Triangle Enumeration in Massive Graphs using Map Reduce

by

Pooja Bhojwani
B.Tech, Government Women Engineering College, 2012

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Triangle Enumeration in Massive Graphs using Map Reduce

by

Pooja Bhojwani
B.Tech, Government Women Engineering College, 2012

Supervisory Committee

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-supervisor
(Department of Computer Science)

ABSTRACT

In this era of big data, *graph*, which adds the advantage of structural representation of data has gained extreme importance. Analyzing the graphical structure of the data provides deep, meaningful insights about it and is widely used for a vast number of applications. Enumerating triangles is one of the crucial pillars of complex graph analysis and lays the basis for two most fundamental measures of the stability of a network, clustering coefficient, and transitivity ratio. Besides, triangle listing also has applications in wide range of domains, such as spam detection, finding communities, fake account detection in social networks, and many more. Several internal memory algorithms have been proposed to tackle this problem. However, these algorithms are not scalable for the massive graphs generated from big data. One way to solve this is by utilizing the power of parallel computation and thereby distributing the work to various machines. Google's map-reduce model implements parallel computation and also manages data partition.

In this project, our goal is to list triangles in massive directed and undirected graphs using map-reduce. For triangle enumeration in undirected graphs, we implement existing map-reduce algorithmic solution. We also propose an extension to the algorithm for directed cycle and trust triangles detection. Finally, we perform an extensive evaluation of the proposed map-reduce solution for both directed and undirected graphs on real-world datasets. Experimental results show that these algorithms are able to enumerate the triangles in very large within a very short span of time.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My supervisors, Dr. Alex Thomo and Dr. Sudhakar Ganti, for being supportive and providing constant encouragement. I am deeply grateful for their mentoring and patient guidance throughout my master's program.

Dr. Venkatesh Srinivasan, for providing his insights and expertise on graph theory and algorithms.

My teammate, Yudi Santoso, for his ideas, inspiration, support and friendship. He has helped me throughout to overcome any hurdles I faced during my project.

DEDICATION

This work is dedicated to my parents and my loving brother.
For always believing in me.

Chapter 1

Introduction

In recent years, the rapid growth of Internet has resulted in an exponential increase in the amount of data that can be accessible. With this boom in data availability, graphs have become extremely important as they are powerful means to visualize the relationships in data and can provide decisive insights. The emerging social networks have transformed the way we interact with the rest of the world. These networks have become forums for the netizens to express their views and opinions on almost every aspect of human life ranging from entertainment to political views. All these interactions and relationships can be mapped to a graph that is self-contained with an enormous amount of vital information. However, the size of such graphs can be gigantic. For example, the number of active monthly users on popular social network, *Facebook*, are 2.2 billions [34]. Thus, extracting crucial information from such enormous graphs is a big challenge.

1.1 Motivation

Enumerating triangles is one of the paramount tasks in graph analysis. As discussed in [26], triangles in a social graph are generated either because of homophily or transitivity. The former means that people with similar interests are friends with each other [32] [29], and according to the later people with mutual friends tend to become friends themselves [29]. Thus, the knowledge of triangles in a social graph, we can detect people with similar interests. With access to this information, one can easily target the audience with similar interests and utilize it for marketing various products and services.

Besides this, two fundamental metrics for graph analysis, clustering-coefficient, and transitivity ratio (also known as global clustering coefficient) [15] are based on the density of triangles in a graph. These measures are highly relevant to quantify the degree of clustering in a graph and also to evaluate the stability of a graph [7]. They have been widely used for various applications such as differentiating normal networks and tumor networks by using gene expression data arising from microarray experiment [10], identifying fake accounts in social networks [33] and measuring potential risks in complex banking networks [24].

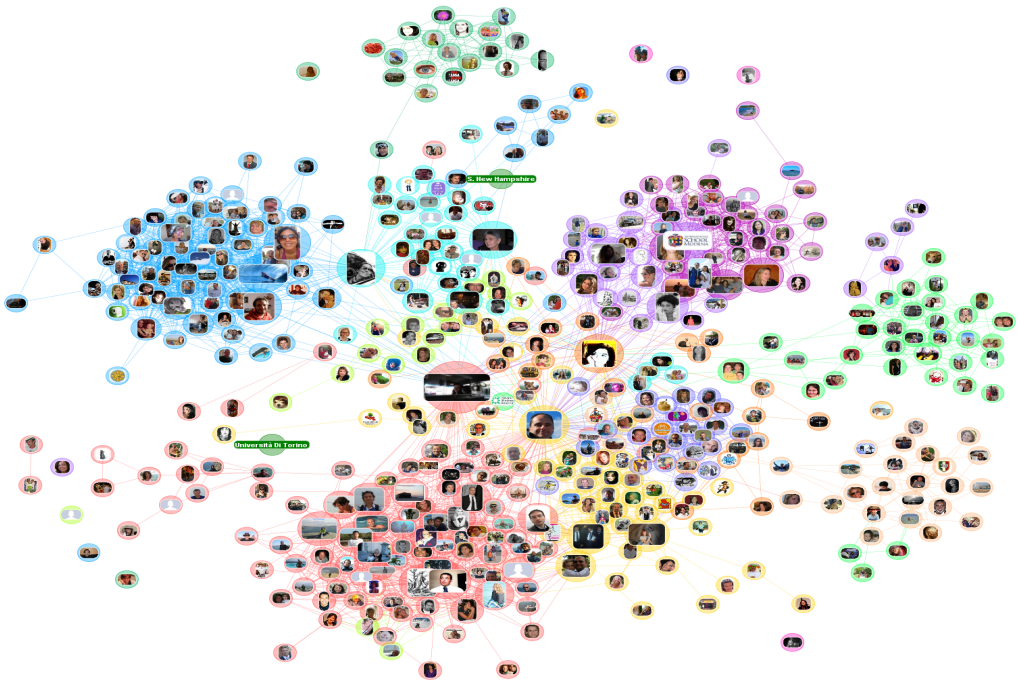


Figure 1.1: A Sample Graph of Social Network Communities [28]

Lastly, one of the most useful applications of these metrics is that they help to estimate the dense sub-graphs (communities) within the graph and also the extent of connectivity within each community [8]. Figure 1.1 represents a social network as a graph. In this graph, we can observe clustering of different communities. There are broad applications of detecting such communities including, building efficient product recommendation system utilizing the community of customers with similar interests [1], clustering the web-clients geographically for improving the performance [11] and identifying group of correlated genes in a biological network [31].

1.2 Contribution

The goal of this project is to explore, analyze and extend the Pre-partitioned Triangle Enumeration (PTE) MapReduce algorithms suggested by Park et al. in [18].

Park et al. proposed three versions of PTE algorithm, namely PTE_{base}, PTE_{cd} and PTE_{sc} for listing triangles in undirected graphs. As part of this project, we propose the extended version of all three PTE algorithms for calculating two main type of triangles in a directed graph: Trust and Cycle (explained in chapter 3). Specifically, the contributions of this project are as follows:

1. We implement the PTE_{base}, PTE_{cd} and PTE_{sc} algorithms as proposed in [18] for undirected graphs. We also perform an evaluation of these algorithms.

2. We extend the PTE algorithms for directed graphs. We propose the algorithm for cyclic-triangle enumeration and trust-triangle enumeration in directed graphs.

3. We conduct extensive experiments on a variety of real-world network datasets to evaluate the performance of the PTE algorithms. Our experiments are setup for both, directed and undirected graphs. The largest directed graph used for experimentation has 639 billion edges.

1.3 Organization

Chapter 1 includes a brief introduction about the relevance of triangle enumeration, the motivation behind taking this project and the contributions.

Chapter 2 provides the literature review for the triangle enumeration problem. It provides a brief description of the major contributions made in the field of triangle computation. It also talks about the limitations and advantages of each approach.

Chapter 3 talks about the background knowledge for triangle enumeration problem. It contains basic terminologies and concepts of graph theory including the type of triangles.

Chapter 4 describes the Pre-partitioned Triangle Enumeration (PTE) algorithms in detail. It explains each of the three PTE algorithms along with their limitations.

Chapter 5 explains the extension of Pre-partitioned Triangle Enumeration algorithm for directed graphs. It is divided into two sections covering the two possible types of triangles in directed graphs.

Chapter 6 contains the experimental results which show the triangle enumeration done on some large real-life datasets. It also provides the comparison of all three PTE algorithms with each other.

Chapter 7 concludes the problem statement and the contributions made. It also contains the discussion about possible future work in this area.

Chapter 2

Related Work

This project is inspired by the extensive research in progress to solve triangle enumeration problem. Finding the triangles in large graphs holds a crucial role in graph analysis. Two fundamental algorithms to perform triangle enumeration on single machine are *node-iterator* and *edge-iterator* algorithm. The *node-iterator* algorithm determines the neighboring nodes for each node and verifies whether the neighboring nodes are connected to each other. In a similar fashion, the *edge-iterator* algorithm traverses through the edges. For each edge (u, v), the algorithm identifies the common neighbors of the nodes u and v to find the triangles. There also have been several proposed refinements of these algorithms [13]. *Forward* algorithm and the *compact-forward* algorithms propose the improved version of *edge-iterator* algorithm by which there is no need to compare all the neighbors of the nodes u and v. These algorithms have been discussed in detail by T. Schank in [20]. However, single machine algorithms can only work when the graph is small enough to fit in the main memory.

It is possible to tackle large graphs by utilizing the power of parallel computing. Map-Reduce is a parallel distributed programming model which has been designed to deal with large datasets [6]. A MapReduce framework consists of three main steps: Map, Shuffle and Reduce. Map step transforms the data into a tuple form, that is, each instance is converted to a key-value pair. The shuffle step bounds the output by the keys. The last step is the reduce step, combines the values with same keys across all the partitions and outputs the reduced key-value pair. Hadoop is a Java-based open source implementation of MapReduce Framework and is widely used because of its availability, scalability, and cost-effectiveness. Spark [35], another open source framework is a cluster computing tool which can run on top of Hadoop and enhances

its computation speed. Ralf Lömmel provides a rigorous description of Google's map-reduce programming paradigm in [12].

The first map-reduce algorithm to enumerate triangles was proposed by Cohen [5]. The proposed algorithm is a modification of node-iterator algorithm [21]. He suggested to use two map-reduce jobs to enumerate triangles; the first step maps the edges to lower degree vertex, then reduces it to emit each pair of edges with that vertex as the apex. This leads to finding all the triads in the graph. The next map-reduce job finds if the open vertex ends of triads are connected to each other, and hence, finding the triangles. However, the performance of this algorithm reduces for graphs with a large degree of nodes.

Doulion [26], another map-reduce algorithm proposed for counting triangles uses a randomized probabilistic approach. Along the similar lines, [16] and [27] also use some sampling techniques to reduce the number of edges and get the approximate number of triangles. However, the approach cannot be used to list the triangles and can only be used to get a rough estimate of the number of triangles.

Suri and Vassilvitskii [23], proposed a map-reduce algorithm, GP, based on partitioning the graph to list the triangles in gigantic graphs. The idea is to partition the graph into overlapping subsets of sub-graphs such that each triangle in the graph comes in at least one of the partition. Dealing with small sub-graphs solves the memory capacity problem of [5]. After partitioning the graph into n partitions, GP algorithm considers the edges in three partitions at a time and enumerates triangles in the merged sub-graph using any internal memory algorithm. This process is repeated until all three partition combinations are covered. For instance, for a graph divided into 4 partitions(1, 2, 3 and 4), the triangles are enumerated in the subgraphs created by following partition combinations: $(1, 2, 3)$, $(1, 2, 4)$, $(1, 3, 4)$ and $(2, 3, 4)$. However, GP algorithm performs a weighted count of the triangles, has a lot of redundant operations which results in creation of unwanted intermediate data and thereby increasing the processing time.

To rectify the duplication problem in GP algorithm, Park and Chung [17] proposed a new triangle enumeration Map-Reduce algorithm, Triangle Type Partition (TTP). After partitioning the graph, TTP classifies the triangles into three types, $type-1$, $type-2$ and $type-3$ triangles. This classification is based on the partition in which the nodes of a triangle belong to. The triangles with all three nodes in the same partition are classified as $type-1$ triangles, triangles with two nodes in one partition and one node in another partition as $type-2$ triangles and the triangles with each

node in a different partition as $type - 3$ triangles. Type of these triangles have been discussed in detail in chapter 3 because of its relevance to this project. TTP separates the enumeration of $type - 1$ and $type - 2$ triangles from the enumeration of $type - 3$ triangles. Thus, avoiding the duplicate counting experienced in GP and reducing the computation time. However, in case of very large graphs, the algorithm causes out of memory error when the size of shuffled data is larger than the total available space.

Park et al. [19] proposed a multi-round MapReduce algorithm Colored Triangle Type Partition (CTTP) to reduce the amount of shuffled data. The idea is splitting the TTP algorithm into multiple rounds and thus limiting the shuffled data in each round and increasing the performance. However, the net shuffled data across all rounds remains the same as TTP.

Park et al. [18] proposed a new MapReduce algorithm, Pre-partitioned Triangle Enumeration algorithm which significantly reduces the shuffled data. They present three versions of the PTE algorithm. All three PTE algorithms have been discussed in detail in chapter 4. The first is the PTE_{base} algorithm which separates the graph-partitioning from triangle enumeration. After partitioning the graph, the set of edges are stored in a distributed file system because of which results in reducing the shuffle time considerably. The second algorithm PTE_{cd} extends PTE_{base} algorithm and considers color-direction of each of the edges to reduce redundant operations. The final algorithm, PTE_{sc} schedules triangle computation carefully to reduce the net network read. However, the PTE algorithm has been implemented just for triangle computation in undirected graphs. As a part of this project, we have implemented the PTE algorithms for undirected graphs and also extended the algorithm for directed graphs. The extended algorithms have been explained in detail in chapter 5. The next chapter covers the preliminary information required to understand this project.

Chapter 3

Preliminaries

The importance of finding triangles and the purpose of this project has been discussed in detail in previous two chapters. This chapter provides the background information and the terminologies that are essential in understanding the future chapters. Section 3.1 covers the basics of graph theory which are widely used throughout the course of this project report. The next section (section 3.2), explains the concept of graph partitioning and assigning types to the triangles after partitioning as per [17].

3.1 Graph Theory

A *graph* is a representation which is used to define the relationship between two entities. The study of the graphs is referred to as Graph Theory. The theory of graph is of immense importance primarily because of the wonderful new insights a graph can lead to and also because of the large amount of applications of graphs. Typically a graph consists of two entities, vertices and edges, and can be represented as $G = (V, E)$ which implies that graph G is composed of the vertex set V and edge set E. The $out - degree$ of a vertex can be defined as the number of edges going out from that vertex, and the $in - degree$ of a vertex can be defined as the number of edges going towards that vertex. Figure 3.1 shows two graphs, the one on the left is directed, and the one on the right is undirected. In each of these graphs, the nodes labeled a, b, c, d, e are the vertices and the lines connecting them are called the edges. The Undirected and Directed graphs can be defined as:

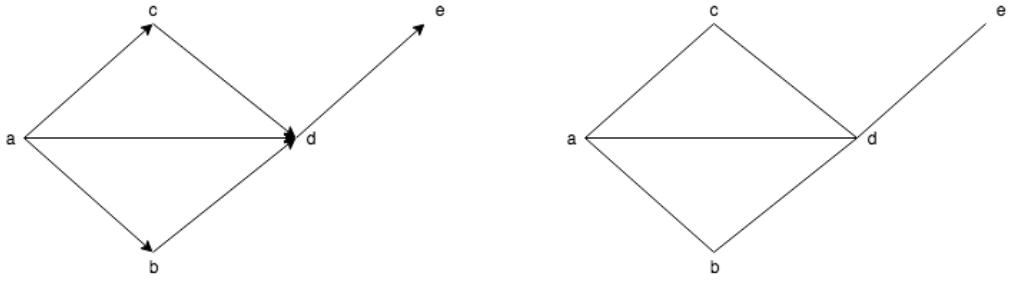


Figure 3.1: Undirected and Directed Graph Example
The graph on the left represents a directed graph, and the graph on the right represents an undirected graph.

3.1.1 Undirected Graph

When the edges of the graph have no direction, the graph is referred to as undirected graph (right side of the figure 3.1). A real-life application of undirected graph is a graph representing friends on social media such as Facebook, if we say a is friends with b, this implies b is also friends with a, the relationship between a and b can easily be defined using an undirected graph.

3.1.2 Directed Graph

When the edges of the graph have direction, the graph is referred to as directed graph (left graph of figure 3.1). Directed Graph can be used to represent the cases when vertexes are connected to each other, but both vertexes do not share the same sort of relationship. For instance, the social-network graph of Twitter can be represented as a directed graph, and an edge from a directed towards c, can be used to depict that a follows c, but the reverse is not true, i.e. c does not follow a.

3.1.3 Triangles in Undirected and Directed Graph

A triangle can be defined as a closed figure with three vertexes and three edges. In Figure 3.1, graph on left and right, both contain two triangles each, (a, c, d) and (a, b, d). The immense applications of finding such figures have been discussed in the previous chapters. The type of triangles differs in a graph depending on whether its directed or undirected, figure 3.2 shows the different variants of the triangles in such graphs. These variants have been discussed in detail below:

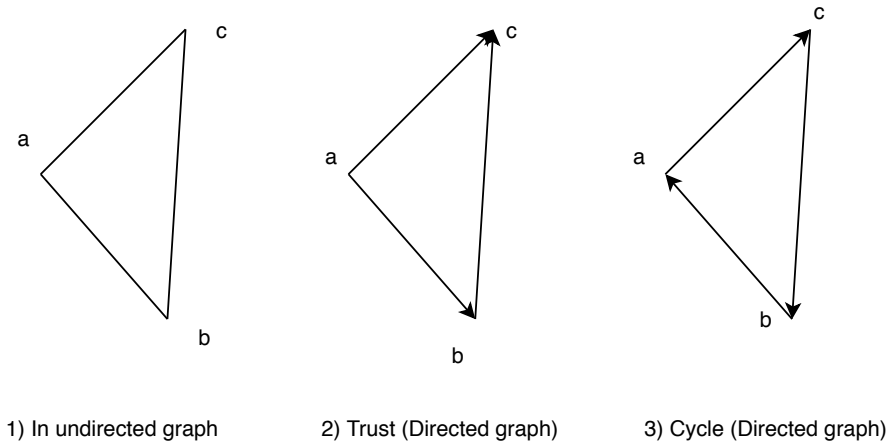


Figure 3.2: Triangles in directed and undirected graphs

Undirected Graph

In an undirected graph, a triangle can be defined as any three vertexes fully connected to each other. The leftmost graph of figure 3.2 represents a triangle in an undirected graph. In this triangle, there exist edges from a to b and c, b to a and c and c to a and b, which implies that the figure is a fully connected, three vertex closed figure and hence a triangle.

Directed Graph

A closed figure with three vertexes and three directed edges is a triangle in a directed graph. The middle graph and the right graph of figure 3.2 represent the triangles in a directed graph. Depending on the direction of the edges, the directed triangles can be further categorized into two types.

Trust Triangles Three-connected vertexes in a graph forming a closed figure, in a way that the out-degree of each of these vertexes are two, one and zero respectively, forms a trust triangle [25]. The middle graph of figure 3.2, represents a trust (a, b, c) with $a's$ out-degree as two, $b's$ out-degree as one and $c's$ out-degree as zero.

Cycle Triangles Three-connected vertexes in a graph forming a closed figure, in a way such that the out-degree of each of these vertexes is one, forms a cycle [25]. The right most graph of figure 3.2 represents a cycle. Each of the vertexes a, c and b,

have one edge going out of them, (a, c), (c, b) and (b, a) respectively. Thus, the graph (a, c, b) represents a cycle.

3.2 Graph Partition and Triangle Types

This section covers in detail the essence of graph partition and how triangles can be divided according to that. As the name suggests, the Graph Partition means partitioning the graph in a way such that only a portion of the graph is read for triangle computation. The reason for partitioning the graph is that because of the large amount of data; some graphs are so big that its hard to compute triangles in the entire graph at once. Thus, [23] suggested that such graphs can be partitioned in a way that each triangle appears in at least one partition. For triangle enumeration, only a portion of the graph is loaded, and triangles are computed in that. Depending on whether the vertexes of the triangle are in the same partition or different partitions, triangles can be categorized into different types [17] shown in figure 3.3. Section contains the description of how they are differentiated. Further, [23] divides the sub-problems in two types depending on the type of triangles being enumerated. Details of this have been discussed in section 3.2.2.

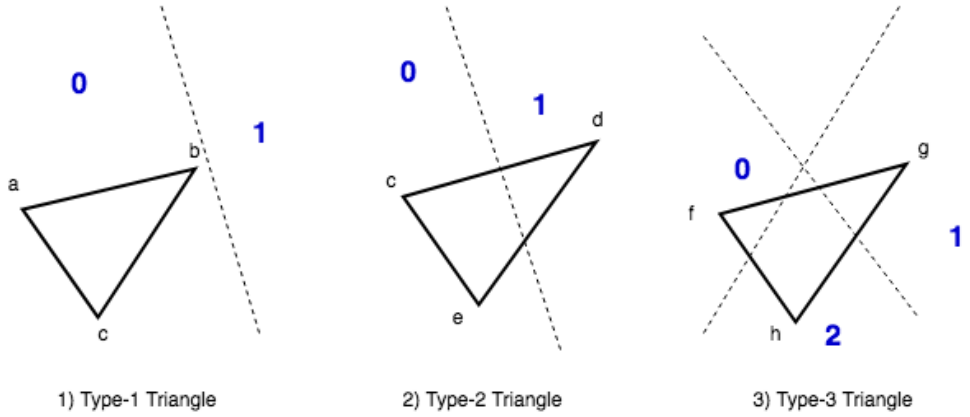


Figure 3.3: Type of triangles in partitioned graph

3.2.1 Partitioned-Graph Triangle Types

To reduce redundant operations in GP [23], Park and Chung [17], propose a map reduce algorithm, Triangle Type Partition (TTP), which categorizes triangles into

following three categories:

Type-1 Triangles When all three vertexes of the triangle are in same partition, the triangle is referred to as type-1 triangle. In the figure 3.3, the left most graph has two partitions 0 and 1, and a type-1 triangle (a, b, c) with all three vertexes in same partition ($partition\,0$).

Type-2 Triangles With two vertexes of the triangle in same partition, and one in different partition, as triangle (c, d, e) in the middle graph of figure 3.3 with vertex c, e in $partition\,0$ and vertex d in $partition\,1$, the resulting triangle can be categorized as type-2 triangle.

Type-3 Triangles The last case covers the triangles with all three vertexes in three different partitions (as shown in the last graph of the figure 3.3). The type-3 triangle (f, g, h) has vertex f in $partition\,0$, vertex g in $partition\,1$ and vertex h in $partition\,2$.

3.2.2 Partition-Pair and Partition-Triplets

After categorizing the triangles into above three categories, TTP [17] divides the triangle enumeration problem into sub-problem of two types, one with considering two-partitions at a time (referred to as (i, j) subproblem by TTP, where the two partitions considered are i^{th} and j^{th} partition) and the other one with considering three-partitions at a time (referred to as (i, j, k) subproblem by TTP, with partitions considered being i^{th}, j^{th} and k^{th} partition). Throughout the course of this project, triangle enumeration in these two types of sub-problems has been referred to as triangle enumeration in partition-pair (referring to (i, j) sub-problem) and partition-triplet (referring to (i, j, k) sub-problem). Figure 3.4 gives a good example to explain the concept of partition-pair and partition-triplet, the details of which have been discussed below:

Partition-Pair The edge set in a partition-pair (i, j) sub-problem (as shown in equation (3.1)) comprises of all the edges with both vertexes in i^{th} partition, edges with both vertexes in j^{th} partition and the edges with one vertex in i^{th} and one vertex in j^{th} partition. Triangle enumeration on this edge set results in finding all type-1

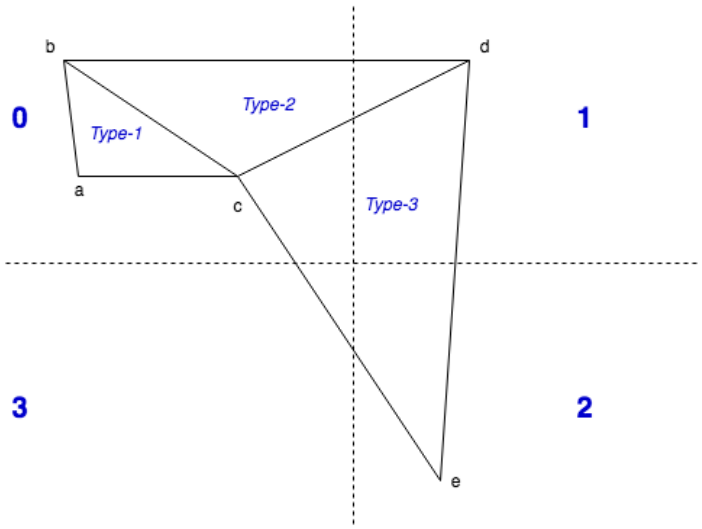


Figure 3.4: Triangle enumeration in partitioned graph

triangles in i^{th} and j^{th} partition and type-2 triangles in the inter-partition i, j.

$$E_{ij} = E_{ij} \cup E_{ii} \cup E_{jj} \tag{3.1}$$

Partition-Triplet In a partition-triplet (i, j, k) sub-problem, the edge set comprises of just the inter-partition edges that is, edges with one vertex in i^{th} partition, other in j^{th} partition, edges with one vertex in j^{th} partition, other in k^{th} partition and the edges with one vertex in k^{th} partition and the other in i^{th} partition (as shown in in equation (3.2)). Thus, enumerating triangles in such sub-problem results in enumerating type-3 triangle enumeration in that partition-triplet.

$$E_{ijk} = E_{ij} \cup E_{ik} \cup E_{jk} \tag{3.2}$$

This chapter covered the necessary information which is essential to understand before proceeding further. The next chapter gives a detailed description of the Pre-partitioned Triangle Enumeration algorithm for undirected graphs.

Chapter 4

Methodology

This chapter explores the map-reduce triangle enumeration methods suggested by Park et al. in [18]. The algorithms proposed in the paper perform considerably well in comparison to the previous work done along the similar lines. For a large graph, Clue Web 9 (with 7.9 billion edges), the algorithm outperforms CTTP [19] by 27x times. The proposed method uses TTP [17] algorithm as base that is it partitions the graph, classifies triangles as type-1, type-2 and type-3 (mentioned in section 3.2.1) and divides the sub-problems generated into two types (mentioned in section 3.2.2). Based on that, they proposed three algorithms which considerably reduce the computation time.

The first proposed algorithm, PTE_{base}, addresses the key issue of reducing the shuffled data to increase the performance. The key idea is to store the partitioned graph before triangle enumeration. The second proposed algorithm, PTE_{cd} focuses on reducing the redundant operations and hence results in improving the performance further. PTE_{cd} introduces a color-direction to each of the edges, that means direction based on the partition. This helps in further sub-classifying the edges which should be considered together to enumerate triangles in a partition. The final algorithm, PTE_{sc} recommends the approach of scheduling triangle computations to minimize the network read.

Park et al. proposed these algorithms for undirected graphs. As part of this project, we have re-implemented them using spark and have also performed a thorough evaluation of their performance. The results of which have been discussed in detail in chapter 6. In this chapter we discuss the PTE_{base} (section 4.1), PTE_{cd} (section 4.2) and PTE_{sc} (section 4.3) algorithms, their shortcomings and their implementation in detail.

4.1 PTE BASE Algorithm

In this section, we discuss the base algorithm, PTE$_{base}$, proposed in [18]. Previously proposed MapReduce algorithms produce an enormous amount of shuffle data for large graphs, because of which the program fails with out of memory errors when the size of shuffled data is larger than the total available space. PTE$_{base}$ *pre-partitions* the graph and decreases the shuffled data significantly. The idea is to partition the graph and save the sub-graphs in some form of distributed storage. After this, when the algorithm needs some edge-set for enumeration, it directly accesses it from the storage system. Thus the data shuffle is converted to network read which is a much less expensive task. Thereby, resulting in a decrease in computation time, especially for larger graphs which have a massive amount of edges. For a huge graph ClueWeb09 with 7.9 billion edges, PTE$_{base}$ reduces the shuffled data by 175x times [18]. This section describes each step of the algorithm in detail. In the end, we discuss the shortcomings of this algorithm and how can it be further improved.

4.1.1 For Undirected Graphs

Figure 4.1 shows the basic flow architecture of the PTE$_{base}$ algorithm. The steps involved are: get the number of partitions, partition the graph and save the sub-graphs in some file system. The final step is to enumerate triangles by loading the significant edges in memory. Below is a description of each and every step:

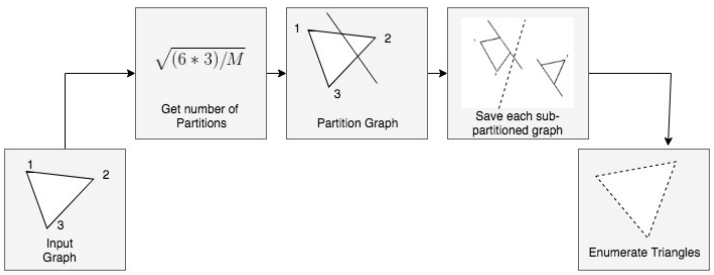


Figure 4.1: PTE Base Flow Architecture

Step 1: Get the number of partitions

Prior to creation of sub-graphs, it is important to decide the number of partitions. As per [18], for ρ number of partitions, the estimated size of edge set for each partition is $2E/\rho^2$. Since for enumeration of type-1 triangle, we need edge

set from one partition, for enumeration of type-2 triangle, we need edge set from two partitions and for enumeration of type-3 triangle, we need edges-set from three partitions. Therefore, the estimated size of edge-set for type-1, type-2 and type-3 triangles are $2E/\rho^2$, $4E/\rho^2$ and $6E/\rho^2$ respectively. As we need these edge-sets to fit in main memory(M), hence:

For Type -1 problem,

$$2E/\rho^2 < M \tag{4.1}$$

For Type-2 problem,

$$2E/\rho^2 + 2E/\rho^2 < M \tag{4.2}$$

For Type-3 problem,

$$2E/\rho^2 + 2E/\rho^2 + 2E/\rho^2 < M \tag{4.3}$$

Thus, from (4.1), (4.2) and (4.3) number of partitions should be:

$$max(2E/\rho^2, 4E/\rho^2, 6E/\rho^2) < M$$
$$6E/\rho^2 < M$$

$$\rho = \sqrt{6E/M} \tag{4.4}$$

Step 2: Partition the nodes

The next step is to assign a unique color to each node. The method used to assign color to nodes is:

$$Color\ of\ each\ vertex = Vertex\ \%\ number\ of\ Colors(\rho) \tag{4.5}$$

where each color represents each partition. With this method, each edge u, v gets mapped to the partition-pair x, y where x is the color assigned to u, and y is the color assigned to v

Step 3: Storing the sub-graph from each partition-pair into a distributed storage system

Since all edges are mapped to a partition by now, in this step we store the partitioned sub-graph in some distributed file system. For instance, if the edges

(u, v) and (w, z) are mapped to partition-pair (x, y), they are stored together. When needed, these can be directly accessed from the distributed file system instead of shuffling the data to access the sub-graph.

Step 5: Generate the Seed File

Once we have all the partitioned graphs, the next step is selecting the right partition-pairs/partition-triplets sub-problems to enumerate triangles such that we each triangle is counted only once. Lets consider a toy example to understand this better. Lets assume we have two partitions/colors, 0 and 1. With 2 partition, the possible partition-pair combinations are: 00, 01, 10 and 11. Since we know that partition-pair counts both type-1 and type-2 triangles (equation (3.1)), therefore considering just one of these partition-pair (either 01 or 10) would be a wise decision. To make the selection simple, we have used the condition $i < j$ (in case of a pair) and $i < j < k$ (in case of a triplet). We can choose the pair/triplet in any order, all we need is to avoid any duplicity during enumeration.

Step 6: Enumerate Triangles

For each sub-problem listed in the seed file, read the corresponding sub-graphs and enumerate triangles. A partition-pair sub-problem finds triangles of type-1 and type-2 (reading the edge-sets as in equation (3.1)). A partition-triplet sub-problem, finds the type-3 triangles (reading the edge-sets listed in equation (3.2)).

Once we have read the subgraph, we can use any single machine algorithms to enumerate triangles.

4.1.2 Redundant Operations in PTE BASE algorithm

While the PTE_{base} algorithm reduces the computation time considerably, but still it performs some redundant operations which if avoided, can result in significant performance improvement. In this section, we are going to discuss first the problems in PTE_{base} and then the solution to overcome them. To understand redundant operations in PTE_{base}, lets try to find triangles in the undirected graph as shown in figure 4.2 using PTE_{base} algorithm.

Figure 4.3(a) shows the partitioned graph and the generated seed file. The number of partitions is decided as per Step 1, equation (4.4). Next, the graph is partitioned

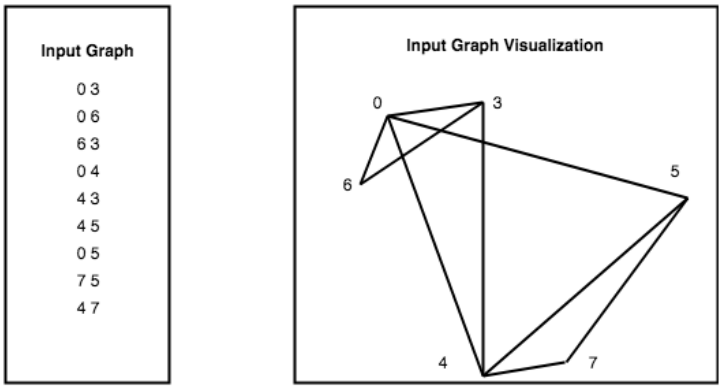


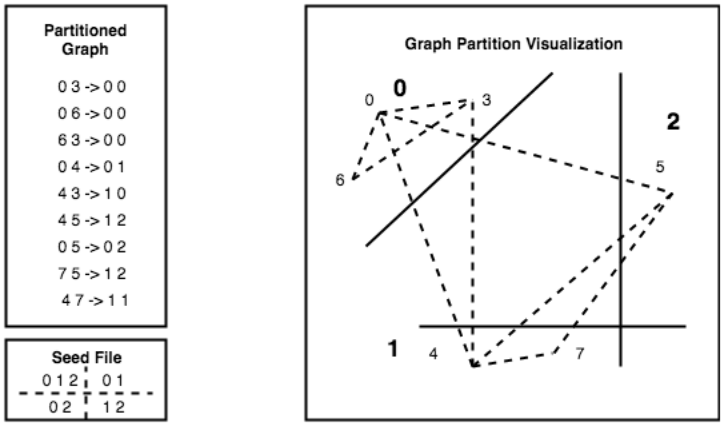
Figure 4.2: Graph Example

as per equation (4.5). After that, edges mapped to the same partition-pair are stored together. For instance, partition-pair $1, 2$ stores the edges $(4, 5)$ and $(7, 5)$. In figure 4.3(b), since the number of partitions are three, the seed file has four values (calculated using procedure mentioned in Step 5). For the seed input $(0, 1)$, the edges loaded in memory are the ones belonging to partition $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. In the similar fashion, edges are loaded for partition-pair $(0, 2)$ and $(1, 2)$ as per equation (3.1). For the partition-triplet, $(0, 1, 2)$, all the cross partition edges (across partition 0, 1 and 2) are read (equation 3.2). Based on the edges read, the triangles are enumerated for each seed input, but there are two critical problems in using PTE Base Algorithm:

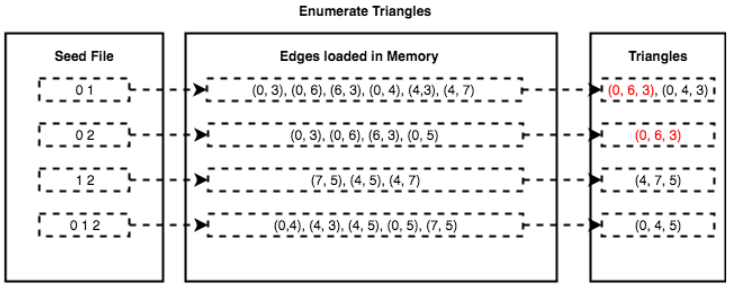
1. Redundant Triangle Counting For Type-1 Triangles

   The issue with type-1 triangle enumeration using PTE_{base} algorithm is that type-1 triangle present in j^{th} partition appears every time the partition-pair sub-problem with j as one of the partition appears in seed file. This implies, that such triangles get enumerated $\rho - 1$ times, where ρ is the number of partitions. For instance, in Figure 4.3, the type-1 triangle$(0, 6, 3)$ in 0^{th} partition, gets counted two times (since ρ is 3). The solution to this scenario is simple, assuming that there are k number of type-1 triangles in a given graph, and each such triangle appears $\rho - 1$ times, giving a total type-1 triangle count of l, the actual number of type-1 triangles (k) can be calculated as:

   $$k = l/(\rho - 1)$$



(a) Partitioned Graph



(b) Triangle enumeration in sub-graphs

Figure 4.3: Triangle Enumeration using PTE_{base}

But still there is redundant computation of such triangles which adds up in the overall computation time.

2. Access edges loaded in memory for Type-2 and Type-3 Triangles

   The second issue is while enumeration of type-3 triangles, the algorithm considers unnecessary access edge combinations. To understand this using figure 4.3, for the type-3 triangle, $(0, 4, 5)$, if edge a is $(4, 0)$, edge b or c can not be $(4, 3)$, since it lies in the same partition-pair. However, PTE_{base} doesn't rule out this possibility and considers the edges in same partition-pair too, which eventually affects the enumeration time of the algorithm.

The next section explains how considering the color direction of edges can help in reducing the enumeration time.

4.2 PTE CD algorithm

The problems with PTE_{base} algorithm can be solved if each edge (a, b) is assigned a direction from a to b. The same rule can be applied to a partition pair, each partition-pair (x, y), can be assigned a direction from x to y. The idea is to use only the edges with the desired direction and enumerate triangles accordingly . All possible color direction for type-2 and type-3 triangles are listed in the table 4.1. Thus, while enumerating triangle (a, b, c) of any type, let's consider the color-direction (i, i, j) , the edge a should belong to partition-pair (i, i), similarly edge b to partition-pair (i, j) and edge c to (j, i). Same rule applies to all the combinations listed in table 4.1.

Type-2	**Type-3**
(i, i) , (i, j) , (i, j)	(i, j) , (j, k) , (i, k)
(i, j) , (j, i) , (i, i)	(j, i) , (i, k) , (j, k)
(j, i) , (i, i) , (j, i)	(k, i) , (i, j) , (k, j)
(j, j) , (j, i) , (j, i)	(i, k) , (k, j) , (i, j)
(j, i) , (i, j) , (j, j)	(j, k) , (k, i) , (j, i)
(i, j) , (j, j) , (i, j)	(k, j) , (j, i) , (k, i)

Table 4.1: Color-Direction combinations for undirected graph

Considering the color-direction solves the huge problem of using access edges for type-2 and type-3 triangles. To solve the $\rho - 1$ redundant enumeration of each Type-1 triangle, the algorithm filters out the other conditions and enumerates triangle in i^{th} partition only when

$$i + 1 = j \; mod \; \rho \tag{4.6}$$

This ensures that each type-1 triangle gets counted only once.

4.2.1 For Undirected Graphs

To see how PTE_{cd} works on the undirected graphs, let's reconsider the example in figure 4.2. For each edge (a, b) in the graph, we'll consider the color direction from a to b. After reading the seed file, the edges loaded in the memory are same, but only a partition/chunk of those edges are sent for enumerating triangle (based on the color direction). Figure 4.4 illustrates the process of reading input from seed file, loading edges to the main memory and then enumerating triangles based on the color direction. The input graph consists of all three types of triangles. Type-1 triangle $(0, 6, 3)$, gets enumerated only once, when the equation (4.6) gets satisfied i.e., $i = 0$

and $j = 1$ (since number of partitions $\rho = 2$). The color direction for type-2 triangle $(0, 4, 3)$, can be considered as (i, j, i) where $i = 0^{th}$ *partition* and $j = 1^{st}$ *partition*. Thus, for this triangle, the edges a, b and c should be from partition-pair $(0, 1)$, $(0, 0)$ and $(1, 0)$ respectively (as shown in Figure 4.4). Lastly, the type-3 triangle $(0, 4, 5)$ gets counted when counting the triangle for the color-direction, (i, j, k), where $i = 0^{th}$ *partition*, $j = 1^{st}$ *partition* and $k = 2^{nd}$ *partition*. The edges a, b and c for this triangle come for partition-pair $(0, 1)$, $(1, 2)$ and $(0, 2)$.

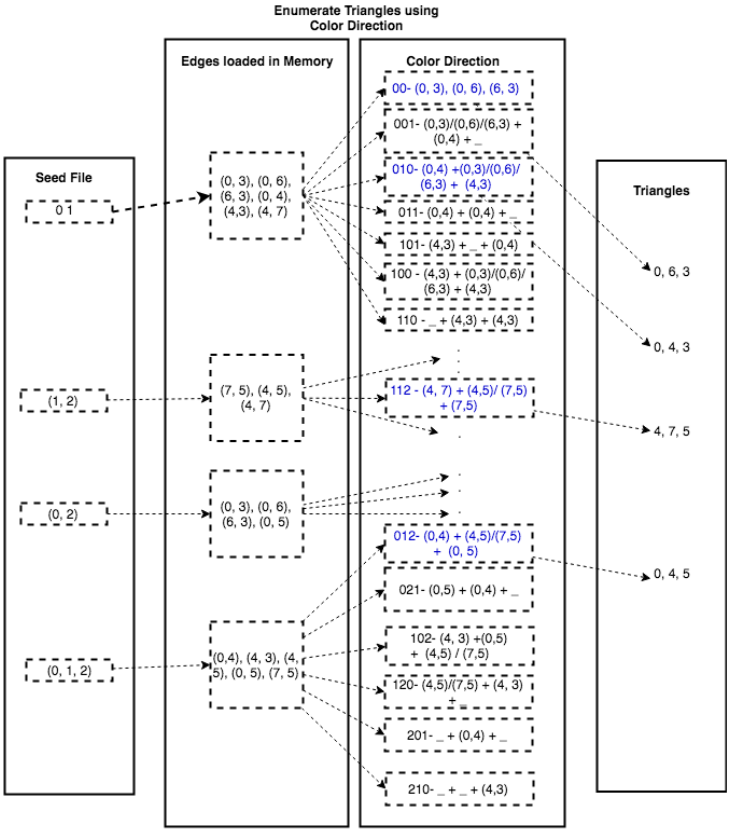


Figure 4.4: Enumerate Triangles using PTE CD

Thus, this further distributes the triangle into categories based on the color-direction, eventually reducing the number of edges needed for any triangle computation.

4.2.2 Excessive Network Read in PTE CD

Although PTE Color Direction reduces the redundant operations, still the amount of network read can be reduced further. The problem is that for a type-3 triangle, all the

edges loaded are not required for all six color direction triangle computation. However, PTE^{CD} never releases those edges and hence they occupy memory unnecessarily. For instance, in table 4.1, edges in partition-pair (i, j) aren't required by Color Direction, $[j, i, k]$, $[j, k, i]$ and $[k, j, i]$ (row 2, 4 and 5 of Type-3 column), thus they can be released before computation of triangles in these color direction, reducing the network read.

4.3 PTE SC algorithm

PTE Scheduling calls address the problem mentioned above with PTE CD by re-ordering the type-3 Color Direction Triangle Enumeration in a way that the color-directions requiring same edges in same partition-pair are grouped together, so that the memory can be released after computation of that set of color-directions. Table 4.2 shows the data loading scheduling order in PTE SC algorithm. The rows in table 4.2 depict the edges in the partition-pair and the columns depict the triangles in the color-direction . A check-mark in n^{th} row with m^{th} column, represents that the n^{th} edge-set is required to enumerate the m^{th} triangle. For instance, the first row of check-marks can be interpreted as the edges in partition-pair (i, j) are required for triangle enumeration in color-directions, $[i, j, k]$, $[i, k, j]$ and $[j, i, k]$. Thus, by scheduling this way, the edges in partition-pair (i, k) can be released after triangle enumeration in color direction $[i, j, k]$, $[i, k, j]$ and $[j, i, k]$ and so on.

	$\triangle_{ijk}$	$\triangle_{ikj}$	$\triangle_{jik}$	$\triangle_{jki}$	$\triangle_{kij}$	$\triangle_{kji}$
E_{ij}	✓	✓			✓	
E_{ik}	✓	✓	✓			
E_{ji}			✓	✓		✓
E_{jk}	✓		✓	✓		
E_{ki}				✓	✓	✓
E_{kj}		✓			✓	✓

Table 4.2: Data Loading Schedule in PTE SC

4.3.1 For Undirected Graphs

The enumerate triangles process for PTE_{SC} can also be explained using figure 4.4, just like PTE color direction, the input is read from the seed File, and then the corresponding edges are loaded in the memory, after that the edges are sent for enumeration based on the color direction, with just a slight variation in the order of

color-direction calls for type-3 triangles (typically the last seed input of figure 4.4). As per table 4.2, the order of color-direction will change to: (0,1,2), (0,2,1), (1,0,2), freeing up the edge-set 0,2 and then considering the color-direction (1,2,0) (2,0,1) and (2,1,0) . Thereby, reducing the network read and the computation time.

Chapter 5

Extended Implementation for Directed Graphs

Park et al. in [18] have given the Pre-partitioned Triangle Enumeration algorithm for undirected graphs. As part of this project, the PTE algorithm has been extended for directed graphs. This chapter is divided into two sections. The first section gives the modified PTE_{base}, PTE_{CD} and PTE_{SC} algorithms to list all the cycles in a graph. The second section describes how existing PTE algorithms can be leveraged for trust enumeration in a directed graph. The results of this implementation have been discussed in chapter 6.

5.1 Cycle-Triangle Enumeration

The definition and the structure (the rightmost graph of figure 3.2) of cycle directed triangles has been discussed in Chapter 3. Listing these structures in a social networking graph can help in providing deep insights about its stability [7]. The application of cycle enumeration also lies in detecting a deadlock in any distributed system [4]. Presence of a cycle (a, b, c) implies that a wants some resources which are currently allocated to b, the resources b wants are allocated to c, and c is waiting for resources held by a, thus resulting in a deadlock. Considering the immense applications of cycle listing, there has been a lots of research going on about how to optimally detect cycles in massive graph. This section covers the details of the new extended Base, Color Direction and Scheduled Calls algorithms for cyclic-triangle listing.

5.1.1 PTE BASE

Before getting into the details of how PTE_{base} algorithm can be tweaked to enumerate cycles, it is important to understand the process of triangle enumeration in a single machine. As discussed in Step 6 of section 4.1.1, once we have the sub-problem and the sub-graph associated with it, the triangle enumeration can be done using any single machine algorithm. This section illustrates the enumeration using edge-iterator algorithm which was introduced in [9] and discussed in [21].

Algorithm 1 *Edge − Iterator Algorithm*

```
1: A(G) ← Adjacency List representation of G
2: N(u) ← Neighbors of any vertex u in Graph G
3: for all vertex u ∈ A(G) do
4:     for all vertex v ∈ N(u) do
5:         for all w ∈ N(u) ∩ N(v) do
6:             Enumerate triangle {u,v,w}
```

The input to the algorithm 1 is the the adjacency list representation of the sub-graph G (where G contains the edges of the partition-pair/partition-triplet sub-problem being considered). The output is the triangles present in that sub-problem.

Example Figure 5.1 shows a sample subgraph along with its adjacency list and adjacency matrix representation. To find the trust triangles in this graph, we can use the $edge - iterator$ (algorithm 1). The input to the algorithm $A(G)$ can be the adjacency list representation. For each of the vertex u, find the neighbor list for each of $u's$ neighbor v, and intersect $N(u)$ and $N(v)$ to list the triangle. For the sample graph:

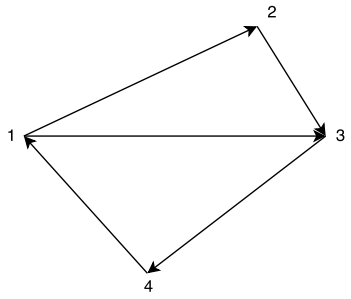
$$N(1) \cap N(2) = (2,3) \cap (3)$$

results in listing the triangle $(1,2,3)$ and

$$N(1) \cap N(3) = (2,3) \cap (4)$$

returns ϕ. Similar process is repeated for all the vertex in adjacency list and results in ϕ, since there is only one trust triangle present. However, this algorithm can not capture the cyclic-triangle $((1,3,4))$.

The core concept behind edge-iterator algorithm is to find a connected edge from



Adjacency Matrix				
	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Adjacency List	
1	2,3
2	3
3	4
4	1

Figure 5.1: Sample Graph

a to b; if there is a common neighbor amongst neighbors of a and neighbors of b, then there exists a directed trust triangle. For instance, consider the edge from 1 to 2, since 3 is a common neighbor of both vertex 1 and 2, edge-iterator finds the triangle $1, 2, 3$. However, the same methodology cannot be used to find the cyclic-triangle $1, 3, 4$, considering the edge 1 to 3, 4 isn't the common neighbor of 1 and 3. Now, if we tweak the algorithm and instead of intersecting neighbors of node 1 and node 3, we intersect the neighbors of adjacency list of node 1 in transposed graph G with the neighbors of node 3, we will get node 4 and hence can detect the cycle.

Algorithm 2, sums this up and gives the procedure to find cycles in a directed graph.

Algorithm 2 *Enumerate Cycles in Directed graph*(G)

1: $A(G) \leftarrow$ *Adjacency List representation of* G
2: $A(G') \leftarrow$ *Adjacency List representation of transposed edges of* $G(E_T)$
3: $N(u) \leftarrow$ *Neighbors of any vertex* u *in Graph* G
4: $N(u') \leftarrow$ *Neighbors of any vertex* u *in graph of transposed edges of* G
5: **for all** *vertex* $u \in A(G)$ **do**
6: **for all** *vertex* $v \in N(u)$ **do**
7: **for all** $w \in N'(u) \cap N(v)$ **do**
8: Enumerate cycles {u,v,w}

Figure 5.2 shows two tables, the left one represents the adjacent list of the sub-graph G with edge-set E and the right one represents the adjacent list of the transpose of the sub-graph G with edge-set E_T. To create the edge-set E_T, the direction of each

edge in the graph is reversed. Cyclic triangle is enumerated when:

$$N'(1) \cap N(3) = (4) \cap (4)$$

listing cycle $1, 3, 4$

Adjacency List with E	
1	2,3
2	3
3	4
4	1

Adjacency List with E_T	
1	4
2	1
3	1,2
4	3

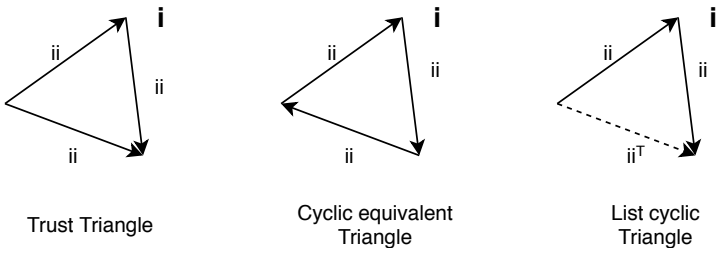
Figure 5.2: Adjacency Lists of Edges and Transposed Edges

Thus, to sum up, PTE_{base} algorithm can be used to find the cycles in a directed graph but we need the adjacency-list of transposed edge-set E (i.e E_T) along with the adjacency-list representation of edge-set E for triangle enumeration (algorithm 2).

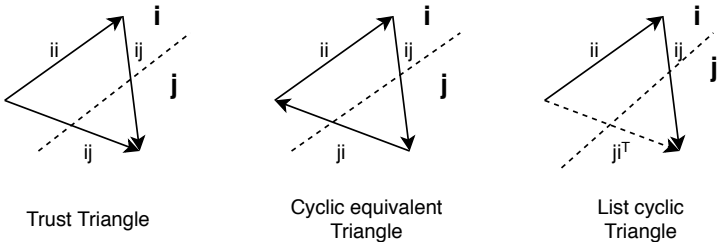
5.1.2 PTE Color Direction

As discussed in section 4.2, PTE_{CD} imparts the color direction to each edge and enumerates triangles accordingly. This section covers how PTE_{cd} can be extended to list the cycles present in the graph. To enumerate a triangle (a, b, c), PTE_{cd} reads edge-sets from three partition-pairs (for instance: ij, ij, ii) such that a belongs to ij, b belongs to ij and c belongs to ii. These edge-sets are read according to the color-direction (as discussed in table 4.1 for Type-2 and Type-3 triangle). The core concept for cycle enumeration using PTE_{CD} is instead of reading three edge-sets from the sub-graph G, two edge-sets are read from G and one is read from transposed graph G^T.

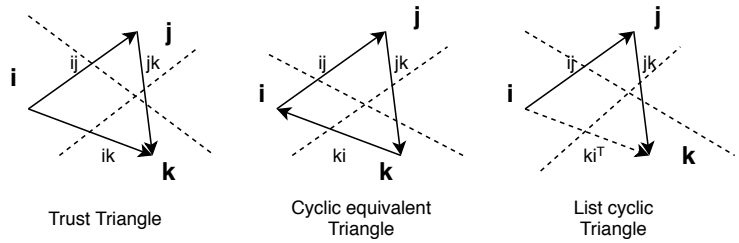
Figure 5.3 shows one example each of type-1, type-2 and type-3 triangle. Each of the sub-figures shows three graphs. The first graph shows the trust enumeration with color-direction (i, i, i), (i, j, i) and (i, j, k) for type-1, type-2 and type-3 triangles respectively. The second graph shows how does a cycle look like in same color-direction. The last graph shows how can we transpose an edge-set and list the cycles present within the same color-direction. For type-2 triangle, the cycle equivalent triangle (a, b, c) for color direction (i, j, i) is the cycle with edge-sets present in partition ii, ij and ji. Instead of edge-set in partition ji, if we consider the transposed edge-set ji^T, as shown in last graph of figure 5.3 (b), we can then list this triangle by $E_{ji^T} \cap E_{ij}$ where E_{ji^T} and E_{ij} represents edge-sets of directional-partition ji^T and ij



(a) Type-1 triangle of (i, i, i) color-direction



(b) Type-2 triangle of (i, j, i) color-direction



(c) Type-3 triangle of (i, j, k) color-direction

Figure 5.3: PTE_{CD} for cycle enumeration

respectively. Similarly, type-1 cycle can be listed by finding the common vertexes in E_{ii^T} and E_{ii}(i.e $E_{ii^T} \cap E_{ii}$) and type-3 cycle with color-direction (i, j, k) can be listed using $E_{ki^T} \cap E_{jk}$. Table 5.1 sums up and shows all possible color-directions and the corresponding edge-sets required for cyclic-triangle enumeration.

Type-2	**Type-3**
(i, i) , $(j, i)^T$, (i, j)	(i, j) , $(k, i)^T$, (j, k)
(i, j) , $(i, j)^T$, (i, i)	(i, k) , $(j, i)^T$, (k, j)
(i, j) , $(i, i)^T$, (j, i)	(j, i) , $(k, j)^T$, (i, k)
(j, j) , $(i, j)^T$, (j, i)	(j, k) , $(i, j)^T$, (k, i)
(i, j) , $(j, i)^T$, (j, j)	(k, i) , $(j, k)^T$, (i, j)
(j, i) , $(j, j)^T$, (i, j)	(k, j) , $(i, k)^T$, (j, i)

Table 5.1: Color Direction Combinations for Cycle Enumeration

5.1.3 PTE Scheduled Calls

PTE_{sc} algorithm (discussed in Chapter 4) also uses the color-direction, but it schedules the sequence of triangle computation in a way that edge-sets which are commonly required by specific color-directions are called together (as shown in figure 4.2). We can leverage the same pattern while enumerating cycles, but the triangle schedule is now different from the schedule for trust enumeration. This is primarily because of the difference in the edges required for cycle and trust listing. Table 5.2 sums up the fairly efficient way to enumerate triangles as per the color-direction. As clearly indicated in the table, the memory used by edge-sets, E_{ik}, E_{ji} and E_{kj} can be released after triangle computation of the color-directions kji, ikj and jik. Thus, this results in speeding up the algorithm further.

	$\triangle_{kji}$	$\triangle_{ikj}$	$\triangle_{jik}$	$\triangle_{jki}$	$\triangle_{kij}$	$\triangle_{ijk}$
E_{ij}				✓	✓	✓
E_{ik}	✓	✓	✓			
E_{ji}	✓	✓	✓			
E_{jk}				✓	✓	✓
E_{ki}				✓	✓	✓
E_{kj}	✓	✓	✓			

Table 5.2: Data Loading Schedule for cycles in PTE SC

5.2 Trust-Triangle Enumeration

Section 3.1 gave a quick introduction about trust-triangles and their structure (the middle graph of figure 3.2). Detection of such triangles in a directed-graph can be very powerful in understanding relationships in a social graph. Trust-triangles exhibits transitivity, which implies that if a is connected to b and b is connected to c, that means a should also be connected to c [30]. The property of transitivity is widely studied to understand the nature of a graph and often holds more importance for a social-networking graph rather than other graphs [14]. Also, [29] describes how transitivity is a key structural property for a social graph and can be used to measure how balanced are the relationships in it. This section covers how we can use the existing PTE algorithms for detecting such trust-triangles in huge graphs with reduced computation time.

While describing the problem, Park et al. in [18] explain that PTE algorithm is designed to work for undirected graphs. The way each edge (u, v) is ordered is such that $d(u) < d(v)$ where $d(u)$ and $d(v)$ are degrees of u and v respectively. The reason for such ordering is that in the core, PTE uses Compact-Forward Algorithm [13] which requires the edges to be sorted by decreasing order of degree. We can use the PTE algorithm to identify the trust-triangles just by changing the input and the core single machine algorithm used for triangle identification after partitioning the graph. The input edge of the form (u, v) is such that the edge is directed from u to v. With this, all three PTE algorithms can be utilized to calculate the trust-triangles in a directed graph. The basic steps of the algorithm remain same. Below is a detailed example showing the flow of PTE_{base} algorithm for a directed graph.

5.2.1 Example

Figure 5.4 shows the sample input file where each edge (u, v) is directed from u to v.

Step 1: Find the number of partitions

The number of edges in graph shown in figure 5.4 is 7. Thus, using equation (4.4), the number of partitions is:

$$\begin{aligned} number\ of\ partitions &= max(\sqrt{6 * 7/M}, 2) \\ &= 2 \end{aligned}$$

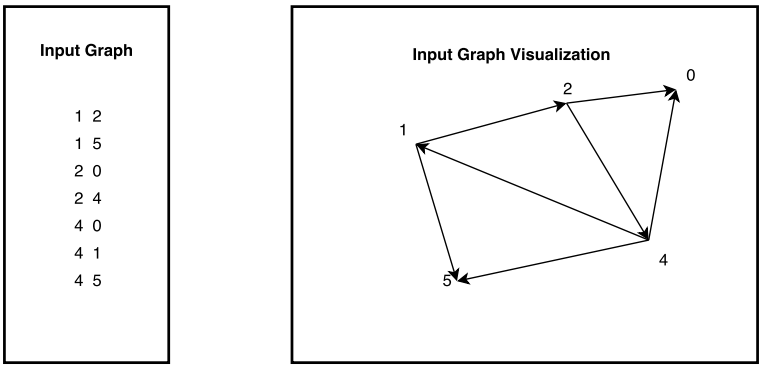


Figure 5.4: Sample graph for Triangle Enumeration

Step 2: Partition Graph to two partitions: 0 and 1

We can use (4.5) equation to assign each vertex to a partition. Figure 5.5 shows the partition created. The vertexes which are on the left side of the solid line are in partition 1 and the vertexes which are on the right side of the solid line are in partition 0.

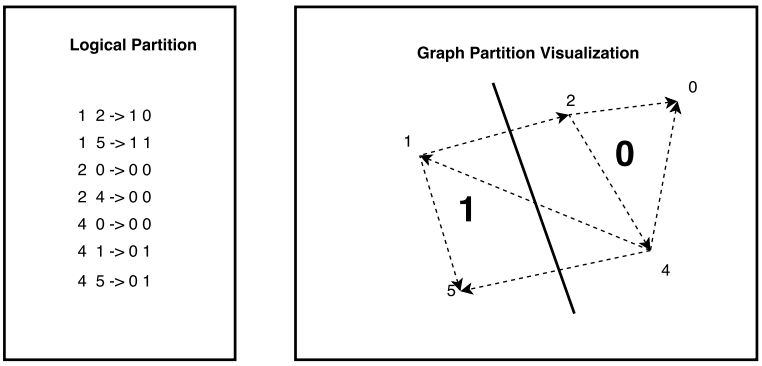


Figure 5.5: Partition Graph based on the edge color

Step 3: Create small sub-graphs and save them

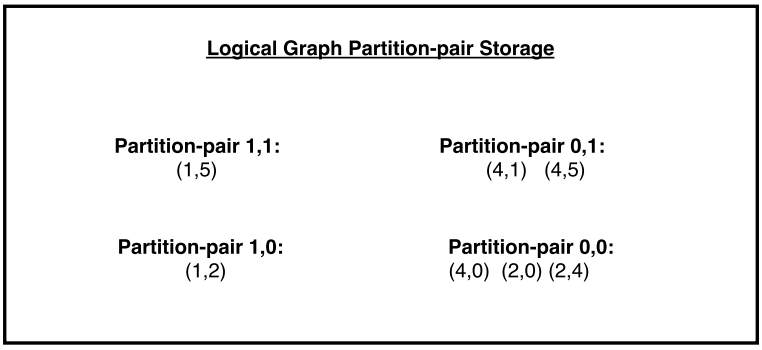
The entire graph is divided into n number of sub-graphs where n depends on the number of partitions. Each partition-pair combination formulates a unique sub-graph and is stored separately. These can be later loaded into the main-memory as and when required for triangle enumeration. Figure 5.8(a) shows the edges saved for each partition-pair.

Step 4: Create seed file and enumerate triangles

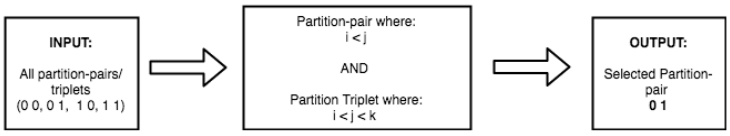
The sub-problems are selected based on the number of partitions (using the logic shown in Figure 5.8(b)). The final step is to select the sub-problems from the seed-file, load the required edge-sets and enumerate triangles for the generated sub-graph(shown in Figure 5.8(c)). In this example, the edge set loaded are:

$$E_{01} = E_{01}{}^{*} \cup E_{10}{}^{*} \cup E_{00} \cup E_{11} (3.1)$$

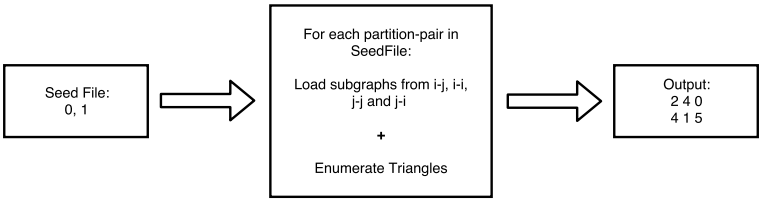
.Hence, resulting in enumeration of the trust triangles $(2, 4, 0)$ and $(4, 1, 5)$.



(a) Save generated sub-graphs



(b) Generate Seed File



(c) Enumerate Triangles

Figure 5.6: Extended PTE CD for cycle enumeration

The above steps give an example of how PTE_{base} can be used to enumerate trust triangles. Similarly, PTE_{cd} and PTE_{sc} can also be leveraged for trust-triangle listing.

Chapter 6

Evaluation, Analysis, and Comparisons

In this chapter, we discuss evaluation experiments and results for the three algorithms covered in previous chapters. The experimental study has been performed on undirected graphs using the algorithms discussed in chapter 4 and on directed graphs using the extended algorithms discussed in chapter 5. Section 6.1 provides the details of various datasets used for analysis. In section 6.2, we have discussed the details of equipments used for conducting these experiments. Finally, section 6.3 and 6.4 gives the experimental results and analysis based on comparison of directed and undirected graphs.

6.1 Datasets

We perform the evaluation of these algorithms on five real word datasets of varied sizes. All these datasets have been obtained from the Laboratory of Web Algorithms which provides the compressed form of large datasets using Layered Label Propagation [2] and WebGraph [3] . Below is a brief overview of each of the datasets:

dblp-2011 Digital Bibliography and Library Project is a popular service which provides the bibliography for published papers. The nodes in dataset *dblp-2011* represent the publishers, and an edge between two publishers implies that they have worked together. This dataset has roughly 6.5 million researchers. This is the smallest dataset used for analysis.

dewiki-2013 This graph represents the Wikipedia in German (`https://de.wikipedia.`

`org/wiki/Wikipedia:Hauptseite`). Its named dewiki as .de is the country code top-level domain for Germany. The nodes are the title of individual pages, and an edge represents reference of one article in another. *dewiki-2013* is roughly six times the size of dblp-2011 dataset.

ljournal-2008 Live Journal is a virtual social-networking site started in 1999. The node represents the users, and an edge from user a to user b represents that a registered b among his friends. *ljournal-2008* dataset has approximately 79 million edges, and it can be categorized as a medium-sized graph amongst the graphs used for experimentation.

hollywood-2009 The *hollywood-2009* graph is an interesting social-network graph of movie actors. An edge between the nodes represents that these actors have done a movie together. *hollywood-2009* dataset can be categorized as a large dataset and has 113 million edges.

arabic-2005 *Arabic-2005* is the largest dataset used for evaluation with a total of 639 million edges. This dataset has been obtained from a crawl which focused on fetching websites with pages potentially written in Arabic.

6.2 Equipments

All our experiments are conducted on Amazon Elastic Map Reduce (EMR) Service with a cluster of 1 master node and four slave nodes. Both master and the core nodes are m4.2xlarge instances with specification 16 vCore, 32 GiB memory and physical processor as Intel Xeon E5-2676 v3. We implemented all our algorithms in Java, and we used Spark for implementing map-reduce. We used the latest version of spark (Spark 2.3.0) and the Java version, 'Open JDK 1.8.0'.

6.3 Results

This section summarizes the triangles obtained for directed and undirected versions of the datasets mentioned in section 6.1. For undirected graph, we implemented the algorithms mentioned in [18] and described in detail in chapter 4 . Table 6.1 shows the summary of number of triangles captured for undirected graphs. For the largest

graph, the arabic-2005 dataset, the number of triangles retrieved are approximately 36.8 billion.

Datasets	Nodes	Edges	Triangles
arabic-2005	22,744,080	553,903,073	36,882,260,984
hollywood-2009	1,139,905	56,375,711	4,916,056,794
ljournal-2008	5,363,260	49,514,271	411,138,537
dewiki-2013	1,532,354	33,093,029	88,611,129
dblp-2011	986,324	3,353,618	7,005,235

Table 6.1: Summary of triangles in undirected graphs

For the directed graphs, we perform triangle enumeration using the extended version of algorithms mentioned in chapter 5. Table 6.2 summarizes the number of cycles and trust triangles found in directed graph datasets.

Datasets	Nodes	Edges	Trust Triangles	Cycle Triangles
arabic-2005	22,744,080	639,999,458	133,008,851,527	32,922,986,977
hollywood-2009	1,139,905	113,891,327	29,836,646,935	9,888,583,427
ljournal-2008	5,363,260	79,023,142	1,356,906,465	339,359,031
dewiki-2013	1,532,354	36,722,696	148,604,080	19,033,503
dblp-2011	986,324	6,707,236	42,031,410	14,010,470

Table 6.2: Summary of triangles in directed graphs

The number of triangles shown in table 6.1 and table 6.2 have been calculated using all three algorithms discussed, namely, PTE_{base}, PTE_{cd} and PTE_{sc}. The next section performs analysis on the results retrieved from these algorithms for both directed and undirected graphs.

6.4 Comparisons

This section lists out various experiments performed to evaluate the performance of the algorithms proposed. Section 6.4.1 compares the triangle computation time taken by PTE_{base}, PTE_{sc} and PTE_{cd} algorithms for directed and undirected graph datasets. The goal of second part of the experiment (section 6.4.2) is to find the correlation between the number of edges and computation time for PTE_{base}, PTE_{cd} and PTE_{sc}

algorithms. Finally, section 6.4.3 summarizes the experiment done to study the effect of the number of partitions on the performance of the algorithm.

6.4.1 Computation Time for different datasets

For each dataset we computed two categories of triangles for directed graph dataset, namely trust triangles and cycle triangles, and one category of triangle for undirected version of dataset, for simplicity we refer them as 'undirected' triangles. The number of triangles retrieved have been summarized in section 6.3. This section provides comparison of the computation time taken by PTE_{base}, PTE_{sc} and PTE_{cd} algorithms for these triangles enumeration. Figure 6.1, 6.2, 6.3, 6.4 and 6.5 shows results for five datasets: dblp-2011, dewiki-2013, ljournal-2008, hollywood-2009 and arabic-2005 respectively.

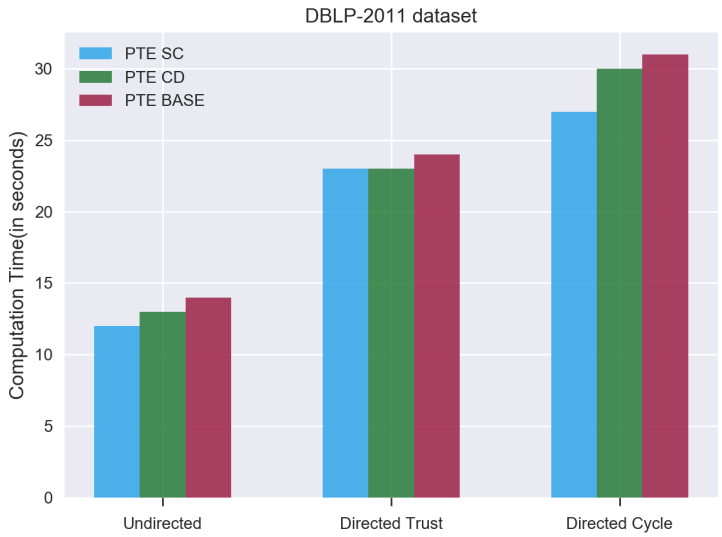


Figure 6.1: DBLP-2011
Triangle computation time in DBLP-2011 dataset

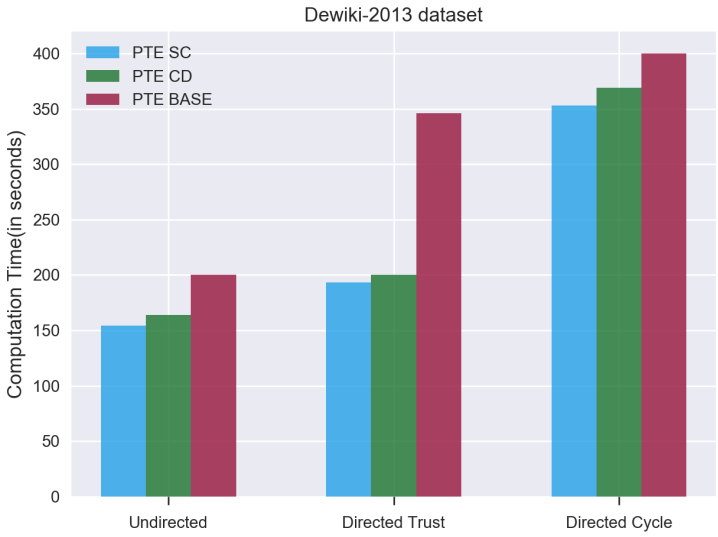


Figure 6.2: Dewiki-2013
Triangle computation time in Dewiki-2013 dataset

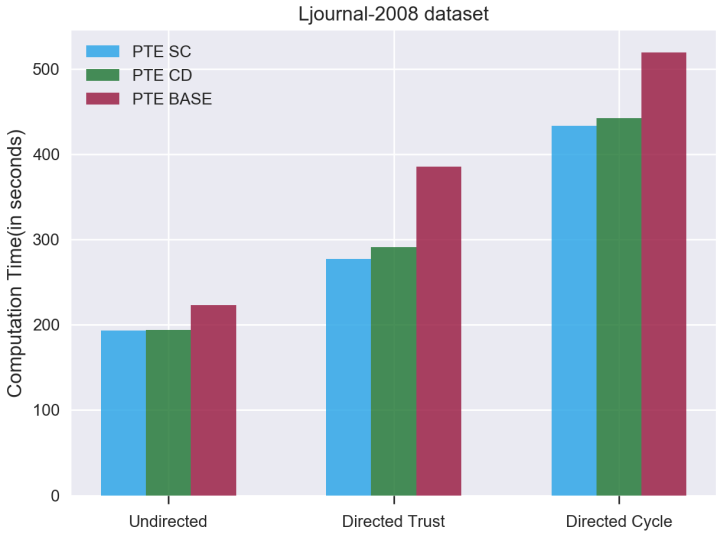


Figure 6.3: Ljournal-2008
Triangle computation time in Ljournal-2008 dataset

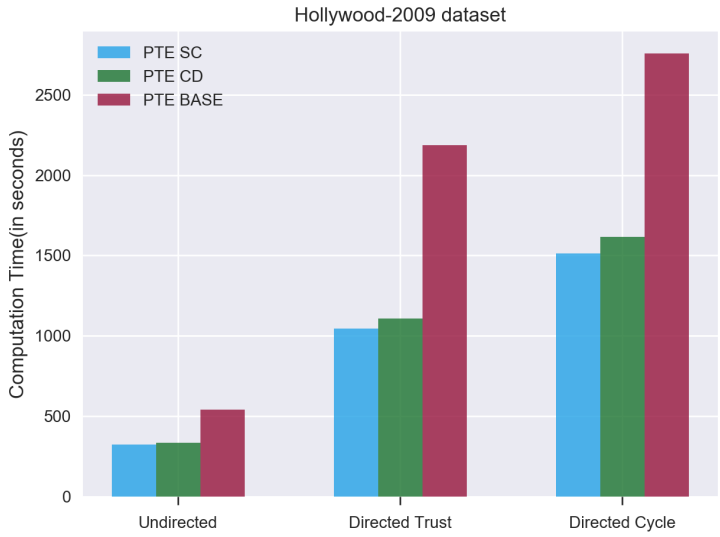


Figure 6.4: Hollywood-2009
Triangle computation time in Hollywood-2009 dataset

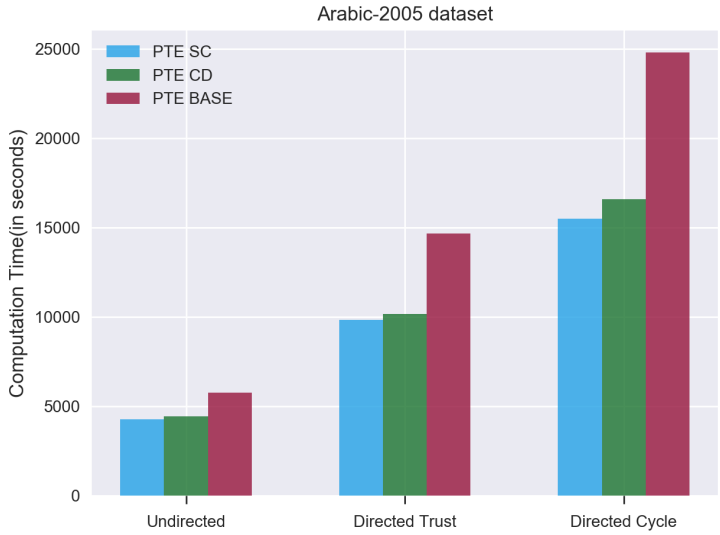


Figure 6.5: Arabic-2005
Triangle computation time in Arabic-2005 dataset

These charts show that the PTE_{sc} algorithm outperforms PTE_{cd} and PTE_{base} algorithm. This observation is in accordance with our discussion in previous chapters. Also, one interesting thing to note is that for each of these graph datasets, triangle

enumeration in undirected graphs takes much less time than in directed graphs. This is primarily because the number of edges in an undirected graph is much less than the directed graph (see table 6.1 and 6.2). A directed graph can have an edge from u to v and v to u, whereas in undirected graph we have an edge between u and v. We can also observe that computation of cycle triangles takes maximum time. This is because for cycle computation we need the transpose of the neighbor graph too for each node in each partition (refer chapter 5). There is a considerable difference between the performance of PTE_{base} and PTE_{sc} for each of these graphs. For cycle enumeration of the smallest dataset (figure 6.1), PTE_{sc} outperforms PTE_{base} by 4 seconds (PTE_{base} takes 31 seconds, whereas PTE_{sc} takes 27 seconds). Also, for the cycle enumeration of largest graph, PTE_{sc} outperforms PTE_{base} by 9,314 seconds (PTE_{base} takes 24,791 seconds, whereas PTE_{sc} takes 15,477 seconds).

6.4.2 Effect of number of edges

Here, we compare the computation time taken by PTE_{base}, PTE_{cd} and PTE_{sc} algorithms for triangle enumeration of datasets of varied edges. Figure 6.6, 6.7 and 6.8 shows the comparison of these algorithms for undirected triangles, directed cycle triangles and directed trust triangles computation respectively.

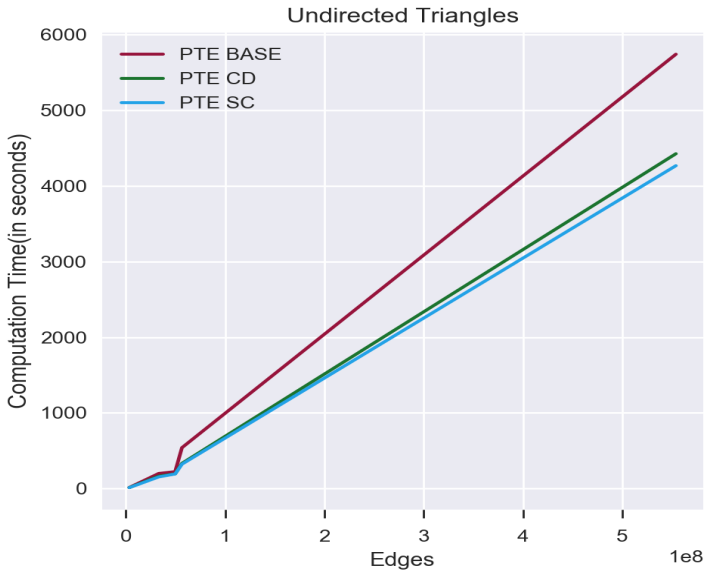


Figure 6.6: Undirected Triangles
Computation time for undirected triangles with various number of edges

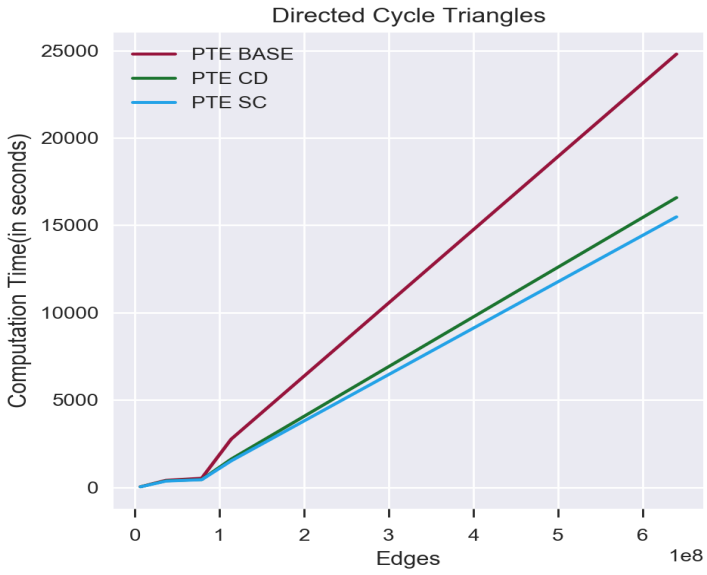


Figure 6.7: Directed Cycle Triangles
Computation time for directed cycle triangles with various number of edges

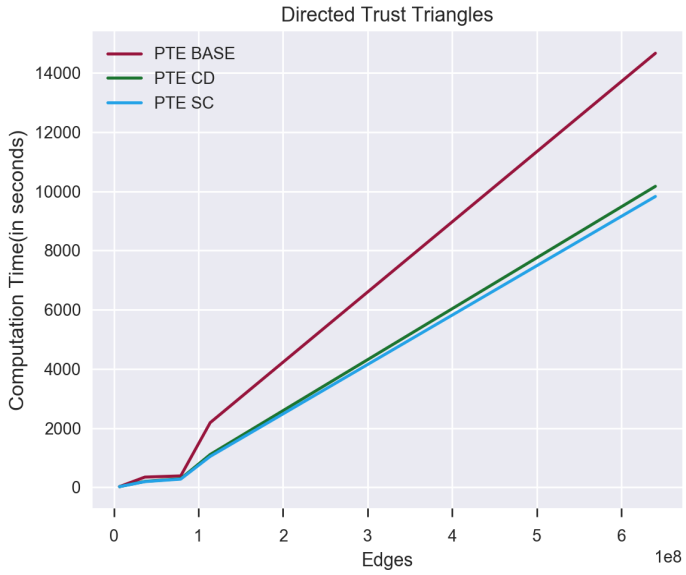


Figure 6.8: Directed Trust Triangles
Computation time for directed trust triangles with various number of edges

The above graphs show the performance of PTE_{base}, PTE_{sc} and PTE_{cd} for edges varying from 1 to 100 million. It is clear from these graphs that the performance of

these three algorithms are comparable for smaller graphs. However, with increase in the number of edges, the number of redundant operations for PTE_{base} also increases. Thus, for larger graphs with the humongous number of edges, reducing the redundant operations and network read results in a significant improvement in performance. This is clear from the above graphs as PTE_{sc} and PTE_{cd} performs much better than PTE_{base} as the number of edges increases.

6.4.3 Varying number of partitions

In this section, we show the effect of increasing the number of partitions on the performance of PTE_{sc} algorithm. For this experiment, we have used the Hollywood-2009 dataset for which the ideal number of partitions is 5 (using equation (4.4)).

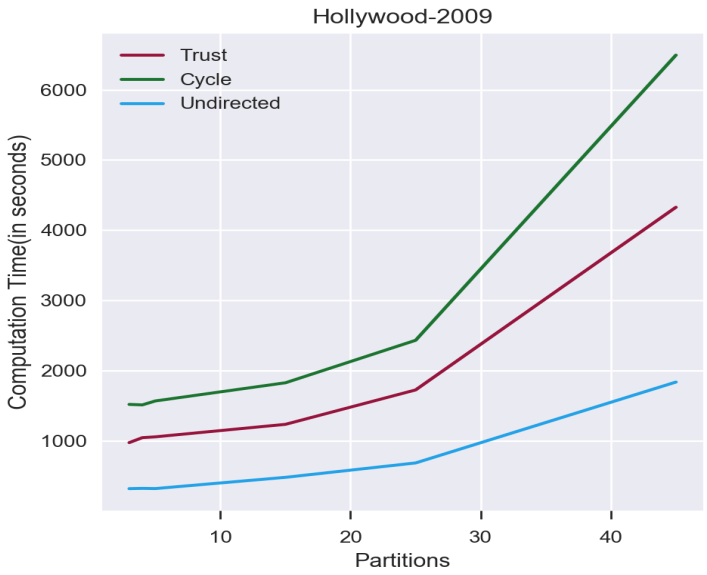


Figure 6.9: Hollywood-2009
Computation time for Undirected, Trust and Cycle enumeration with varied number of partitions

Figure 6.9 shows the effect of increasing the number of partitions on the performance of PTE_{sc} algorithm for trust and cycle computation in the directed Hollywood-2009 dataset and undirected triangle computation for the undirected Hollywood-2009 dataset. All the three lines in the graph start from partition value 3. This is because the program fails with out of memory error for the number of partitions smaller than

that. For such cases, the size of sub-graph created is too big to fit in the memory. Thus, having a feasible number of partitions is important. For partition values approximately close to the ideal value of partitions (5 in this case), the computation time is approximately similar. However, with the increase in the number of partitions, the graph gets divided into many more sub-graphs hence increasing the computation time exponentially.

Chapter 7

Conclusions

This project focuses on three existing fast map-reduce algorithms: PTE_{base}, PTE_{cd} and PTE_{sc} [18], for listing triangles in very large undirected graphs. The primary focus of each of these algorithms are as follows:

1. PTE_{base} targets the reduction of shuffle data generated during sub-graph creation by previous algorithms proposed in this field.

2. PTE_{sc} addresses the issue of redundant operations performed and thereby reducing the computation time.

3. PTE_{cd} aims at minimizing the amount of network read done during the triangle enumeration.

Then we propose an extension of all three of these algorithms for trust-triangle and cycle-triangle computation in directed graphs. The information of trust-triangles and cycle-triangles present in a directed graph have varied applications, the presence of trust-triangles in a graph can help us understand the degree of transitivity and homophily present in a directed graph, whereas, the existence of cycle-triangles can help us to deduce any deadlocks in a system.

Finally, we perform an evaluation of PTE_{base}, PTE_{cd} and PTE_{sc} algorithms for enumeration of all three types of triangles, namely undirected triangles, directed-trusts and directed-cycles on five real-life datasets. The biggest directed graph dataset used, arabic-2005 has about 22.5 million vertexes and 639 million edges. Our results show that the arabic-2005 directed graph dataset has approximately 133 billion trust-triangles and 33 million cycle-triangles.

We observe that PTE_{sc} algorithm performs better than PTE_{base} and PTE_{cd} for enumeration of both directed and undirected triangles. We are able to compute the directed triangles in these real-life datasets within reasonable amount of time. Also, the computation of directed trust-triangles takes slightly more time than the computation of undirected triangles. This is primarily because the number of edges in directed and undirected graphs is different. And hence, there is a huge difference in the number of directed-trust and undirected triangles. For instance, for arabic-2005, the number of edges for undirected graph is approx 554 million and number of undirected triangles found is roughly 36 billion. Likewise, for directed arabic-2005 graph dataset, number of edges is roughly 640 million, which has approx 133 billion directed-trust triangles (3.7 times the number of undirected triangles). Hence, there is a slight difference in the computation time.

Moreover, there is a significant difference between the computation time of directed-cycle triangles and the undirected triangles. This happens because for cycle enumeration we need the transposed sub-graph as well as the original graph for each partition. But, considering the complexity of cycle computation, PTE algorithms take significantly less time for even large graphs such as arabic-2005.

This project lays the ground for future work in several different directions. We have implemented PTE algorithms for two types of directed triangles. However, in [22], Seshadhri et al. explain seven possible types of directed triangles. Each of these types have their unique application. It will be great to extend PTE algorithm to perform enumeration of all six types of directed triangles for huge graphs. Furthermore, with the ever-growing sizes of the graphs, the evaluation of the algorithms can be extended to much more massive datasets spread across various domains.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[2] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[3] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[4] Shigang Chen, Yi Deng, Paul C. Attie, and Wei Sun. Optimal deadlock detection in distributed systems based on locally constructed wait-for graphs - distributed computing systems, 1996., proceedings of the 16th international conference on. 2004.

[5] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] Ernesto Estrada and Michele Benzi. Are social networks really balanced? *CoRR*, abs/1406.2132, 2014.

[8] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[9] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 1–10, New York, NY, USA, 1977. ACM.

[10] Gabriela Kalna and Desmond J. Higham. A clustering coefficient for weighted networks, with application to gene expression data. *AI Commun.*, 20(4):263–271, 2007.

[11] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. *ACM SIGCOMM Computer Communication Review*, 30(4):97–110, 2000.

[12] Ralf Lämmel. Googles mapreduce programming modelrevisited. *Science of computer programming*, 70(1):1–30, 2008.

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

[14] MEJ Newman and J. Park. Why social networks are different from other types of networks. *Physical Review E*, 68(3):36122, 2003.

[15] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155–163, 2009.

[16] Rasmus Pagh and Charalampos E Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

[17] Ha-Myung Park and Chin-Wan Chung. An efficient mapreduce algorithm for counting triangles in a very large graph. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 539–548. ACM, 2013.

[18] 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.

[19] 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.

[20] Thomas Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, Universität Karlsruhe, 2007.

[21] Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, volume 3503 of *Lecture Notes in Computer Science*, pages 606–609. Springer, 2005.

[22] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, volume 4, page 5, 2013.

[23] 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.

[24] Benjamin M. Tabak, Marcelo Takami, Jadson M.C. Rocha, Daniel O. Cajueiro, and Sergio R.S. Souza. Directed clustering coefficient as a measure of systemic risk in complex banking networks. *Physica A: Statistical Mechanics and its Applications*, 394:211 – 216, 2014.

[25] Taro Takaguchi and Yuichi Yoshida. Cycle and flow trusses in directed networks. *CoRR*, abs/1603.03519, 2016.

[26] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 837–846, 2009.

[27] Charalampos E Tsourakakis, Mihail N Kolountzakis, and Gary L Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.

[28] Sentinel Visualizer. *Social Network Analysis (SNA) Diagrams*, (accessed April 9, 2018).

[29] Stanley Wasserman and Katherine Faust. *Social Network Analysis. Methods and Applications.* 1994.

[30] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[31] Dennis M Wilkinson and Bernardo A Huberman. A method for finding communities of related genes. *proceedings of the national Academy of sciences*, 101(suppl 1):5241–5248, 2004.

[32] Andreas Wimmer and Kevin Lewis. Beyond and below racial homophily: Erg models of a friendship network documented on facebook. *American Journal of Sociology*, 116(2):583–642, 2010.

[33] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. Uncovering social network sybils in the wild. *TKDD*, 8(1):2:1–2:29, 2014.

[34] 2018 Digital Yearbook. *The statistics portal*, (accessed April 9, 2018).

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