in V$ to $\theta(u)$. We say that vertex u has *color* $\theta(u)$. We also say that edge $(u, v) \in E$ has color (i, j) where u, v have colors i and j respectively. In this thesis, we choose $\theta(u) = u\%\rho$ where ρ is a user-specified constant - the *number of colors*.

Color Assignment. The color representation of a graphlet. For example, we say a graphlet (u, v, w, z) has a color assignment $(\theta(u), \theta(v), \theta(w), \theta(z))$ after applying color function θ.

Edge-set. If coloring is applied to an unirected graph, edges with the same colors can be grouped together to form a sub-graph of $G(V, E)$. Such sub-graph is called an *edge-set*, denoted by E_{ij}, where i and j are the colors of the endpoints of all the edges.

Directed Edge-set. If coloring is applied to a directed graph, the resulting edge-sets are called directed edge-set, 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. Notice that the directed edge-set E^*_{ij} can be seen as a subset of the edge-set E_{ij}: for $i \neq j$, $E^*_{ij} \cup E^*_{ji} = E_{ij}$; for $i = j$, $E^*_{ij} = E_{ij}$.

Sub-problem. A sub-problem refers to the union of edge-sets of particular colors where the distributed workers can discover all the graphlets in that union independently. It is the fundamental enumeration task assigned to each distributed worker. In this thesis, the idea of sub-problems is adapted to three different distributed schemes: two algorithms from existing literature and an algorithm that we propose. More detailed explanations of the various sub-problem compositions are in Chapter 2.4.1, Chapter 2.4.3 and Chapter 3.

All three schemes share the same sub-problem definition. For a k-order graphlet enumeration task, we denote a sub-problem by $S_{\{c_0,c_1,...,c_l\}}$ where $|\{c_0, c_1, ..., c_l\}| \in \{1, 2, ...k\}$ and $c_l \in \{0, 1, ...\rho - 1\}$. For example, for $\rho = 3$ and $k = 3$ (3-node graphlets), the sub-problems are: S_0, S_1, S_2, S_{01}, S_{02}, S_{12} and S_{012}. Here $S_0 = E_{00}$, $S_1 = E_{11}$, $S_2 = E_{22}$, $S_{01} = E_{00} \cup E_{01} \cup E_{11}$, $S_{02} = E_{00} \cup E_{02} \cup E_{22}$, $S_{12} = E_{11} \cup E_{12} \cup E_{22}$ and $S_{012} = E_{01} \cup E_{02} \cup E_{12}$.

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

2.2 Related Algorithms

Substantial amount of work has been devoted into sub-graph enumeration research. Albeit the existing literature listed below do not directly contribute to this thesis, all the algorithms here address similar and related problems, thus their significance.

Yu et al. have first migrated PTE algorithms suite ([19], a distributed triangle enumeration algorithm, to be introduced in Section 2.4) onto a serverless setting [32].

Yu et al. explored the scalability limit with respect the the number of tasks of PTE, and the trade-off between the time and cost of triangle enumeration, by employing AWS Lambda platform, which can provide extremely high concurrency. Through experimental results, Yu et al. revealed the importance of using a higher number of tasks in order to improve the efficiency of PTE. This fact is also discovered and experimented by us in this work, and will be discussed in Section 4.3.

Inspired by PTE_{CD} (one of the algorithms of PTE suite), Ahmed et al. carefully engineered a triangle enumeration solution in the relational database management system (RDBMS) setting [1]. Ahmed et al. showed that their solution was capable of enumerating billions of triangles from billion-edge graphs, and was orders of magnitude faster than the competitors.

Singh et al. focused on solving the triangle counting problem in a different computational model: the Edge Stream [28]. The authors first categorizes the existing literature into two categories: fixed memory (FM) and fixed probability (FP), and identified the limitations of both categories. Singh et al. then proposed their solution to the problem using two hybrid approaches: Neighbourhood Hybrid Multisampling (NHMS) and Triest/ThinkD Hybrid Sampling (THS), inheriting the strengths of FM and FP. Both proposed algorithms were proven to out-perform the SotA from existing literature in a space-limited computing environment.

Santoso et al. designed their prominent 4-node graphlet enumeration algorithm (Section 2.3.2) building on their earlier triad enumeration algorithm [27]. Started with a modified version of the previously SotA intersection-based algorithm, Santoso et al. quickly identified the drawbacks of the compact data structure used underneath; the authors then proposed a brand-new solution, Four Pointer Enumeration (4P), to the problem, employing both the idea of *edge-orientation* and conscientious listing of the directional edges from the original graph and the transpose. Santoso et al.'s results

were phenomenal - the 4P algorithm was able to enumerate, exactly, all the triads in the largest graph available on the Internet - *ClueWeb12*, which consists of one billion vertices and forty-three billions of edges.

Lastly, we want to draw the attention of the readers that while the later-proposed algorithm of this thesis diverges from all the existing work, it shall not be considered as an isolated attempt to solving a unique problem - the contribution of this thesis should be evaluated from a broader view point, that it might inspire future work in the grand picture of sub-graph enumeration.

2.3 Serial Graph Enumerations

2.3.1 Triangle Enumeration

Compact-Forward algorithm [13] is the *de-facto* serial algorithm for triangle enumeration. Compact-Forward requires that the input graph is oriented-by-degree: the input graph $G(V, E)$ must be pre-processed into a DAG $\overrightarrow{G}(V, \overrightarrow{E})$ such that $\forall(u, v) \in E$, orientations are assigned where $\forall\overrightarrow{(u, v)} \in \overrightarrow{E}$ must satisfy $d^+(u) > d^+(v)$. Compact-Forward enumerates all triangles by intersecting the outgoing neighborhoods of the two end-points of each edge. Compact-Forward requires $O(|\overrightarrow{E}|^{3/2})$ operations. The pseudo code of Compact-Foward is given as Algorithm 1 below.

Algorithm 1 COMPACT-FORWARD

Require: An oriented-by-degree graph $\overrightarrow{G}(V, \overrightarrow{E})$

1: **for all** $(\overrightarrow{u, v}) \in \overrightarrow{E}$ **do**
2: **for all** $w \in N^+(u) \cap N^+(v)$ **do**
3: ENUMERATE (u, v, w)

Orientating-by-degree is the necessary condition for Compact-Forward to achieve the claimed efficiency of $O(m^{3/2})$. It effectively redistributes the degree of each vertex

without changing the topology of the original graph, guaranteeing the maximum degree of each vertex is bounded by $O(m^{1/2})$. The details are documented in [13].

2.3.2 4-node Graphlet Enumeration

A single machine algorithm, *Simultaneous 4-node Graphlets Enumeration* (S4GE), for enumerating all six types of 4-node graphlets in a single run was introduced in [26]. The algorithm first discovers triangles and wedges in a similar fashion as Compact-Forward, 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 finds, 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 [26]. 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 2; the pseudo code of S4GE for discovering all the six types of 4-node graphlets are listed as Algorithms 3, 4, and 5.

Algorithm 2 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 3 EXPLORE TRIANGLE

Require: Triangle $(u, v, w)_3$ with $u < v < w$; symmetrized neighbourhood $N(u)$, $N(v)$ and $N(w)$.

$N^{>u}(u) \equiv \{z | z \in N(u), \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z | z \in N(v), \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z | z \in N(w), \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(u) \cap N^{>u}(v) \cap N^{>u}(w)$ with $z > w$ **do**
    ENUMERATE4CLIQUE (u, v, w, z)
for all z in two sets and $z >$ opposite vertex **do**
    ENUMERATEDIAMOND (u, v, w, z)
for all z in one set only **do**
    ENUMERATETAILEDTRIANGLE (u, v, w, z)

Algorithm 4 EXPLORE WEDGE TYPE-1

Require: Wedge $(v, u, w)_1$ with $u < v < w$; symmetrized neighbourhood $N(u)$, $N(v)$ and $N(w)$.

$N^{>u}(u) \equiv \{z | z \in N(u), \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z | z \in N(v), \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z | z \in N(w), \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ **do**
    ENUMERATERECTANGLE (u, v, z, w)
for all $z \in N^{>u}(u)$ only **do**
    if $z > w$ **then**
        ENUMERATE3STAR (u, v, w, z)
for all $z \in N^{>u}(v)$ only **do**
    ENUMERATE3PATH (w, u, v, z)
for all $z \in N^{>u}(w)$ only **do**
    ENUMERATE3PATH (v, u, w, z)

Algorithm 5 EXPLORE WEDGE TYPE-2

Require: Wedge $(u, v, w)_2$ with $u < v$ and $u < w$; symmetrized neighbourhood $N(v)$ and $N(w)$.

1: $N^{>u}(v) \equiv \{z | z \in N(v), \eta(z) > \eta(u)\}$
2: $N^{>u}(w) \equiv \{z | z \in N(w), \eta(z) > \eta(u)\}$
3: **for all** $z \in N^{>u}(v)$ only **do**
4: **if** $z > w$ **then**
5: ENUMERATE3STAR (v, u, w, z)
6: **for all** $z \in N^{>u}(w)$ only **do**
7: **if** $z \neq v$ **then**
8: ENUMERATE3PATH (u, v, w, z)

S4GE algorithm requires the undirected input graph to be symmetrized. This is the precondition for the correctness of the algorithm. Consider the following DAG:

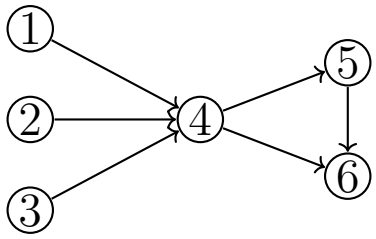


Figure 2.3: A graph illustrating the need for symmetrization.

It has an 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 unable to discover tailed-triangle $(1, 4, 5, 6)_4$, $(2, 4, 5, 6)_4$ and $(3, 4, 5, 6)_4$, as vertices 1, 2, 3 are not the neighbours of vertex 4. With symmetrization applied, 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.

It is also worth noting that, while S4GE's input graph is oriented-by-degree, the

edge-orientation here is not as effective as one applied in Compact-Forward. The runtime of S4GE, as derived by Santoso et al. [26], is proportional to d_{max}^{sym}, and d_{max}^{sym} is an intrinsic property of the graph itself, which cannot be reduced by edge-orientation (edge-orientation does not alter the topology of the graph). In Chapter 4 we will show that the runtime of S4GE is heavily affected by d_{max}^{sym}. The orientation-by-degree still redistributes the degrees information over the vertices, but it only affects the discovery of the triangles of S4GE [26].

2.4 Distributed Sub-graph Enumeration

2.4.1 PTE_{Base}

Park et al. [19] proposed three variants of distributed algorithms for triangle enumeration, and PTE_{Base} is the base version. PTE_{Base} partitions the undirected input graph into edge-sets 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 edge-sets that are necessary to solve the sub-problem.

PTE_{Base} defines three types of triangles: type-1 triangles being the ones with all three vertices of the same color; type-2 being the ones with two vertices of one color and another vertex of a different color; type-3 being the ones with all three vertices of different colors. The types of triangles also correspond to the types of sub-problems. For example, all type-1 triangles can be enumerated from a sub-problem S_i, where $i \in \{0, 1, \ldots, \rho - 1\}$, thus S_i is called a type-1 sub-problem.

PTE_{Base} partitions the input graph into edge-sets, E_{ij} for $(i, j) \in \{0, 1, \ldots, \rho-1\}^2$. For $i == j$ there are $\binom{\rho}{1}$ of such edge-sets 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 edge-set

$E_{(u\%\rho)(v\%\rho)}$.

PTE_{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 not only embedded 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_{Base} employs Compact-Forward algorithm as the local serial algorithm. Because PTE_{Base} partitions the input graph into $O(\rho^2)$ colored edge-sets, each sub-problem is expected to process a sub-graph of size $O(m/\rho^2)$, 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_{Base} does $O(m^{1.5})$ amount of work overall, which is the same amount of work as applying CompactForward on a single machine. The network read for PTE_{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_{Base} is given as Algorithm 6:

Algorithm 6 PTE_{Base}

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

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 ρ colors
3: Generate all sub-problems $\{S\}$ for the given ρ
4: **for all** sub-problem $S_{\text{Cs}} \in \{S\}$ **do** ▷ Distributed-for
5: E'=ReadEdgeSets(Cs, 3)
6: Compact-Forward (E')

Each distributed worker reconstructs the sub-graph defined by the sub-problem by reading edge-sets from the distributed storage. The algorithm for reading the edge sets is given in Algorithm 7.

Algorithm 7 ReadEdgeSets

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

```
Initialize empty edge-set E'
for all (i, j) ∈ Cs^2 do
    if i = j and |{c_0, c_1, ..., c_l}| ≠ k then
        E' ← E' ∪ E_{i,i}
    if i < j then
        E' ← E' ∪ E_{i,j}
return E'
```

2.4.2 PTE_{CD}

Park et al. [19] improve on PTE_{Base} by introducing PTE_{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_{Base} does extra work when intersecting $\overrightarrow{N}(u)$ and $\overrightarrow{N}(v)$. When intersecting, CompactForward algorithm used in PTE_{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_{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_{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_{CD} lists all six cases for any type-2 sub-problem as well.

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

The pseudo code of PTE_{CD} is given as Algorithm 8 and Algorithm 9:

Algorithm 8 PTE_{CD}

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

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 ρ
3: Generate all sub-problems $\{S\}$ given ρ
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$_{\text{CD}}$ $(E^*_{ii}, E^*_{ii}, E^*_{ii})$ ▷ 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$_{\text{CD}}$ $(E^*_{pq}, E^*_{pr}, E^*_{qr})$ ▷ 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$_{\text{CD}}$ $(E^*_{pq}, E^*_{pr}, E^*_{qr})$ ▷ Type-3

Algorithm 9 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)

2.4.3 PSE

Park et al. generalise PTE_{Base} to support non-induced sub-graph query of an arbitrary order, called PSE (Pre-partitioned Sub-graph Enumeration) [22]. PSE takes a query sub-graph $G_q(V_q, E_q)$ of order k as input where $k = |V_q|$, and enumerates all the matching sub-graphs to G_q. PSE employs VF2 algorithm [8] 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 sub-graphs from S_{012} also enumerates all the sub-graphs 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 called 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 dominate 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 sub-problems to the existing 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 assigned sub-problem is dominant within that group. Ties are broken randomly.

3. Repeat Step 2 until all sub-problems are assigned.

In the example of $k = 4$ and $\rho = 4$, one possible group assignment is listed below, with the first sub-problem of each group being the dominant sub-problem:

$$\{S_{012}, S_0, S_1, S_2\},$$
$$\{S_{013}, S_{01}, S_{03}\},$$
$$\{S_{023}, S_{02}, S_{23}, S_3\},$$
$$\{S_{123}, S_{12}, S_{13}\},$$
$$\{S_{0123}\}$$

The pseudo code of PSE is given in Algorithm 10:

Algorithm 10 PSE

Require: An undirected graph $G(V, E)$; a query graph $G_q(V_q, E_q)$; the number of colors ρ

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 ρ
3: Generate sub-problem groups $\{SGs\}$
4: **for all** sub-problem group $SG \in \{SGs\}$ **do** ▷ Distributed-for
5: Dominant sub-problem S_d=$SG[0]$
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. show that PSE requires at most $\binom{\rho-1}{k-2} * |E|$ amount of network read, for k-order sub-graph query with input graph E.

Chapter 3

Algorithms and Implementations

In this chapter we list the algorithms that we propose, and give the detailed analysis of them. In the later part, we explain our implementations. We re-state that our goal is to enumerate all 4-node graphlets in a distributed setting, as fast as possible.

3.1 Duplication in PSE

We started experimenting, by deploying S4GE [26] algorithm with PSE partitioning scheme; however, while correctly enumerating all 4-node graphlets, this combination discovers certain sub-graphs more than once, and the issue lies within PSE partitioning scheme itself. Consider the the example of $k = 4$ and $\rho = 4$, one possible group assignment is:

$$\{S_{012}, S_0, S_1, S_2\},$$

$$\{S_{013}, S_{01}, S_{03}\},$$

$$\{S_{023}, S_{02}, S_{23}, S_3\},$$

$$\{S_{123}, S_{12}, S_{13}\},$$

$$\{S_{0123}\}$$

Let us assume there is a 4-node sub-graph (u, v, w, z) in the input graph that matches the query, and, without losing the generality, this sub-graph has color $(0, 0, 1, 1)$. To discover this particular sub-graph, by definition, we require knowledge of edge-sets $E_{00} \cup E_{01} \cup E_{11}$ - all the edges between the 0-colored vertices, between the 1-colored vertices, and across 0-colored and 1-colored vertices. This edge-sets union exists in two sub-problem groups:

1. Group $\{S_{012}, S_0, S_1, S_2\}$, since the group is dominated by the sub-problem $S_{012} = E_{00} \cup E_{11} \cup E_{22} \cup E_{01} \cup E_{02} \cup E_{12}$. $E_{00} \cup E_{11}$, and $(E_{00} \cup E_{01} \cup E_{11}) \subset S_{012}$;

2. Group $\{S_{013}, S_{01}, S_{03}\}$, since the group is dominated by the sub-problem $S_{013} = E_{00} \cup E_{11} \cup E_{33} \cup E_{01} \cup E_{03} \cup E_{13}$, and $(E_{00} \cup E_{01} \cup E_{11}) \subset S_{013}$.

Since both groups contain the same exact edge-set union, if there exists any 4-node sub-graph (u, v, w, z) of color $(0, 0, 1, 1)$ that requires knowledge of $E_{00} \cup E_{01} \cup E_{11}$, it will be discovered by both groups. The duplication is resulted from the fact that the sub-problem groups are overlapping each other by definition. Despite of the query graph and the number of colors, duplications can always be expected from PSE. In Chapter 4.7 we will present the percentage duplication when using PSE with different serial algorithms.

To the best of our knowledge, Park et al. [22] post-filters out the duplicates by checking the sub-problem that represents the discovered sub-graph. With the same example, the 4-node sub-graph (u, v, w, z) of color $(0, 0, 1, 1)$ is discovered from $E_{00} \cup E_{01} \cup E_{11}$ belonging to sub-problem S_{01}, hence (u, v, w, z) should be discovered from the second group, and filtered out from the first group. This is implemented by line 8 of Algorithm 10. The detail of this post-filtering process is omitted from the original literature [22], and was only discovered from the implementation[1].

[1] https://datalab.snu.ac.kr/pegasusn/download.php

This motivates us to revise a duplication-free partition scheme to truly empower the serial S4GE algorithm on a distributed setting.

3.2 Generalized Color-Direction

3.2.1 Edge-orientation Revisit

Before we propose our improved algorithm that guarantees no duplication, let us revisit the nature of edge-orientation, and how edge-orientation helps to avoid duplicated enumeration.

Recall that edge-orientation refers to the technique that turns an undirected graph $G(V, E)$ into a directed one $\overrightarrow{G}(V, \overrightarrow{E})$, by assigning a deterministic ordering on all edges $\in E$. In Section 2.3.1 we have seen that edge-orientation is a crucial step in the performance of Compact-Forward algorithm. In addition, edge-orientation serves a more important role in graph enumeration in general: it ensures the uniqueness of the listings of the same graph.

Consider the following graph:

(a) Original graph (b) After edge-orientation

Figure 3.1: Two graphs illustrating triangle enumeration.

On the left we have an undirected graph. It is obvious that there is only one triangle in the left sub-graph. However, the answer might be ambiguous if asking different enumeration algorithms to list all the triangles. Because in fact, there are six possible listings of the same triangle: $(1, 2, 3)_3, (1, 3, 2)_3, (2, 1, 3)_3, (2, 3, 1)_3, (3, 1, 2)_3$ and

$(3, 2, 1)_3$. Which listing to emit might depend on the order of vertices being visited; if not designed carefully, it is even possible for an algorithm to emit all six variations.

However, if we apply edge-orientation to the same graph, the ambiguity in enumeration is completely void. Assume we apply edge-orientation to Figure 3.1a: edge $\overrightarrow{(u, v)}$ exists if and only if $u < v$. The resulting directed graph is illustrated as Figure 3.1b. If we tweak our query, instead of asking the number of triangles, we ask the number of triangles (u, v, w) that satisfies $u < v < w$, we can then guarantee that there in only one listing emitted by the enumeration algorithm, which is $(1, 2, 3)_3$. The constraint on u, v, w can be any order, and the uniqueness of enumeration is always preserved.

The key takeaway is that the edge-orientation here not only assigns ordering to the edges, but also derives linearity of the target graphs, and the linearity can be leveraged to guarantee uniqueness. The idea of order and linearization is the key to avoid duplication.

3.2.2 color-assignment

In this section we will present how we leverage the idea of order and linearization to completely eliminate the duplication, yet enumerate the correct number of subgraphs.

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

Next, we assert linearity on the 4-node graphlets query, 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 $\eta(u) < \eta(v) < \eta(w) < \eta(z)$. While the mapping between the graphlet to color-assignment is many-to-one, both the graphlets and the color-assignments follow the same ordering. Given the number of color ρ for 4-node graphlet enumeration, how many ordered color-assignments exist? The answer is ρ^4: the exact number of all possible combinations of $i \prec j \prec k \prec l$, $\forall\ i, j, k, l \in \{0, 1, \ldots, \rho - 1\}$.

The above formulates the fundamental idea of our proposed distributed algorithm: the distributed algorithm first enumerates all ordered ρ^4 color-assignments $\{K_{ijkl}, \ldots\}$, and for each color-assignment K_{ijkl}, a serial algorithm is employed to enumerate all the 4-node graphlets.

3.3 Directed 4-node Graphlet Enumeration

We call our scheme, applied to four-node graphlets, as Distributed 4-node Graphlet Enumeration (D4GE). The pseudo code of D4GE is given as Algorithms 11, 12, 13 and 14. We want to stress that while previous work employed the idea of color direction to reduce the amount of work, we differentiate ourselves by exploiting both the linearity of the DAG and the color-assignment and are able to exploit the fact that the color-assignment problem is essentially a combination problem, and the unique relationship between any sub-graph and its color-assignment guarantees the duplication-free nature of our algorithm. In addition, previous works explicitly list

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 11 D4GE

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

1: Construct $\overrightarrow{G}(V, \overrightarrow{E})$ by applying edge-orientation to $G(V, E)$ ▷ Distributedly
2: Symmetrise $\overrightarrow{G}(V, \overrightarrow{E})$ into $G^{\text{sym}}(V, E^{\text{sym}})$
3: PartitionEdgeset(E^{sym}, ρ)
4: Generate sub-problems $S_{ij} \cup S_{ijk} \cup S_{ijkl}$.
5: **for all** $S_{\text{Cs}} \in S_{ij} \cup S_{ijk} \cup S_{ijkl}$ **do** ▷ Distributed-for
6: $K_s \leftarrow$GroupColorAssignments(S_{Cs}, ρ)
7: $E_{\text{map}} \leftarrow$ ReadEdgeSets$_{\text{CD}}$ $(S_{\text{Cs}}, 4)$
8: **for all** $K_{ijkl} \in K_s$ **do**
9: S4GE$_{\text{CD}}$ $(E_{\text{map}}, ijkl)$

Algorithm 12 PartitionEdgeset

Require: Symmetrized edges E^{sym}; number of colors ρ.

1: **for all** $\overrightarrow{(u, v)} \in E^{\text{sym}}$ **do** ▷ Distributed-for
2: $i \leftarrow u\%\rho$, $j \leftarrow v\%\rho$;
3: Initialize empty edge-set E^*_{ij} if not exists.
4: Append $\overrightarrow{(u, v)}$ into E^*_{ij}.

Algorithm 13 GroupColorAssignments

Require: Sub-problem S_{Cs} with $Cs = \{c_0, c_1, ..., c_l\}$; number of color ρ.

Initialize empty color-assignments array K_s.
for all $K_{ijkl} \in \rho^4$ permutations **do**
    $c \leftarrow sort(reduce(\{i, j, k, l\}))$
    if $c = C_s$ **then**
        Append K_{ijkl} to K_s.
return K_s.

Algorithm 14 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
        $E_{\text{map}}[(i, j)] \leftarrow E^*_{ij}$
return E_{map}

Here we introduce the grouping strategy to form sub-problems from ordered color-assignments (Algorithm 13). 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 inherently 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 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 re-

duced form of $\{2, 0, 1, 0\}$ is $\{0, 1, 2\}$. Therefore K_{2010} belongs to S_{012}. An example of color-assignments grouping is shown in Table 3.1.

Sub-problems	color-assignments
(0, 1)	(0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), (0,1,0,1), (0,1,1,0), (0,1,1,1), (1,0,0,0), (1,0,0,1), (1,0,1,0), (1,0,1,1), (1,1,0,0), (1,1,0,1), (1,1,1,0)
(0, 2)	(0,0,0,2), (0,0,2,0), (0,0,2,2), (0,2,0,0), (0,2,0,2), (0,2,2,0), (0,2,2,2), (2,0,0,0), (2,0,0,2), (2,0,2,0), (2,0,2,2), (2,2,0,0), (2,2,0,2), (2,2,2,0)
(0, 3)	(0,0,0,3), (0,0,3,0), (0,0,3,3), (0,3,0,0), (0,3,0,3), (0,3,3,0), (0,3,3,3), (3,0,0,0), (3,0,0,3), (3,0,3,0), (3,0,3,3), (3,3,0,0), (3,3,0,3), (3,3,3,0), (3,3,3,3)
(1, 2)	(1,1,1,1), (1,1,1,2), (1,1,2,1), (1,1,2,2), (1,2,1,1), (1,2,1,2), (1,2,2,1), (1,2,2,2), (2,1,1,1), (2,1,1,2), (2,1,2,1), (2,1,2,2), (2,2,1,1), (2,2,1,2), (2,2,2,1)
(1, 3)	(1,1,1,3), (1,1,3,1), (1,1,3,3), (1,3,1,1), (1,3,1,3), (1,3,3,1), (1,3,3,3), (3,1,1,1), (3,1,1,3), (3,1,3,1), (3,1,3,3), (3,3,1,1), (3,3,1,3), (3,3,3,1)
(2, 3)	(2,2,2,2), (2,2,2,3), (2,2,3,2), (2,2,3,3), (2,3,2,2), (2,3,2,3), (2,3,3,2), (2,3,3,3), (3,2,2,2), (3,2,2,3), (3,2,3,2), (3,2,3,3), (3,3,2,2), (3,3,2,3), (3,3,3,2)
(0, 1, 2)	(0,0,1,2), (0,0,2,1), (0,1,0,2), (0,1,1,2), (0,1,2,0), (0,1,2,1), (0,1,2,2), (0,2,0,1), (0,2,1,0), (0,2,1,1), (0,2,1,2), (0,2,2,1), (1,0,0,2), (1,0,1,2), (1,0,2,0), (1,0,2,1), (1,0,2,2), (1,1,0,2), (1,1,2,0), (1,2,0,0), (1,2,0,1), (1,2,0,2), (1,2,1,0), (1,2,2,0), (2,0,0,1), (2,0,1,0), (2,0,1,1), (2,0,1,2), (2,0,2,1), (2,1,0,0), (2,1,0,1), (2,1,0,2), (2,1,1,0), (2,1,2,0), (2,2,0,1), (2,2,1,0)
(0, 1, 3)	(0,0,1,3), (0,0,3,1), (0,1,0,3), (0,1,1,3), (0,1,3,0), (0,1,3,1), (0,1,3,3), (0,3,0,1), (0,3,1,0), (0,3,1,1), (0,3,1,3), (0,3,3,1), (1,0,0,3), (1,0,1,3), (1,0,3,0), (1,0,3,1), (1,0,3,3), (1,1,0,3), (1,1,3,0), (1,3,0,0), (1,3,0,1), (1,3,0,3), (1,3,1,0), (1,3,3,0), (3,0,0,1), (3,0,1,0), (3,0,1,1), (3,0,1,3), (3,0,3,1), (3,1,0,0), (3,1,0,1), (3,1,0,3), (3,1,1,0), (3,1,3,0), (3,3,0,1), (3,3,1,0)
(0, 2, 3)	(0,0,2,3), (0,0,3,2), (0,2,0,3), (0,2,2,3), (0,2,3,0), (0,2,3,2), (0,2,3,3), (0,3,0,2), (0,3,2,0), (0,3,2,2), (0,3,2,3), (0,3,3,2), (2,0,0,3), (2,0,2,3), (2,0,3,0), (2,0,3,2), (2,0,3,3), (2,2,0,3), (2,2,3,0), (2,3,0,0), (2,3,0,2), (2,3,0,3), (2,3,2,0), (2,3,3,0), (3,0,0,2), (3,0,2,0), (3,0,2,2), (3,0,2,3), (3,0,3,2), (3,2,0,0), (3,2,0,2), (3,2,0,3), (3,2,2,0), (3,2,3,0), (3,3,0,2), (3,3,2,0)
(1, 2, 3)	(1,1,2,3), (1,1,3,2), (1,2,1,3), (1,2,2,3), (1,2,3,1), (1,2,3,2), (1,2,3,3), (1,3,1,2), (1,3,2,1), (1,3,2,2), (1,3,2,3), (1,3,3,2), (2,1,1,3), (2,1,2,3), (2,1,3,1), (2,1,3,2), (2,1,3,3), (2,2,1,3), (2,2,3,1), (2,3,1,1), (2,3,1,2), (2,3,1,3), (2,3,2,1), (2,3,3,1), (3,1,1,2), (3,1,2,1), (3,1,2,2), (3,1,2,3), (3,1,3,2), (3,2,1,1), (3,2,1,2), (3,2,1,3), (3,2,2,1), (3,2,3,1), (3,3,1,2), (3,3,2,1)
(0, 1, 2, 3)	(0,1,2,3), (0,1,3,2), (0,2,1,3), (0,2,3,1), (0,3,1,2), (0,3,2,1), (1,0,2,3), (1,0,3,2), (1,2,0,3), (1,2,3,0), (1,3,0,2), (1,3,2,0), (2,0,1,3), (2,0,3,1), (2,1,0,3), (2,1,3,0), (2,3,0,1), (2,3,1,0), (3,0,1,2), (3,0,2,1), (3,1,0,2), (3,1,2,0), (3,2,0,1), (3,2,1,0)

Table 3.1: An example of color-assignment grouping into sub-problems with $\rho = 4$. $4^4 = 256$ color-assignments are grouped into $\binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 11$ sub-problems.

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.

One thing to notice is that the number of color-assignments under each sub-problems are different. As shown in Table 3.1, the sub-problem (i, j, k) has more color-assignments associated compared to (i, j). This means that, if we naively assign each sub-problem to one distributed worker, the worker with smaller sub-problem will finish execution faster than the one with larger sub-problem, creating an uneven workload distribution. We will present our solution to the workload distribution in Section 4.3.

Consider all the ordered color-assignments K_{pqrs} under S_{ijkl}, there are only two possible relative orders of two arbitrary colors p and q: p precedes q or the reverse. This means that for an arbitrary sub-problem S_{ijkl}, for any two colors p and q from $\{i, j, k, l\}$, we need $E^*_{pq} \cup E^*_{qp} = E_{pq}$. 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. Similarly, 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; upon receiving the sub-problem, the worker iterate all the ordered color-assignments. For each colored-assignment the worker reads the directed edge sets from distributed storage, and enumerates the 4-node graphlets by

applying the a modified S4GE algorithm. Finally, the enumeration outputs across all distributed workers are aggregated to yield to correct total enumeration count.

3.3.1 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 15, with the details of the explore-functions are given in Algorithms 16, 17, and 18 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 applied to different edge-based enumeration algorithm to suit different needs. All it requires is to modify the input to accommodate a directed sub-set of the input graph.

Algorithm 15 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**
          ExploreTriangle $(u, v, u', E_{\text{map}}, ijkl)$
        if $(u' < v') \wedge (u' > v)$ **then**
          ExploreWedge-1 $(v, u, u', E_{\text{map}}, ijkl)$
        if $u' > v'$ **then**
          ExploreWedge-2 $(u, v, v', E_{\text{map}}, ijkl)$

Algorithm 16 Explore Triangle

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

$E^*_{il} \equiv E_{\text{map}}[il]$, $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
$N^{>u}(u) \equiv \{z | z \in N(u)|_{E^*_{il}}, \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(u \cap N^{>u}(v) \cap N^{>u}(w)$ with $z > w$ **do**
  Enumerate4Clique (u, v, w, z)
for all z in two sets and $z >$ opposite node **do**
  EnumerateDiamond (u, v, w, z)
for all z in one set only **do**
  EnumerateTailedTriangle (u, v, w, z)

Algorithm 17 EXPLORE WEDGE TYPE-1

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

$E^*_{il} \equiv E_{\text{map}}[il]$, $E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
$N^{>u}(u) \equiv \{z | z \in N(u)|_{E^*_{il}}, \eta(z) > \eta(u)\}$
$N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(v) \cap N^{>u}(w)$ with $z \notin N^{>u}(u)$ **do**
    ENUMERATERECTANGLE (u, v, z, w)
for all $z \in N^{>u}(u)$ only **do**
    if $z > w$ **then**
        ENUMERATE3STAR (u, v, w, z)
for all $z \in N^{>u}(v)$ only **do**
    ENUMERATE3PATH (w, u, v, z)
for all $z \in N^{>u}(w)$ only **do**
    ENUMERATE3PATH (v, u, w, z)

Algorithm 18 EXPLORE WEDGE TYPE-2

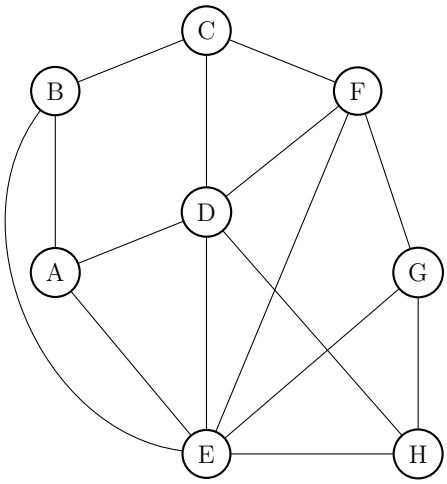
Require: Given wedge (v, u, w); $E_{\text{map}} \equiv \{(i, j) \mapsto E^*_{ij}\}$; ordered color-assignment $ijkl$.

$E^*_{jl} \equiv E_{\text{map}}[jl]$, $E^*_{kl} \equiv E_{\text{map}}[kl]$
$N^{>u}(v) \equiv \{z | z \in N(v)|_{E^*_{jl}}, \eta(z) > \eta(u)\}$
$N^{>u}(w) \equiv \{z | z \in N(w)|_{E^*_{kl}}, \eta(z) > \eta(u)\}$
for all $z \in N^{>u}(v)$ only **do**
    if $z > w$ **then**
        ENUMERATE3STAR (v, u, w, z)
for all $z \in N^{>u}(w)$ only **do**
    if $z \neq v$ **then**
        ENUMERATE3PATH (u, v, w, z)

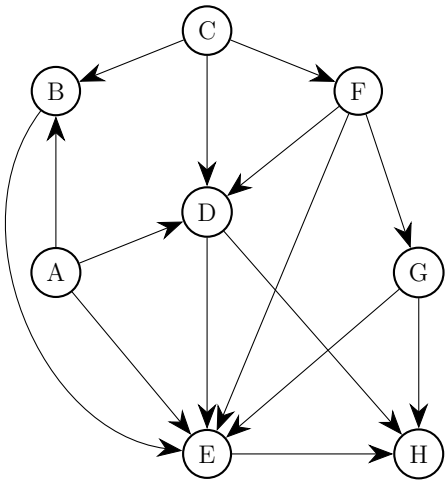
3.3.2 D4GE walk through

Here we use a concrete example to demonstrate how D4GE works. Because the simplicity of this example, the edge-orientation and symmetrization seem unnecessary. We want to emphasize that the edge-orientation in our implementation is far more involved. The details and analysis of the real-world edge-orientation can be found in [26].

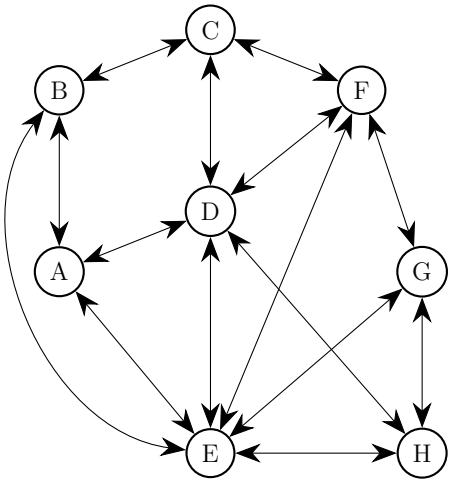
Consider the undirected graph Figure 3.2a:



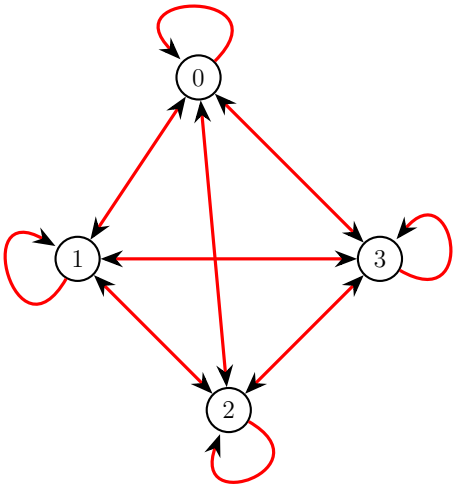
(a) Original graph



(b) After edge-orientation



(c) After symmetrization



(d) After partitioning

Figure 3.2: D4GE illustration.

D4GE first applies edge-orientation to Figure 3.2a (line 1 of Algorithm 11). Assuming $\rho = 4$ and the orientation rule is the following: $\{C, F \mapsto 0; A, B \mapsto 1; D, G \mapsto 2; E, H \mapsto 3\}$ and ties are broken by alphabetical order, we obtain Figure 3.2b where each edge has its own assigned orientation. D4GE then symmetrizes Figure 3.2b to obtain Figure 3.2c (line 2 of Algorithm 11), and proceeds to partition the symmetrized graph into directed edge-sets (Algorithm 12).

We use a condensed representation as Figure 3.2d to demonstrate the partitioning where:

- Each vertex is a union of the original vertices of the same color;
- Each edge is a union of the inter-color edges from the original graph;
- Each self-loop is a union of the intra-color edges from the original graph.

For example, the self-loop on vertex 1 is E^*_{11}, which includes $\overrightarrow{(A, B)}$ and $\overrightarrow{(B, A)}$ in Figure 3.2c; the edge pointing from vertex 2 to vertex 3 is E^*_{23}, which includes $\overrightarrow{(D, E)}$, $\overrightarrow{(D, H)}$, $\overrightarrow{(G, E)}$ and $\overrightarrow{(G, H)}$ in Figure 3.2c. The entire Figure 3.2d is saved as 16 (ρ^2) different directed edge-sets E^*_{ij} on the distributed file system by D4GE.

D4GE now enters its enumeration stage. It generates $\binom{\rho}{2} + \binom{\rho}{3} + \binom{\rho}{4}$ sub-problems and assigns each sub-problem to a distributed worker (line 4 of Algorithm 11). Upon receiving the sub-problem, the distributed worker starts to select grouped color-assignments. For example, with $\rho = 4$, the worker that received sub-problem S_{02} will go through all the $4^4 = 64$ permutations of 4-color tuples, and realizes that only $(0, 0, 0, 2)$, $(0, 0, 2, 0)$, $(0, 0, 2, 2)$, $(0, 2, 0, 0)$, $(0, 2, 0, 2)$, $(0, 2, 2, 0)$, $(0, 2, 2, 2)$, $(2, 0, 0, 0)$, $(2, 0, 0, 2)$, $(2, 0, 2, 0)$, $(2, 0, 2, 2)$, $(2, 2, 0, 0)$, $(2, 2, 0, 2)$ and $(2, 2, 2, 0)$ belong to the sub-problem S_{02} (Algorithm 13), where all of them require knowledge of, or subset of $E^*_{00} \cup E^*_{02} \cup E^*_{20} \cup E^*_{22}$. The distributed worker then reads the all the directed edge-sets (Algorithm 14) and stores them in the hash-map (line 7 of Algorithm 11).

Finally D4GE invokes S4GE_{CD} to enumerate over each color-assignment sequentially with the loaded edge-sets.

3.3.3 Compact-Forward for 4-clique listing

PSE with VF2 only supports one query graph per execution. For the purpose of comparing our D4GE partitioning scheme against PSE with VF2, 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. 19:

Algorithm 19 CF4_{CD}

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

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

Note that CF4 extends the idea of Compact-Forward algorithm (Algorithm 1) 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.

3.3.4 Analysis

In this analysis , first we show that D4GE with S4GE_{CD} correctly enumerates all the 4-node graphlets.

Theorem 1. *Enumerating over all ordered color-assignments enumerates all 4-node graphlets once and once only.*

Proof. By definition, D4GE works by generating all possible color-assignments of all 4-node graphlets. In other words, any 4-node graphlet must be found from one and

only one of the color-assignments. D4GE then applies S4GE_{CD} algorithm on each individual color-assignment. Since S4GE correctly enumerates all 4-node graphlets for any given graph, D4GE/S4GE_{CD} correctly enumerates all 4-node graphlets for all color-assignments of $G^{sym}(V, E^{sym})$. □

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 with VF2.

Theorem 2. *D4GE with S4GE$_{\text{CD}}$ requires at most $2\,m^{\text{sym}}$ amount of additional network reads compared to PSE with VF2.*

Proof. Recall that $E_{ii} = E^*_{ii}$, $E_{ij} = E^*_{ij} \cup E^*_{ji}$. The network read for D4GE/S4GE_{CD} can be summarized by:

1. The edge set E_{ii} is requested $(\rho-1)$ times by sub-problems S_{ij} when $i+1 = j\%\rho$, and $\binom{\rho-1}{2}$ times by sub-problems S_{ikl}. Hence the amount of network read is $\sum_{i=0}^{\rho-1} |E_{ii}| \binom{\rho}{2}$.

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:

$$
\begin{aligned}
&\sum_{i=0}^{\rho-1} |E_{ii}| \binom{\rho}{2} + \sum_{i=0}^{\rho-1} \sum_{j=i+1}^{\rho-1} |E_{ij}| \left[1 + \binom{\rho-1}{2}\right] \\
&= \binom{\rho}{2} \left[\sum_{i=0}^{\rho-1} |E_{ii}| + \sum_{i=0}^{\rho-1} \sum_{j=i+1}^{\rho-1} |E_{ij}|\right] - (\rho-2) \sum_{i=0}^{\rho-1} \sum_{j=i+1}^{\rho-1} |E_{ij}| \qquad (3.1) \\
&\equiv \binom{\rho}{2} m^{\text{sym}} - (\rho-2) m^{\text{sym}}_{\neq}
\end{aligned}
$$

where $m^{sym}_{\neq}$ stands for the number of edges of the edge induced sub-graph of the symmetrised graph, where all edges' endpoints are of different colors.

Recall that for an input graph, D4GE partitions the graph into ρ^2 directed edge sets, and ρ of such edge sets have the endpoints of the same color (E^*_{ii}). If we assume the edges are distributed evenly, the expected size of $m^{\text{sym}}_{\neq}$ is $\frac{\rho^2-\rho}{\rho^2}$=$1-\frac{1}{\rho}$ of m^{sym}. Thus expression 3.1 can be further simplified to:

$$\begin{aligned}&\binom{\rho}{2} m^{sym} - (\rho-2) m^{sym}_{\neq} = [\binom{\rho}{2} - (\rho-2)(1-\frac{1}{\rho})] m^{sym} \\ &= [\binom{\rho}{2} - \rho + 3 - \frac{2}{\rho}] m^{sym}\end{aligned} \tag{3.2}$$

Recall that for $k=4$, PSE requires $\binom{\rho-1}{2} m^{\text{sym}}$ amount of network read. Comparing Expression 3.2 to PSE network read, the difference is:

$$\left[\binom{\rho}{2} - \rho + 3 - \frac{2}{\rho}\right] m^{\text{sym}} - \binom{\rho-1}{2} m^{\text{sym}} = \left(2 - \frac{2}{\rho}\right) m^{\text{sym}} \tag{3.3}$$

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 use S4GE_{CD} as the local algorithm for both partitioning schemes. Since S4GE_{CD} enumerates all 4-node graphlets by discovering the base triangles and wedges first, we separate the work calculation into two parts: one being the amount of work to discover all the base triangles and wedges, the other to discover the fourth vertex.

Let us consider the first part. We can see that $\sum_{(u,v)\in E^{\text{sym}}} (d^{\text{sym}}(u) + d^{\text{sym}}(v))$ is the amount of work to intersect all pairs of edges for a symmetrised graph. This sum is bounded by and can be estimated by $2\, m^{\text{sym}} d^{\text{sym}}_{\text{max}}$, where $d^{\text{sym}}_{\text{max}}$ is the maximum degree of the symmetrised graph. Following the analysis to derive expression 3.1,

D4GE does

$$2\binom{\rho}{2} m_{=}^{\text{sym}} d_{\max}^{\text{sym}} + 2[1 + \binom{\rho - 1}{2}] m_{\neq}^{\text{sym}} d_{\max}^{\text{sym}} \tag{3.4}$$

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

$$2\binom{\rho - 1}{2} m_{=}^{\text{sym}} d_{\max}^{\text{sym}} + 2\binom{\rho - 1}{2} m_{\neq}^{\text{sym}} d_{\max}^{\text{sym}}. \tag{3.5}$$

Subtracting Expression 3.4 by Expression 3.5 yields

$$\begin{aligned} & 2\binom{\rho - 1}{1} m_{=}^{\text{sym}} d_{\max}^{\text{sym}} + 2 m_{\neq}^{\text{sym}} d_{\max}^{\text{sym}} \\ & = 2(\rho - 2)\, m_{=}^{\text{sym}} d_{\max}^{\text{sym}} + 2 m^{\text{sym}} d_{\max}^{\text{sym}} \end{aligned} \tag{3.6}$$

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 3.6 can be simplified to

$$\begin{aligned} & 2(\rho - 2)\, m_{=}^{\text{sym}} d_{\max}^{\text{sym}} + 2 m^{\text{sym}} d_{\max}^{\text{sym}} \\ & = 2\frac{\rho - 2}{\rho} m^{\text{sym}} d_{\max}^{\text{sym}} + 2 m^{\text{sym}} d_{\max}^{\text{sym}} \\ & = \left(4 - \frac{4}{\rho}\right) m^{\text{sym}} d_{\max}^{\text{sym}} \end{aligned} \tag{3.7}$$

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 $3\, d_{\max}^{\text{sym}}$. Also for each graph dataset, the numbers of triangles and wedges are fixed. Denote the uni-color triangles and wedges as Δ_I and $\angle_I$, the bi-color ones as Δ_{II} and $\angle_{II}$,

and the tri-color ones as Δ_{III} and $\angle_{III}$. For PSE with S4GE, each Δ_I and $\angle_I$ is emitted $\binom{\rho-1}{2}$ times from E_{ii}; each Δ_{II} and $\angle_{II}$ is emitted $\binom{\rho-2}{1}$ times from $E_{ii} \cup E_{ij}$; each Δ_{III} and $\angle_{III}$ is emitted $1+\binom{\rho-3}{1}$ from $E_{ij} \cup E_{jk} \cup E_{ik}$. Hence the total work for locating the 4th vertex z, using PSE partitioning scheme and S4GE serial algorithm is:

$$\begin{aligned} &3\binom{\rho-1}{2} d^{\text{sym}}_{\max}(|\Delta_I| + |\angle_I|) + 3\binom{\rho-2}{1} d^{\text{sym}}_{\max}(|\Delta_{II}| + |\angle_{II}|) \\ &+ 3\left(1 + \binom{\rho-3}{1}\right) d^{\text{sym}}_{\max}(|\Delta_{III}| + |\angle_{III}|) \end{aligned} \tag{3.8}$$

D4GE/S4GE$_{\text{CD}}$ enumerates each triangle and wedge 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 to ensure the correctness. Thus overall, D4GE with S4GE$_{\text{CD}}$ does

$$3\,\rho\, d^{\text{sym}}_{\max}(|\Delta| + |\angle|) \tag{3.9}$$

amount of work. Expression 3.8 minus expression 3.9 yields

$$\begin{aligned} &3\left(\binom{\rho-1}{2} - \rho\right) d^{\text{sym}}_{\max}(|\Delta_I| + |\angle_I|) \\ &- 6\, d^{\text{sym}}_{\max}(|\Delta_{II}| + |\angle_{II}| + |\Delta_{III}| + |\angle_{III}|) \end{aligned} \tag{3.10}$$

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

Now consider expressions 3.7 and 3.10. Expression 3.7 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 3.10 shows that the extra work performed by PSE with S4GE to list the 4th vertex,

has a quadratic growth with respect to ρ, and is sensitive to the number of uni-color triangles and wedges.

We emphasize 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_{CD} can often achieve greater performance improvement. This will be confirmed in the Experiments chapter.

3.4 Implementation

We target out algorithm towards real world graphs that can grow very large in size with hundreds of millions of vertices and edges. The sheer size of such graphs can easily exceed the capacity of any commodity computer. This issue, fortunately, can be addressed using compression. The WebGraph framework ([3] and [4]) provides a highly compressed file format for graph data. It also features an easy-to-use interface that allows lazy reading, where only part of the graph requested is loaded from the disk, which further reduces the operational memory footprint and simultaneous file I/O's.

We implemented our D4GE with S4GE_{CD} algorithm using Apache Spark [33] version 3.0.0[2] and OpenJDK 11[3]. For the comparison experiments, we also implemented PSE according to [22], and CF4_{CD} described in Algorithm 19 with the same technologies.

By default, Spark does not support WebGraph as an input format. We employed an open-source package on Github[4] that makes the WebGraph file format recognizable by Spark.

D4GE algorithm requires a shared persistent storage accessible by all Spark work-

[2]https://archive.apache.org/dist/spark/spark-3.0.0/
[3]https://openjdk.java.net/projects/jdk/11/
[4]https://github.com/helgeho/HadoopWebGraph/

ers for storing the partitioned edge-sets. This storage can be in the format of either local disk or network-mounted drive, or a Hadoop Distributed File System[5].

Our implementations are managed by Apache Maven[6]. To build all the relevant code, execute: `mvn clean compile assembly:single`. For example, to run our algorithms against ljournal-2008 dataset, execute:

```
spark-submit --conf spark.hadoop.fs.default.name=file:/// \
         --driver-memory 8g \
         --class quad.CD \
         target/triangles-1.0.jar \
         --input-dir $(pwd)/ljournal-2008/ \
         --original ljournal-2008 \
         --num-colors 5
```

where `--conf spark.hadoop.fs.default.name=file:///` specifies the type of persistent storage; `--driver-memory 8g` specifies the the amount of memory allocated for each Spark worker machine; `--class quad.CD` specifies which algorithm to execute; `target/triangles-1.0.jar` is the path to the jar file built by Maven; `--input-dir $(pwd)/ljournal-2008/` specifies the absolute input path directory; `--original ljournal-2008` specifies the WebGraph file name (without extension) in the input directory; `--num-colors 5` specifies the number of colors, i.e. ρ.

Finally, our implementations are compatible with all *nix operation systems, and on clusters such as UVic's BigDataGrid and Compute Canada. Details on the cluster configurations are documented in Section 4.1.

[5]http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[6]https://maven.apache.org/

Chapter 4

Experiments

In this chapter we describe the experiments that we performed with the proposed D4GE/S4GE algorithm. First we describe the computing cluster configurations for the experiments as well as the graph datasets; then we show the experimental results for D4GE compared against the known SotA MapReduce-based graph enumeration algorithms and partitioning schemes.

4.1 Cluster Configurations

We employed **two** sets of clusters to conduct our experiments. For *small-medium* sized graph datasets, we employed UVic's BigDataGrid[1] cluster; for *large* graphs, we deployed our algorithms onto Compute Canada[2]. The detailed configurations are listed below:

1. BigDataGrid:
    - Intel E5430 quad-core CPU per machine;
    - 6GB RAM per machine;

[1]Deprecated.
[2]`https://docs.computecanada.ca/wiki/Cedar`

- 30 machines.

2. Compute Canada:

   - Two Intel Platinum 8260 CPUs per machine, with 24 cores each CPU.
   - 192GB RAM per machine;
   - 14 machines.

Effectively, BigDataGrid gives 120 workers with 1.5GB RAM per worker; Compute Canada gives 672 workers with 4GB RAM per worker. For both clusters, the algorithms are compiled with OpenJDK 11, and are executed with Spark 3.0.0.

4.2 Graph Datasets

We applied our algorithms on eleven networks in compressed WebGraph format. The datasets were downloaded from the WebGraph website[3]. We downloaded both the graph and its transpose for each dataset, and we symmetrized them to get the corresponding undirected graphs. The datasets are:

1. **enron**: email communications among Enron employees, made public by the Federal Energy Regulatory Commission.
2. **cnr-2000**: a small crawl from the Italian CNR (Consiglio Nazionale delle Ricerche).
3. **amazon-2008**: A symmetric graph describing similarity among books as reported by the Amazon store.

[3]http://law.di.unimi.it/datasets.php

4. **hollywood-2009**: One of the most popular undirected social graphs: the graph of movie actors. Vertices are actors, and two actors are joined by an edge whenever they appeared in a movie together.

5. **hollywood-2011**: same as **hollywood-2009**, except the data were from year 2011.

6. **dewiki-2013**: a snapshot of the German part of Wikipedia[4] as of late February 2013

7. **orkut-2007**: a social networking and discussion site operated by Google. This snapshot was part of the IMC 2007 Data Sets [16].

8. **ljournal-2008**: a snapshot from LiveJournal[5] in 2008.

9. **uk-2002**: a crawl of .uk domain, performed by UbiCrawler [2] in 2002.

10. **enwiki-2018**: a snapshot of the English part of Wikipedia[6] as of mid 2018.

11. **indichina-2004**: a fairly large crawl of the country domains of Indochina performed for the Nagaoka University of Technology.

For each network `basename` we downloaded `basename.graph`, `basename.properties`, `basename-t.graph` and `basename-t.properties`. We used the WebGraph toolkit to create the `offsets` by the following commands:

```
java it.unimi.dsi.webgraph.BVGraph -o -O -L basename
```

and

```
java it.unimi.dsi.webgraph.BVGraph -o -O -L basename-t
```

[4]http://de.wikipedia.org/
[5]https://www.livejournal.com/
[6]http://en.wikipedia.org/

assuming the WebGraph library is already under Java's CLASSPATH. We then used the following command to symmetrize the original and transpose:

```
java it.unimi.dsi.webgraph.Transform union \
    basename \
    basename-t \
    basename-sym
```

The statistics of the symmetrized graphs are listed in Table 4.1. We order them by the ascending in number of edges m. The smallest dataset, **enron**, has only 510K edges, while the largest dataset, **uk02**, has more than 529M edges. As we shall see, that the enumeration times do not correlate to the number of vertices n nor edges m; rather, the enumeration times are sensitive to $d_{\max}^{\text{sym}} * (|\Delta| + |\angle|)$ - the product of the maximum degree of the symmetrized graph, and the sum of the number of triangles and wedges, pointed out by Santoso et al. [26]. We list $d_{\max}^{\text{sym}}$, $|\Delta|$ and $|\angle|$ in Table 4.1 as well. We can see that these three key characteristics of the graphs vary differently, and are independent of n and m of the graphs. Specifically, **dewiki** has only 1.5M vertices and 33M edges but a $d_{\max}^{\text{sym}}$ of 118K, which is almost 10 times greater than the graphs that are much larger itself, such as **hollywood09** and **hollywood11**; **enwiki18**, **indochina** and **uk02** also have high $d_{\max}^{\text{sym}}$ values that are much greater than the rest. Thus we predict that **dewiki**, **enwiki18**, **indochina** and **uk02** require much more intense computing power than the rest, hence much longer running time. We categorize **enwiki18**, **indochina** and **uk02** as large graph datasets and deployed onto Compute Canada, and the rest as small-medium graphs deployed onto BigDataGrid.

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

Table 4.1: The numbers of vertices n, edges m, wedges $|\angle|$, and triangles $|\Delta|$, and the max degree of the symmetrized graphs.

4.3 The impact of ρ on performance

In this section we address the workload distribution problem mentioned in Section 3.3 (Table 3.1), and how an properly chosen large ρ value can help us achieve this goal. Recall that the workload distribution problem arises from the fact that sub-problems vary in size, hence the worker with smaller sub-problems will finish earlier than the one with larger sub-problem. The slow running worker dwarfs the performance of entire application. This was called *the curse of the last reducer* by Suri and Vassilvitskii [29].

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. [19] 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 Fig. 4.1. The three graph datasets are all small-medium, and deployed on BigDataGrid with 120 distributed workers.

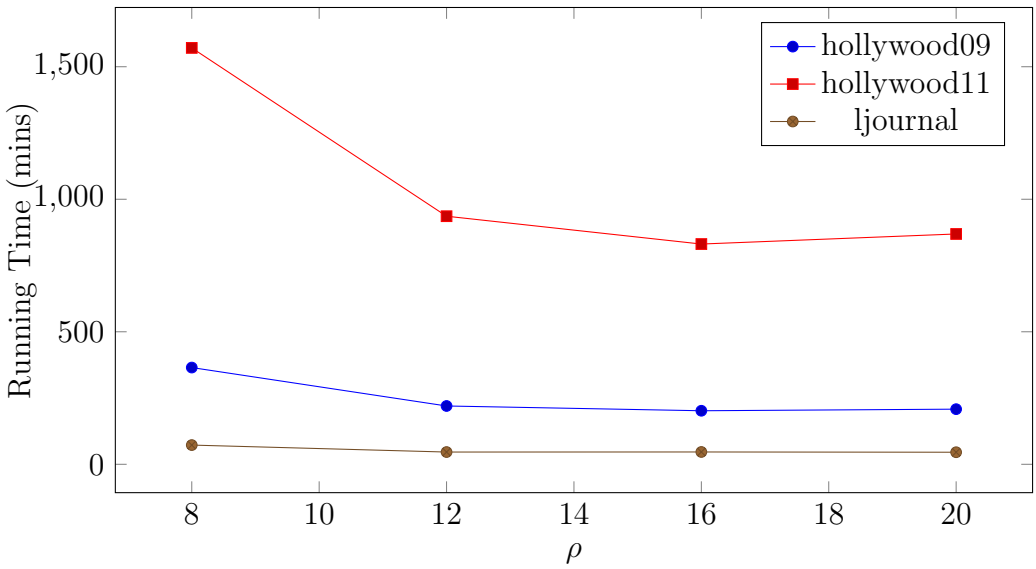


Figure 4.1: The enumeration time (minutes) of D4GE/S4GE$_{\text{CD}}$ on several graphs, with varying value of ρ. Higher ρ does not add much overhead; the lines flatten out rather than slopping up perceptibly.

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. From $\rho = 8$ to 12, we observe improvement in performance,

consistently on various datasets. This is because of better load balancing - with $\rho = 12$, there are almost 5 times more sub-problems than $\rho = 8$, which gives us a better workload distribution. With increasing value of ρ, the number of sub-problems over-saturates the number of the distributed workers, and each sub-problem becomes smaller. The over-saturated sub-problems are waiting to be picked up by the next available worker, and the smaller sub-problems definitions result in a better sub-problem distribution over all the workers.

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 acknowledge that the increasing ρ yields heavier network read of the edge-sets. However, we would like to point out 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.

The choice of large ρ value also has a positive side effect, discussed in the next section.

4.4 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 with suitable ρ values for the number of workers. The experiments are conducted over Computer Canada. The results are presented in Figure 4.2. D4GE/S4GE shows strong scalability: with slope -0.968 and -0.899 re-

spectively, 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 our scalability is on-par with the SotA Map-Reduce based algorithms [19] and [22]. Our scalability results from the choice of large ρ value. We restate that a large ρ value over-saturates the number of distributed workers, thus every time the number of distributed workers doubles, the enumeration time reduces in half.

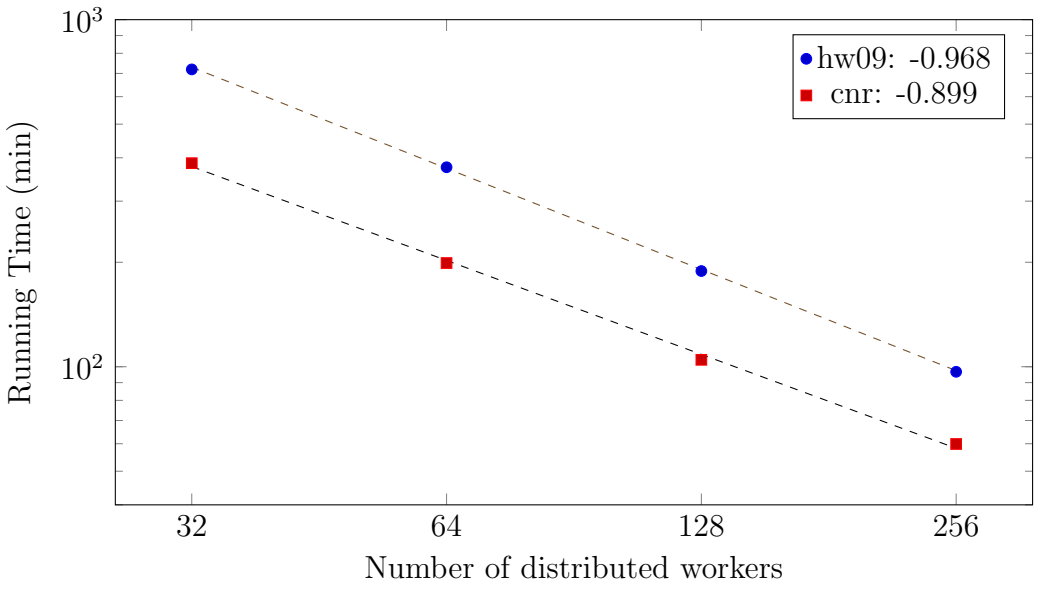


Figure 4.2: Machine scalability of D4GE/S4GE on *cnr* and *hollywood09*. D4GE/S4GE show very strong scalability with slope -0.899 and -0.968 which is very close to -1, the perfect value.

4.5 PSE/S4GE vs D4GE/S4GE$_{\text{CD}}$

In this section we compare the proposed D4GE partitioning scheme against the StoA MapReduced partitioning scheme PSE. In this comparison we choose S4GE as the serial algorithm for both partitioning schemes, and implemented PSE according to [20]. We set $\rho = 16$. The enumeration times are listed in Table 4.2. We found that D4GE/S4GE$_{\text{CD}}$ is more efficient for all of the tested graphs, and D4GE/S4GE$_{\text{CD}}$ is

able to achieve up to 11x speedup, which is on the **cnr** dataset.

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	>7days	2328	/
hollywood11	9186	864	10.6
orkut	3799	390	9.7
ljournal	432	47	9.2

Table 4.2: The enumeration time (minutes) of D4GE/S4GE$_{\text{CD}}$ against PSE/S4GE, with $\rho = 16$, 120 workers.

A significant speedup is achieved on **cnr**, **hollywood09**, **hollywood11**, **orkut** and **ljournal**. For these datasets, the number of wedges plus triangles are much greater than the number of edges, as can be seen in Table 4.1. According to expression 3.7 and 3.10, PSE's performance is penalized by the number of triangles from type-1 triangles and wedges, whereas D4GE's performance is penalized no more than the number of edges from the symmetrised graph. This gives an advantage to D4GE/S4GE$_{\text{CD}}$ compared to PSE/S4GE.

Graphlets	enron	cnr	amazon	hw09	dewiki	hw11	orkut	ljournal
3-path	2.51B	6.12B	372M	21.4T	10.4T	104T	18.6T	1.81T
3-star	8.04B	41.4T	610M	16.7T	661T	92.8T	97.8T	8.85T
square	21.6M	37.9B	2.69M	168B	13.1B	643B	70.1B	8.55B
tailed-triangle	583M	79.4B	92.3M	8.87T	993B	26.8T	1.51T	190B
diamond	46.1M	43.0B	13.1M	635B	11.9B	1.88T	47.8B	27B
4-clique	5M	160M	4.19M	1.39T	158M	728B	3.22B	16.1B
Running Time (min)	0.18	132	0.37	204	2328	864	390	47

Table 4.3: The outputs of D4GE/S4GE$_{\text{CD}}$, on UVic BigDataGrid of 120 workers, $\rho = 16$.

For the largest datasets, **uk02**, **enwiki18** and **indochina**, we experimented D4GE/S4GE$_{\text{CD}}$ using Computer Canada cluster configuration with $rho = 25$, which gives 15,250 sub-problems.

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

Table 4.4: The outputs of D4GE/S4GE$_{\text{CD}}$, on Compute Canada cluster of 672 workers, $\rho = 25$.

D4GE/S4GE$_{\text{CD}}$ 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 14 days to run using the same cluster, which is impractical.

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

In previous section we have seen that D4GE is more than PSE with S4GE serial algorithm. To be more convincing, in the following comparison we swap out S4GE and attempt to enumerate 4-cliques instead of all 4-node graphlets. Specifically, we compare Park et al.'s PSE/VF2 implementation with 4-clique (K4) query, against our D4GE/CF4$_{CD}$. The reference implementation of PSE/VF2 is obtained from `https://datalab.snu.ac.kr/pegasusn/download.php`. This comparison experiment is conducted on UVic BigDataGrid cluster, with $rho = 16$. The results are shown in Table 4.5.

Dataset	PSE/VF2	D4GE/CF4$_{CD}$	Speedup
enron	0.7	0.15	4.7
cnr	1.1	0.33	3.3
amazon	1.3	0.25	5.2
hollywood09	324	16	20.3
dewiki	4.5	1.5	3.0
hollywood11	288	31	9.3
orkut	16	5.0	3.2
ljournal	11	2.5	4.4

Table 4.5: Enumeration time (minutes) of D4GE/CF4$_{CD}$ against PSE/VF2, with $\rho = 16$

Comparing our D4GE/CF4$_{CD}$ suite against one of the SotA sub-graph enumeration algorithm, up to 5.2 fold speedup is observed on small graphs such as **amazon**,

and >20 fold speedup on larger graphs such as **hollywood09**.

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 partitioning scheme can be applied to query 4-cliques, it is designed for a bigger goal - enumerating all 4-node graphlets with S4GE.

4.7 PSE Duplications

In the previous two sections we have seen D4GE partitioning scheme's advantage over PSE, for two different serial algorithms. In the final section of the experiments, we show that overall speedup of D4GE against PSE is because D4GE eliminates the duplicated emissions from PSE.

To show this, we implement a duplication counter in PSE which keeps a record of the over-emitted sub-graphs (line 8 of Algorithm 10). We obtained Park et al.'s PSE/VF2 implementation from `https://datalab.snu.ac.kr/pegasusn/download.php`, version 3.0.1., and modified accordingly. We list the percentages of duplicate emissions from PSE/S4GE, PSE/CF4, 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 4.6.

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%

Table 4.6: Duplicated emissions from PSE partitioning scheme with different local algorithms.

Overall, despite the choices of serial algorithm, we can see the PSE emits duplicates nonetheless. 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.

Comparing the second and the third column of Table 4.6 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 of **hollywood**s and **orkut** datasets.

These duplicated emissions from PSE are the keys to the performance of D4GE. D4GE leverages the relationship between the 4-node graphlets and the color assignments; by listing the color assignments in an ordered fashion, each 4-node graphlet is discovered and emitted once and once only.

Chapter 5

Discussion

It is common in the literature that performance or scalability is measured against the size of the input graph, either by the number of vertices or more commonly the number of edges. We would like to point out that in the context of 4-node graphlet enumeration, using the S4GE algorithm, the number of vertices or edges should not be the primary consideration when it comes to the amount of computation. In [26] 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 4.2, **ljournal** which is three times larger than the **dewiki** in size has a runtime that is only 2% of the **dewiki**'s. Notice that **dewiki** has a much larger number of graphlets, in particular the 3-stars. We plot the enumeration time of eight small-medium datasets and three large datasets against $d_{\max}^{\text{sym}}(|\Delta| + |\angle|)$ in Figure 5.1. 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. the enumer-

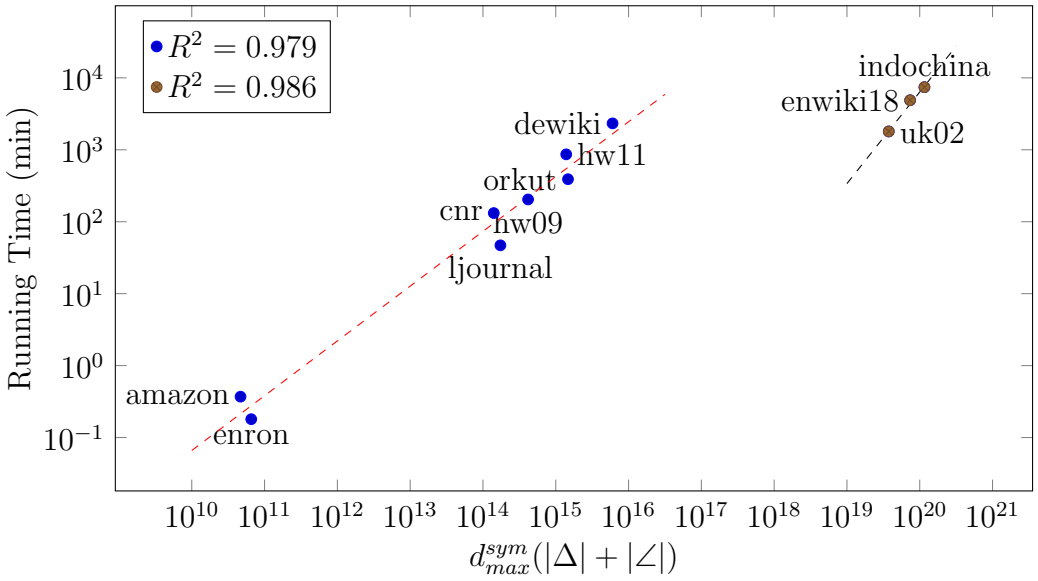


Figure 5.1: Strong correlation between the enumeration time and $d^{sym}_{max}(|\Delta| + |\angle|)$ for both the small-medium and large datasets.

ation time of each distributed worker follows the single machine expectation. If we assume each distributed worker receives an even share of the original graph, then we can expect the distributed work load to be a faction of the single machine work load, hence following the expectation.

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

$$N = M\, I \tag{5.1}$$

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

$$M = \begin{bmatrix} 1 & 0 & 4 & 2 & 6 & 12 \\ 0 & 1 & 0 & 1 & 2 & 4 \\ 0 & 0 & 1 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 & 4 & 12 \\ 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Another question that the audience 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.

Chapter 6

Conclusions and Future Work

In this work, we have presented D4GE scheme that brings 4-node graphlet enumeration into the distributed platform. This scheme is able to suppress duplicate emissions which are unavoidable with other schemes. We show that, D4GE when combined with S4GE, requires at most $2\,m^{sym}$ more network read compared to PSE/S4GE, and is able to reduce the amount of work relative to PSE/S4GE for real world graphs, where the numbers of wedges and triangles are order of magnitudes greater than the number of edges. We experimented D4GE with S4GE_{CD} and CF4_{CD} localized algorithms, and both combinations out-perform the *state-of-the-art* competitors by up to 11x. Last but not least, $\text{D4GE/S4GE}_{\text{CD}}$ 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, and emitting more than 10 quadrillion graphlets.

There are several possibilities to improve the performance of our algorithm even further. When running $D4GE/S4GE_{\text{CD}}$, we noticed that while setting a large ρ partially addressed the imbalanced nature of the sub-problems distribution, there were still dangling workers that ran significantly longer than the rest. This is because of the heavy performance-hitting vertices from the experimental graph. For certain vertices the d^{sym} can be very large thus the projected enumeration time of $O(d^{\text{sym}}(|\Delta| + |\angle|))$

is dwarfed. In hindsight we can acquire the maximum d^{sym} for each sub-problem, and when distributing the sub-problems, we can prioritize the allocations of those with high maximum d^{sym} onto different workers. Effectively, we can use the summation of maximum d^{sym} of each assigned sub-problem to estimate the total enumeration time of each distributed worker, and try to maintain such a balance.

In addition we can alter the order of the edge orientation application. Specifically we can apply the edge orientation onto the sub-problems assigned to individual workers after the partitioning. Recall that edge orientation relabels the vertices and redistributes the degree distribution. While the cost of edge-orientation was not revealed in Santoso et al. [26], it is clear that larger graphs naturally incur more processing time. Thus by dividing the entire graph into sub-problems (sub-graphs) we limit the size of the graph to be processed, effectively distributing the edge-orientation process in the parallel setting.

Last but not least, we can explore different localized algorithms and migrate them under $D4GE$. We want to restate $D4GE$'s extensibility to employ various edge-based enumeration algorithms (for example, $CF4$ was easily fitted to $D4GE$), as the partitioning process is complete opaque to the enumeration algorithm, and the enumeration algorithm solely operates on the partitioned sub-problem (sub-graph).

Bibliography

[1] Aly Ahmed, Keanelek Enns, and Alex Thomo. Triangle enumeration for billion-scale graphs in rdbms. In Leonard Barolli, Isaac Woungang, and Tomoya Enokido, editors, *Advanced Information Networking and Applications*, pages 160–173, Cham, 2021. Springer International Publishing.

[2] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.

[3] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[4] 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.

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

[6] 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.

[7] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

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

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

[16] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.

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

[18] 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.

[19] 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.

[20] Ha-Myung Park, F. Silvestri, R. Pagh, C. Chung, Sung-Hyon Myaeng, and U. Kang. Enumerating trillion subgraphs on distributed systems. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12:1 – 30, 2018.

[21] 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.

[22] 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.

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

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

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

[26] Yudi Santoso, Venkatesh Srinivasan, and Alex Thomo. Efficient enumeration of four node graphlets at trillion-scale. In *23rd EDBT*, pages 439–442, 2020.

[27] Yudi Santoso, Venkatesh Srinivasan, Alex Thomo, and Sean Chester. Triad enumeration at trillion-scale using a single commodity machine. In *22nd EDBT*, 2019.

[28] Paramvir Singh, Venkatesh Srinivasan, and Alex Thomo. Fast and scalable triangle counting in graph streams: The hybrid approach. In Leonard Barolli, Isaac Woungang, and Tomoya Enokido, editors, *Advanced Information Networking and Applications*, pages 107–119, Cham, 2021. Springer International Publishing.

[29] 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.

[30] Jia Wang and James Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9), 2012.

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

[32] Tengkai Yu, Venkatesh Srinivasan, and Alex Thomo. *Triangle Enumeration on Massive Graphs Using AWS Lambda Functions*, pages 226–237. Springer International Publishing, Cham, 2020.

[33] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.