An Efficient MapReduce Algorithm for Counting Triangles in a Very Large Graph

Ha-Myung Park
Division of Web Science and Technology, KAIST
291 Daehak-ro, Yuseoung-gu
Daejeon, Korea
hmpark@islab.kaist.ac.kr

Chin-Wan Chung
Division of Web Science and Technology, KAIST
Department of Computer Science, KAIST
291 Daehak-ro, Yuseoung-gu
Daejeon, Korea
chungcw@kaist.edu

ABSTRACT

Triangle counting problem is one of the fundamental problem in various domains. The problem can be utilized for computation of clustering coefficient, transitivity, trianglular connectivity, trusses, etc. The problem have been extensively studied in internal memory but the algorithms are not scalable for enormous graphs. In recent years, the MapReduce has emerged as a de facto standard framework for processing large data through parallel computing. A MapReduce algorithm was proposed for the problem based on graph partitioning. However, the algorithm redundantly generates a large number of intermediate data that cause network overload and prolong the processing time. In this paper, we propose a new algorithm based on graph partitioning with a novel idea of triangle classification to count the number of triangles in a graph. The algorithm substantially reduces the duplication by classifying triangles into three types and processing each triangle differently according to its type. In the experiments, we compare the proposed algorithm with recent existing algorithms using both synthetic datasets and real-world datasets that are composed of millions of nodes and billions of edges. The proposed algorithm outperforms other algorithms in most cases. Especially, for a twitter dataset, the proposed algorithm is more than twice as fast as existing MapReduce algorithms. Moreover, the performance gap increases as the graph becomes larger and denser.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process

Keywords

Graph; triangle; MapReduce

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505563.

1. INTRODUCTION

With the rapid growth of the Internet technologies and the social network services, the field of graph analysis is receiving significant attention from both academia and industry. Graphs are used extensively to model numerous types of networks, such as social networks, peer-to-peer networks, and the World Wide Web. In many cases, the graphs for these networks can be enormous because these networks usually involve numerous relationships among numerous entities. For example, in the popular social network Facebook, there are more than 1.1 billion users and over 69 billion friendships[1, 24]. Therefore, it is essential to extract only the useful information from a graph, but it is not trivial to analyze an enormous graph to determine patterns and trends within the graph.

Counting the number of triangles in a graph is an important function in graph analysis. In the field of graph analysis, the clustering coefficient[25] and transitivity ratio[20], which are highly relevant to the number of triangles, are regarded as important measures that quantify the degree of clustering in a graph. These measures have been used for various applications such as detecting fake accounts in social networks[27], finding malicious pages on the Web[6], uncovering hidden thematic layers on the Web[13], and detecting communities[7].

The problem of counting the number of triangles has been studied extensively over the past few years due to the numerous applications related to the problem. Thus far, the research has focused primarily on internal memory[15, 17, 21]; however, as a result of the enormity of recent graphs, it is effectively impossible for the internal memory algorithms to count the number of triangles in the graphs. One of the methods to manage such enormous data is to exploit the parallel computing paradigm. MapReduce[11], and its open source version Hadoop[26], has emerged as a de facto standard framework for processing large data through parallel computing. A number of researchers have employed MapReduce to enable their algorithms to be scalable[16, 23, 9, 19].

Suri and Vassilvitskii[22] proposed a MapReduce algorithm, called *Graph Partition*(GP) algorithm, and it uses a graph partitioning technique to count the number of triangles in enormous graphs. The GP algorithm partitions a graph into overlapping subgraphs in order that every triangle in the graph is present in at least one subgraph. Then, it counts the number of triangles in each subgraph in parallel. However, the GP algorithm generates a large number of in-

termediate data that cause network overloads and increase the processing time.

In this paper, we propose a new MapReduce algorithm using a graph partitioning technique to count the number of triangles in an enormous graph. When a graph is partitioned, every triangle can be classified into one of three types according to the number of subgraphs in which the nodes of the triangle are placed. In the GP algorithm, if two or three nodes of a triangle are in the same subgraph, the triangle is calculated redundantly. In particular, when all nodes in a triangle are included in the same subgraph, the triangle is calculated in proportion to the square number of subgraphs.

Our algorithm substantially reduces the number of calculations by using different processes for each type of triangle and partitioning a graph in a different manner to that of the GP algorithm; thus, our algorithm computes far fewer triangles during the process than the GP algorithm.

Contributions. Our significant contributions are summarized as follows.

- We propose an efficient MapReduce algorithm to count the number of triangles in an enormous graph that will not fit in internal memory.
- We also prove the correctness and efficiency of the proposed algorithm; the algorithm operates in $O(m^{\frac{3}{2}})$ time in which the best-known algorithm operates.
- Our algorithm is experimentally evaluated using both real-world datasets and synthetic datasets. The experimental results demonstrate that our algorithm outperforms existing MapReduce algorithms in counting the number of triangles. In particular, for a Twitter dataset, our algorithm is more than twice as fast as the existing algorithms. Moreover, the results demonstrate that the performance gap increases as the graph becomes larger and denser.

Paper Organization. In Section 2, previous works relevant to the problem of counting the number of triangles are reviewed. The problem of counting the number of triangles and a set of frequent notations are formally defined in Section 3. In Section 4, we describe the proposed algorithm and proves its efficiency with theoretical bounds. In Section 5, we give the experimental evaluation and conclusions are made in Section 6.

2. RELATED WORKS

In this section, we first provide an overview of the triangle counting algorithms for internal and external memory. Then, we outline the flow of the MapReduce because our algorithm is a MapReduce algorithm. Finally, we introduce the existing MapReduce algorithms for counting the number of triangles. The notations that are used are listed in Table 1.

2.1 Internal memory algorithms

The triangle counting problem has been studied extensively for internal memory algorithms. Here, we provide an overview of the existing internal memory algorithms.

Table 1: The notation used in the paper

Notation	Explanation
$G = (V, E)$	Input graph
(u, v)	edge between u, v
n	The number of nodes in V
m	The number of edges in E
$\Delta(u, v, w)$	Triangle; set of three nodes such that (u, v), (u, w), $(v, w) \in E$
$N(v)$	Neighbors of a node v
$d(v)$	degree of a node v
ρ	The number of partitions of G
$G_i = (V_i, E_i)$	Partition; induced subgraph of G on V_i
$G_{ij} = (V_{ij}, E_{ij})$	Induced subgraph of G on $V_i \cup V_j$ where $i \neq j$
$G_{ijk} = (V_{ijk}, E_{ijk})$	Induced subgraph of G on $V_i \cup V_j \cup V_k$ where $i \neq j \neq k$

The simplest method of counting the number of triangles is to investigate all possible triples of nodes whether the nodes in a triple are fully connected to each other. There are two foundational algorithms for this: the node-iterator algorithm and the edge-iterator algorithm. For each node n in V, the node-iterator algorithm counts every pair that has an edge between them in $N(n)$. For each edge (n, v) in E, the edge-iterator algorithm computes the intersection of $N(n)$ and $N(v)$, and then counts the number of nodes in the intersection. These two algorithms have the same upper bound $O(d_{max}^2 n)$ where d_{max} is the maximum degree of nodes [21]. Moreover, the two fundamental algorithms output each triangle three times.

The forward algorithm eliminates the output duplication and improves the edge-iterator algorithm by ordering the nodes. It returns a triangle $\Delta(u, v, w)$ if and only if $u \prec v \prec w$ where $\prec$ is a total order on all of the nodes. Usually, the nodes are ordered by the degree of nodes. The compact-forward algorithm is a refined version of the forward algorithm. It reduces the usage of space from $\Theta(3m + 3n)$ to $\Theta(2m + 2n)$ [17].

There is another approach for the triangle counting problem that utilizes a matrix multiplication. If A is the adjacency matrix representation of input graph G, the number of triangles is computed simply by calculating A^3. The AYZ algorithm [4] uses the matrix multiplication. This algorithm is known as the fastest algorithm for counting triangles: its running time is $O(m^{1.41})$ with the fast matrix multiplication [10]. However, the AYZ algorithm uses $O(n^2)$ space, which is a prohibitive space cost for computing enormous graphs.

2.2 External memory algorithms

This section briefly introduces the triangle counting algorithms for external memory. Dementiev [12] proposed an external version of the node-iterator algorithm, called the external node-iterator algorithm. The node-iterator algorithm for internal memory uses an adjacency matrix representation of a graph in order to test the adjacency of two nodes in a constant time. In order to avoid the adjacency matrix representation, the external node-iterator algorithm uses a sorted edge list to verify the adjacency of two nodes. Furthermore, it reduces the number of computations by employing degree ordering. Menegola [18] proposed an external version

of the compact-forward algorithm, called external compact-forward algorithm. In the compact-forward algorithm for internal memory, the node processing order should be respected. The external compact-forward algorithm guarantees the processing order using a mapped index containing the order of each node. Chu and Cheng [8] proposed graph partitioning algorithms for the external memory. These external graph partitioning algorithms have a common base framework and vary according to the partitioning methods: sequential partitioning, dominating set based partitioning, and randomized partitioning. The external graph partitioning algorithm framework partitions the graph and classifies triangles. The concept of this external graph partitioning algorithm is similar to our algorithm, but the two algorithms have significant differences. In the external graph partitioning algorithm, each partition is extended to include all neighboring nodes in the partition. By doing so, every triangle is listed without exception. However, if a node has many neighbors, a partition including the node cannot fit in the memory. In contrast, our algorithm does not extend partitions so that every partition remains within the memory size. Our algorithm guarantees that every triangle is in at least one partition using a different method; moreover, our algorithm is a MapReduce algorithm not an external memory algorithm. Hu et al. [14] proposed a state-of-the-art algorithm for counting triangles in the external memory. This algorithm loads certain size edges in the memory and finds all triangles containing one of these edges by traversing every node.

External memory algorithms are devised in a different environment, in which massive parallel and distributed computing is not available, so they are not in the scope of this paper.

2.3 MapReduce

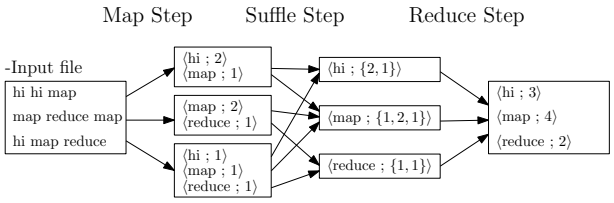


Figure 1: MapReduce example, word counting

MapReduce is a programming framework for processing large data using parallel computation. The name and concept of the MapReduce are from the 'Map' and 'Reduce' operations in functional programming languages. With the MapReduce framework, programmers can easily utilize distributed machines in order to process large data in parallel by writing scripts for the Map and Reduce function. There are three steps in the MapReduce framework: Map, Shuffle, and Reduce.

Map Step. A Map function that is scripted by a programmer is copied to each machine and each copied Map function is called a Map instance. In the Map step, each Map instance receives a line as input from a file and outputs a number of $\langle key; value\rangle$ pairs in response to the input. The number of output pairs may be zero.

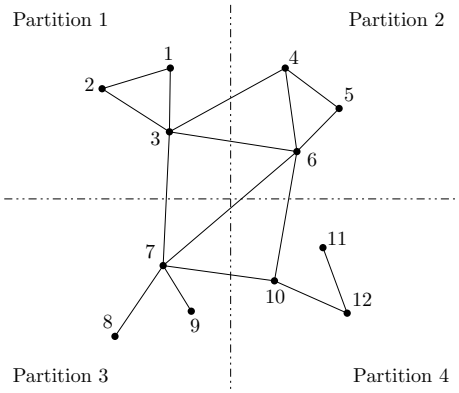


Figure 2: the input graph and partitions in our running example

Shuffle Step. The output pairs from the Map step are bound by keys in the Shuffle step. That is, pairs that have the same key are combined as $\langle key, \{value1, value2, ...\}\rangle$.

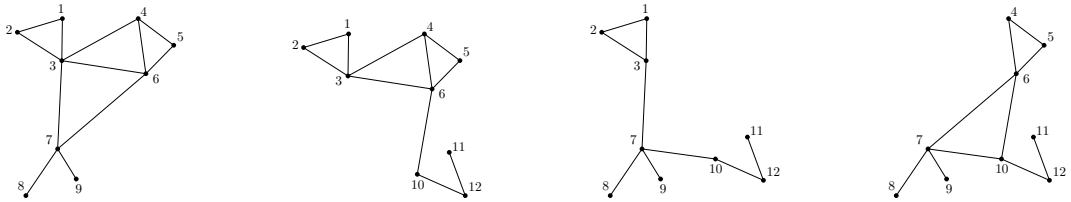
Reduce Step. A Reduce function that is scripted by a programmer is copied to each machine and each copied Reduce function is called a Reduce instance. Each Reduce instance receives one of the output pairs $\langle key, valueSet\rangle$ from the Shuffle step as input and outputs a number of $\langle key; value\rangle$ pairs.

Figure 1 shows a word counting example.

2.4 Existing MapReduce algorithms

Recently, two MapReduce algorithms used to count the number of triangles were proposed. Cohen[9] proposed the first MapReduce algorithm, called the Cohen algorithm, to enumerate triangles. This algorithm is based on the node-iterator algorithm. As mentioned above, the node-iterator algorithm outputs triangles redundantly. In order to avoid the duplication, the algorithm applies an ordering technique; any ordering factor can be applied, e.g. numerical ordering of node numbers. The author chose the degree of nodes as the ordering factor. Using the degree ordering, the processing time of the algorithm is $O(m^{\frac{3}{2}})$, which is the optimal bound for enumerating triangles. However, the algorithm diminished as the average degree of nodes increases. Moreover, if a node has a very high degree, a Reduce instance whose input key contains the node cannot accommodate the values related to the key.

Suri and Vassilvitskii[22] proposed a MapReduce algorithm for the triangle counting problem, which is the current state-of-the-art MapReduce algorithm. It resolves the memory capacity problem of the Cohen algorithm through the use of a graph partitioning technique. Suri and Vassilvitskii's algorithm can modulate the input sizes of the Reduce instances. However, the algorithm generates numerous duplicate edges during the process, which increase the network overload, and calculates triangles redundantly. We review the algorithm in detail in Section 4 because the algorithm is closely related to our algorithm.



(a) G_{123} (b) G_{124} (c) G_{134} (d) G_{234}

Figure 3: 3-partitions of the graph in Figure 2

3. PROBLEM DEFINITION

Let us consider a simple undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. Let n and m be the size of nodes $|V|$ and edges $|E|$, respectively. An edge between two nodes u and v is represented as (u, v). Let $N(v)$ be the set of directly adjacent nodes(neighbors) of a node v in G, and $d(v)$ be the degree of v, i.e. $d(v) = |N(v)|$. A triangle, denoted by $\Delta(u, v, w)$, is a set of three nodes $\{u, v, w\} \subseteq V$ such that $(u, v), (u, w), (v, w) \in E$. Then, the triangle counting problem is formally defined as follows:

Definition 1. (Triangle counting) Given a graph G, the triangle counting problem is to count the number of all triangles in G.

We also define the following important terms that are used to describe our algorithm throughout the paper. Note that the term partition is used informally as a subgraph instead of following the standard mathematical definition in order to be consistent to usual terms for graphs in the MapReduce area.

Definition 2. (Partition or 1-partition) Given an integer ρ, a *partition (or 1-partition)*, denoted by $G_i = (V_i, E_i)$ or $G_j = (V_j, E_j)$, is a subgraph of G such that $V_i \cap V_j = \emptyset$ where $0 < i < j \leq \rho$, and $\bigcup_{i=1}^{\rho} V_i = V$ where $0 < i \leq \rho$. Then, ρ is the number of partitions.

Definition 3. (2-partition and 3-partition) Let G_i, G_j and G_k be three different partitions complying with Definition 2. Then a 2-partition, denoted by $G_{ij} = (V_{ij}, E_{ij})$, is an induced subgraph of G on $V_i \cup V_j$. Similarly, a 3-partition, denoted by $G_{ijk} = (V_{ijk}, E_{ijk})$, is an induced subgraph of G on $V_i \cup V_j \cup V_k$.

When an input graph is divided into ρ partitions, the number of all possible 2-partitions and 3-partitions are $\binom{\rho}{2}$ and $\binom{\rho}{3}$ respectively. Figure 2 shows an input graph and partitions. There are four partitions(G_1, G_2, G_3 and G_4), six 2-partitions(G_{12}, G_{13}, G_{14}, G_{23}, G_{24} and G_{34}), and four 3-partitions(G_{123}, G_{124}, G_{134} and G_{234}). G_1 is composed of three nodes 1, 2 and 3, and three edges $(1, 2)$, $(1, 3)$, and $(2, 3)$, i.e. $G_1 = (\{1, 2, 3\}, \{(1, 2), (1, 3), (2, 3)\})$. Similarly, G_2, G_3 and G_4 are induced subgraphs of the input graph on $\{4, 5, 6\}$, $\{7, 8, 9\}$ and $\{10, 11, 12\}$ respectively. G_{12} is a 2-partition which is composed of six nodes $\{1, 2, 3, 4, 5, 6\}$ and eight edges $\{(1, 2), (1, 3), (2, 3), (3, 4), (3, 6), (4, 5), (4, 6), (5, 6)\}$. G_{123} is a 3-partition which is composed of nine nodes $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and twelve edges $\{(1, 2), (1, 3), (2, 3), (3, 4), (3, 6), (3, 7), (4, 5), (4, 6), (5, 6), (6, 7), (7, 8), (7, 9)\}$.

4. MAPREDUCE ALGORITHM FOR TRIANGLE COUNTING

In this section, we propose a novel MapReduce algorithm to count the number of triangles in a graph. It is essentially a graph partitioning algorithm that divides a graph into multiple overlapping subgraphs that together cover all triangles in the graph. Suri and Vassilvitskii [22] also proposed a partitioning algorithm for counting triangles; however, their algorithm generates numerous duplicated edges during the process and performs redundant computations when counting triangles. First, we introduce the previous GP algorithm [22]; then, we propose our algorithm and analyze its efficiency.

4.1 Graph Partition Algorithm

Suri and Vassilvitskii [22] proposed a MapReduce algorithm, called *graph partition* (GP) algorithm, to count the number of triangles in a graph. The first step of the algorithm is to partition the nodes of a graph into ρ partitions, $G_i = (V_i, E_i)$ where $0 < i \leq \rho$, so that each partition has almost the same number of nodes. The algorithm counts the number of triangles using an internal memory algorithm for each 3-partitions G_{ijk} where $0 < i < j < k \leq \rho$.

Certain triangles appear multiple times during the process, which produces redundant results. Consider a triangle $\Delta(u, v, w)$ where u, v, and w are included in the same partition G_x. The triangle $\Delta(u, v, w)$ is present in every 3-partition G_{ijk} where $x = i$ or $x = j$ or $x = k$. For example, in Figure 3, the triangle $\Delta(1, 2, 3)$ whose nodes are in the same partition G_1 appears three times in G_{123}, G_{124} and G_{134}. Similarly, the triangle $\Delta(4, 5, 6)$ also appears three times in G_{123}, G_{124} and G_{234}. For two nodes of a triangle in the same partition, e.g. u and v are in G_x and w is in G_y, the triangle appears in every 3-partition G_{ijk} where $\{x, y\} \in \{i, j, k\}$. For example, in Figure 3, the triangle $\Delta(3, 4, 6)$ whose two nodes 4 and 6 are in the same partition G_2 appears twice in G_{123} and G_{124}. In order to correctly count such triangles, the algorithm corrects the duplicated triangles by reporting each triangle as 1/(number of the duplicates). For example, in Figure 3, the triangle $\Delta(1, 2, 3)$ is reported as 1/3. Since the triangle $\Delta(1, 2, 3)$ is reported three times, the sum of the reported numbers for the triangle is 1.

The above procedure is realized in a MapReduce algorithm by implementing a Map function and a Reduce function. The Map function takes an edge $(u, v) \in E$ as an input and outputs $\langle (i, j, k), (u, v) \rangle$ pairs such that u and v are included in G_{ijk}. The Reduce function counts the triangles using an internal memory algorithm and corrects the number of triangles.

The primary drawback of the GP algorithm is the redundant computation of triangles. In particular, triangles that are included wholly in a partition are computed in $O(\rho^2)$ time. Our observation is that the primary cause of the redundant computation of triangles is the duplicate edges output from the Map function. The output edge size is also $O(\rho^2)$ for an edge (u, v) if the u and v are in a partition. These edges lead to network overload and delay the Shuffle step of MapReduce.

4.2 Triangle Type Partition Algorithm

We propose a new MapReduce algorithm, named the *Triangle Type Partition* (TTP). The algorithm substantially rectifies the duplication problem in the GP algorithm.

When a graph is partitioned, a triangle can be classified into one of three types according to the number of partitions in which its nodes are positioned.

Definition 4. (Types of Triangle) The three types of triangles are defined as follows:

Type-1. If the three nodes u, v, and w of a triangle $\Delta(u, v, w)$ are placed in the same partition, the triangle type is Type-1.
Type-2. If the two nodes of a triangle belong to one partition and the remaining node belongs to a different partition, the triangle type is Type-2.
Type-3. If the three nodes are in different partitions, the triangle type is Type-3.

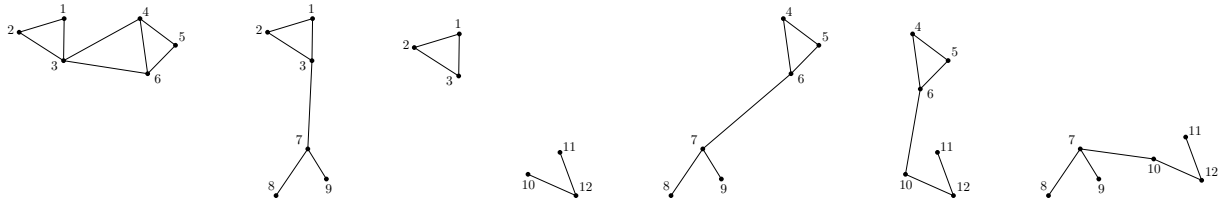
For example, in Figure 2, the triangles $\Delta(1, 2, 3)$ and $\Delta(4, 5, 6)$ are Type-1 triangles. The triangle $\Delta(3, 4, 6)$ is a Type-2 triangle. The triangles $\Delta(3, 6, 7)$ and $\Delta(6, 7, 10)$ are Type-3 triangles.

Let an edge (u, v) be an *inner-edge* if the nodes u and v belong to the same partition, and the edge be an *outer-edge* if the nodes u and v are in different partitions. For example, in Figure 2, the edges $(1, 2)$, $(1, 3)$, $(2, 3)$, $(4, 5)$, $(4, 6)$, $(5, 6)$, $(7, 8)$, $(7, 9)$, $(10, 12)$ and, $(11, 12)$ are inner-edges and the others are outer-edges. As mentioned above, the GP algorithm redundantly processes Type-1 and Type-2 triangles. In order to reduce these duplications, the TTP algorithm utilizes the fact that Type-3 triangles are only composed of outer-edges, i.e. there are no inner-edges in Type-3 triangle. This suggests that Type-3 triangles can be counted correctly without inner-edges. Then, our strategy is to dissociate the Type-1 and Type-2 triangles from the Type-3 triangles. A 3'-partition, denoted by G'_{ijk}, is a subgraph of a 3-partition without inner-edges. In Figure 5, there are examples of G'_{ijk} for the graph in Figure 2. The TTP algorithm counts Type-1 and Type-2 triangles in 2-partitions and Type-3 in 3'-partitions. Type-2 triangles are counted in 2-partitions because every Type-2 triangle is present in one of the 2-partitions; the TTP algorithm also counts Type-1 triangles in the 2-partitions because every Type-1 triangle also appears in the 2-partitions. In Figure 4, the Type-2 triangle $\Delta(3, 4, 6)$ appears only in G_{12} and The Type-1 triangle $\Delta(1, 2, 3)$ also appears in G_{12}, G_{13}, and G_{14}. The algorithm does not count Type-1 triangles in 1-partitions because there is no advantage by doing so. That is, the algorithm should compute triangles in 2-partition even when Type-1 triangles are already counted; thus, Type-1 and Type-2 triangles are processed at the same time.

Algorithm 1: Triangle Type Partition algorithm

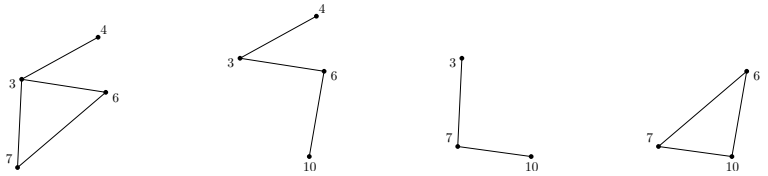
```
   Map      : input ⟨φ ; (u, v)⟩
1  for a ∈ [0, ρ − 2] do
2      for b ∈ [a + 1, ρ − 1] do
3          if {P(u), P(v)} ⊆ {a, b} then
4              emit ⟨(a, b) ; (u, v)⟩;
5  if P(u) ≠ P(v) then
6      for a ∈ [0, ρ − 3] do
7          for b ∈ [a + 1, ρ − 2] do
8              for c ∈ [b + 1, ρ − 1] do
9                  if {P(u), P(v)} ⊆ {a, b, c} then
10                     emit ⟨(a, b, c) ; (u, v)⟩;

   Reduce   : input ⟨(i, j, k) ; E_ijk⟩ or ⟨(i, j) ; E_ij⟩
11 E_t ← E_ijk or E_ij;
12 Δ ← Find all triangles on G_t;
13 foreach (u, v, w) ∈ Δ do
14     if P(u) = P(v) = P(w) then
15         emit ⟨(u, v, w) ; 1/(ρ−1)⟩;
16     else
17         emit ⟨(u, v, w) ; 1⟩;
```

The pseudo code for the TTP algorithm is shown in Algorithm 1. The algorithm is composed of a pair of Map and Reduce functions. Each edge of input graph is sent to one of map instances. The Map instance takes an edge (u, v) as input. Let $P(v)$ be a hash function that returns an integer within $[0, \rho - 1]$ such that the integer corresponds to a partition. For example, in Figure 2, $P(1) = 1$, $P(3) = 1$, $P(4) = 2$, $P(7) = 3$, $P(11) = 4$ and so on. If the input edge is an inner-edge, the Map instance outputs pairs $\langle (a, b)$; $(u, v)\rangle$ for each $a, b \in [0, \rho - 1]$ satisfying $\{P(u), P(v)\} \subseteq \{a, b\}$ (Lines 1-4). This means that the algorithm includes the edge (u, v) in G_{ab} satisfying such a condition. Let us consider an edge $(1, 2)$ in the running example (Figure 2). If the edge is the input of a Map instance, the Map instance will output pairs $\langle (1, 2)$; $(1, 2)\rangle$, $\langle (1, 3)$; $(1, 2)\rangle$ and $\langle (1, 4)$; $(1, 2)\rangle$. If the input edge is an outer-edge, the Map instance outputs pairs in common with an inner-edge. In addition, It also outputs pairs $\langle (a, b, c)$; $(u, v)\rangle$ for each $a, b, c \in [0, \rho - 1]$ satisfying $\{P(u), P(v)\} \subseteq \{a, b, c\}$ (Lines 5-10). For example, if the outer-edge $(3, 4)$ in Figure 2 is the input of a Map instance, the Map instance will output pairs $\langle (1, 2)$; $(3, 4)\rangle$, $\langle (1, 2, 3)$; $(3, 4)\rangle$, and $\langle (1, 2, 4)$; $(3, 4)\rangle$. After all Map instances finish their tasks, the MapReduce framework gathers values that share the same key. Then it assigns the values with the key to a reduce instance. In this algorithm, each key corresponds to a 2-partition or a 3'-partition. For example, if a pair $\langle (2, 3, 4)$; $\{(6, 7), (6, 10), (7, 10)\}\rangle$ is the input of a Reduce instance, it implies that G'_{234} contains edges $(6, 7)$, $(6, 10)$ and $(7, 10)$, and the Reduce instance receives G'_{234}. In summary, each Reduce instnace takes a 2-partition or a 3'-partition as input and counts the number of triangles in the 2-partitions or 3'-partitions. Note that any internal memory algorithm for counting the number of triangles can be used in the Reduce function.



(a) G_{12} (b) G_{13} (c) G_{14} (d) G_{23} (e) G_{24} (f) G_{34}

Figure 4: 2-partitions of the graph in Figure 2



(a) G'_{123} (b) G'_{124} (c) G'_{134} (d) G'_{234}

Figure 5: 3-partitions without inner-edges of the graph in Figure 2

4.3 Analysis

In this section, we analyze the network/space usage and the processing time of the proposed algorithm. We first show the correctness of the algorithm.

Lemma 1. All triangles in a graph are counted correctly by the TTP algorithm.

PROOF. Each Type-3 triangle $\Delta(u, v, w)$ appears only in a 3'-partition G'_{ijk} where $u \in G_i$, $v \in G_j$ and $w \in G_k$; thus, Type-3 triangles are counted correctly. Without loss of generality, on behalf of all Type-2 triangles, let us consider a triangle $\Delta(u, v, w)$ where u and v are in the same partition and w is in another partition. The triangle appears only in a 2-partition G_{ij} where $u, v \in G_i$ and $w \in G_j$ and does not appear in any 3'-partitions because the triangle has an inner-edge but there are no inner-edges in every 3'-partition; thus, Type-2 triangles are counted correctly. Each Type-1 triangle $\Delta(u, v, w)$ whose u, v and w belong to a partition G_i appears in 2-partitions G_{ij} where $j \in [0, \rho-1]$ and $i \neq j$. Then, the number of G_{ij} is $\rho - 1$; it implies that each Type-1 triangle appears $\rho - 1$ times. The algorithm counts each Type-1 triangle as $\frac{1}{\rho-1}$; thus, Type-1 triangles are counted correctly. □

We analyze the network and space usage of the TTP algorithm.

Lemma 2. The expected size of inner-edges is $\frac{m}{\rho}$ and the size of outer-edges is $\frac{(1-\rho)m}{\rho}$. Then, the expected size of output from all Map instances is $m(\rho - 1) = O(m\rho)$.

PROOF. If an input edge for a Map instance is an inner-edge in G_i, it is output to G_{ij} for each $j \in [0, \rho - 1]$ and $i \neq j$. It implies that every inner-edge is output $\rho - 1$ times. Let us consider an inner-edge (u, v). The probability for two nodes u and v to belong to the same partition is $\frac{1}{\rho}$ and the number of edges is m; thus, the number of inner-edges is $\frac{m}{\rho}$ stochastically. Therefore, the size of output inner-edges is $\frac{m}{\rho} \times (\rho - 1)$.

If an input edge for a Map instance is an outer-edge which lies between G_i and G_j, it is emitted to G_{ij} and G'_{ijk} where $k \in [0, \rho - 1]$ and $i \neq j \neq k$, i.e. the outer-edge output $1 + (\rho - 2) = \rho - 1$ times. The number of outer-edges is

$$1 - \frac{m}{\rho} = \frac{\rho - 1}{\rho} m$$

Then, the size of output from outer-edges is $\frac{\rho-1}{\rho} m \times (\rho - 1)$. Finally, the total size of output is

$$\frac{m}{\rho} \times (\rho - 1) + \frac{\rho - 1}{\rho} m \times (\rho - 1) = m(\rho - 1) = O(m\rho)$$

□

We analyze the memory usage of each reduce instance in the TTP algorithm.

Lemma 3. Each reduce instance receives input of $O(\frac{m}{\rho^2})$ size on average.

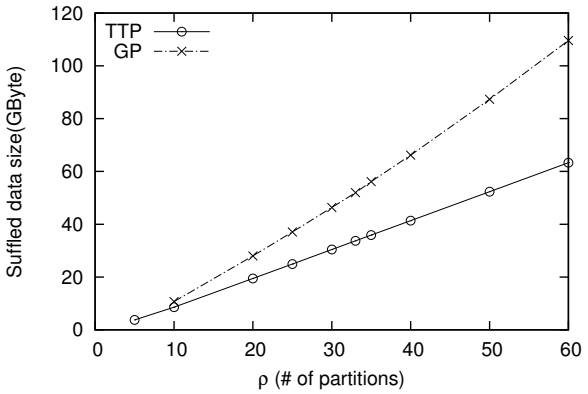
PROOF. Each Reduce instance receives one of two types of input: $\langle (i, j)\ ;\ E_{ij} \rangle$ and $\langle (i, j, k)\ ;\ E'_{ijk} \rangle$, i.e. edges of 2-partitions and edges of 3'-partitions. In the former case, a 2-partition E_{ij} contains both inner-edges and outer-edges. The probability that an edge belong to a specific partition is $\frac{1}{\rho^2}$, because the probability equals to the probability that both of two nodes of the edge belong to a specific partition. In a 2-partition, there are two 1-partitions so the expected size of inner-edges in a 2-partition is $\frac{2m}{\rho^2}$. Similarly, the expected size of outer-edges in a 2-partition is $\frac{m}{\rho^2}$. Therefore, when the input of a Reduce instance is edges of a 2-partition, the expected input size is $\frac{3m}{\rho^2} = O(\frac{m}{\rho^2})$. In the latter case, a 3'-partition E'_{ijk} contains only outer-edges and the size of outer-edges in a 3'-partition is

$$\binom{3}{2} \times \frac{m}{\rho^2} = \frac{3m}{\rho^2} = O(\frac{m}{\rho^2})$$

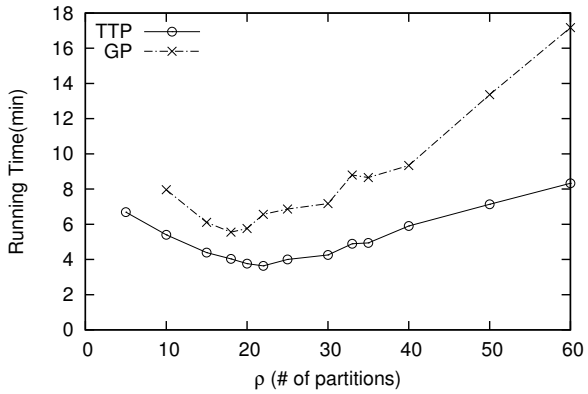
Therefore, regardless of the input type, the expected size of the input to an reduce instance is $O(\frac{m}{\rho^2})$. □

Table 2: The summary for the datasets

Dataset	Nodes	Directed Edges	Undirected Edges	Triangles	File Size
web-BerkStan	6.9×10^5	7.6×10^6	6.6×10^6	6.5×10^7	110 MB
as-Skitter	1.7×10^6	-	1.1×10^7	2.9×10^7	149 MB
soc-LiveJournal1	4.8×10^6	6.9×10^7	4.3×10^7	2.8×10^8	1.1 GB
Twitter	4.2×10^7	1.5×10^9	1.2×10^9	1.5×10^{12}	25 GB



(a) Effect of the partition size ρ on shuffled data size.



(b) Effect of the partition size ρ on the running time.

Figure 6: Effect of the partition size ρ with LiveJournal dataset

Lemma 2 shows that the network and the space usage is proportional to the partition size ρ. Lemma 3 shows that the memory usage of each reduce instance decreases as the partition size increases. The two Lemmas shows that both the total space/network usage and the memory usage of each reduce instance depend on the partition size ρ and the algorithm can mediate between them by modulating the ρ.

Theorem 1. The total amount of work performed by all reduce instances is $O(m^{\frac{2}{3}})$ where $\rho \leq \sqrt{m}$.

PROOF. The running time on each Map instance depends on the size of its edges output. Outputting an edge can be done in $O(1)$. Lemma 2 shows that the total number of edges output is $O(m\rho)$ and it becomes $O(m^{\frac{3}{2}})$ when ρ be less than or equals to $\sqrt{m}$. Lemma 3 shows that the input size of any reduce instance is $O(\frac{m}{\rho^2})$ and the partition size is $O(\rho^3)$. In addition, the running time of the best algorithm for counting triangles is $O(m^{\frac{3}{2}})$(See Section 2). Therefore, the total processing time of reduce instances is

$$O\left(\left(\frac{m}{\rho^2}\right)^{\frac{3}{2}} \times \rho^3\right) = O(m^{\frac{3}{2}})$$

□

Theorem 2. In the Map step, the GP algorithm outputs $O(m\rho)$ more than the TTP algorithm does.

PROOF. First, we consider the case in which the two algorithms use the same number of partitions ρ. For Map instances of the GP algorithm, if the input edge is an inner-edge, the edge is emitted $\binom{\rho-1}{2}$. If the input edge is an outer-edge, the edge is emitted $\rho - 2$. By the Lemma 2, the expected size of inner-edges and outer-edges are $\frac{m}{\rho}$ and $\frac{(\rho-1)m}{\rho}$ respectively. The total outputs size of the map step is

$$\binom{\rho-1}{2} \times \frac{m}{\rho} + (\rho - 2) \times \frac{(\rho-1)m}{\rho} = \frac{3(\rho-1)(\rho-2)m}{2\rho}$$

In addition, the Map step of the TTP algorithm outputs $m(\rho - 1)$ by the lemma 2. Then, the difference between the two output sizes is

$$\frac{(\rho-6)(\rho-1)m}{2\rho} = \Omega(m\rho)$$

We now consider a case in which two algorithms use different ρ to use same memory size in each reduce instance. The number of input edges of the GP algorithm is

$$\binom{3}{1} \times \frac{m}{\rho^2} + \binom{3}{2} \times \frac{m}{\rho^2} = \frac{6m}{\rho^2}$$

In the above expression, the first term represents the number of inner-edges and the second term represents the number of outer-edges in a reduce instance. Lemma 3 shows the number of input edges of the TTP algorithm is $\frac{3m}{\rho^2}$. Suppose that the GP algorithm uses ρ and the TTP algorithm uses ρ_2 then $\rho_2 = \sqrt{2}\rho$ from $\frac{6m}{\rho_2^2} = \frac{3m}{\rho^2}$. Then, the difference of the output sizes of the two algorithm is

$$\frac{3(\sqrt{2}\rho-1)(\sqrt{2}\rho-2)m}{2\sqrt{2}\rho} - m(\rho-1) = \Omega(m\rho)$$

□

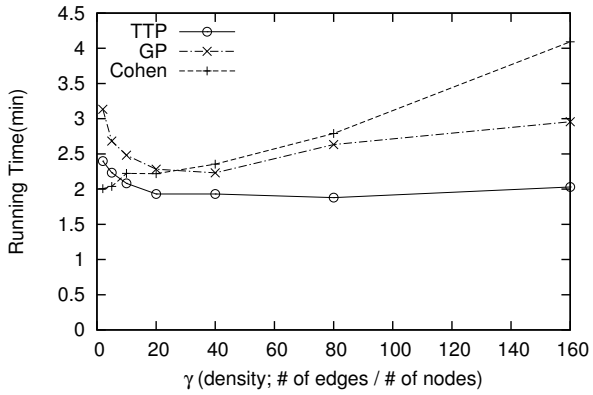


Figure 7: Effect of density with synthetic data. The edge size is fixed.

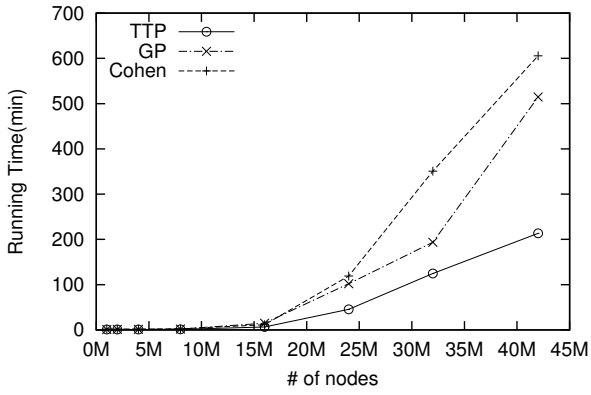


Figure 8: Effect of the graph size with induced subgraphs of twitter data

5. EXPERIMENTS

In this section, we present the results from a comprehensive experimental evaluation to verify the performance of the proposed algorithm.

5.1 Datasets

We use both real world datasets and synthetic datasets to evaluate our algorithm. We first generate synthetic datasets according to the preferential attachment model[5]. Graph sizes vary from 200 thousand nodes to 2 million nodes with 20 million edges. The web-BerkStan, as-Skitter and LiveJournal datasets are from SNAP[2] and the twitter dataset is from [3]. The directed graphs are converted into undirected graphs by deleting duplicate edges. In other words, if there is even one directed edge between two nodes, we consider that there is a both-way edge between the nodes. The real-world datasets are listed in Table 2.

5.2 Implementation

We implement our algorithm and the compare algorithms in hadoop framework that is the de facto standard implementation of MapReduce. All the experiments are conducted on a cluster server with 47 nodes. Each node has 4 Gbyte of memory size. The compact-forward algorithm is used in reduce function of our algorithm and the GP algorithm. The hash function $P(x)$ is implemented as the modular hash function.

5.3 Experimental Results

In this section, we present our experimental results.

5.3.1 *Effect of the partition size ρ*

Figure 6 shows the experimental results for varying the partition size ρ. As proved in Lemma 2, the total output size of map instances, which is the size of shuffled data, is $O(m\rho)$. Moreover, the difference of the shuffled data size between our algorithm and GP algorithm is also $O(m\rho)$. This indicates that the shuffled data size and the difference are directly proportional to the partition size ρ when the edge size m is fixed. Figure 6(a) shows the proportional difference between two algorithms. For counting the number of triangles in the Twitter dataset, 2.1 Tbyte data is shuffled on the GP algorithm($\rho = 45$) while 1.3 Tbyte data is shuffled on the TTP algorithm($\rho = 40$) over reducing 0.8 Tbyte (Table 5). Figure 6(b) shows the running time for varying partition sizes ρ. In every case, the TTP algorithm outperforms the GP algorithm. Both experiments are conducted with the LiveJournal dataset. Note that the GP algorithm yields a memory buffer overflow when the ρ is 5.

Table 3: Induced subgraphs of twitter dataset

Nodes	Edges	Triangles
1.0×10^6	6.7×10^5	5.8×10^6
2.0×10^6	1.1×10^6	8.2×10^6
4.0×10^6	1.8×10^6	1.3×10^7
8.0×10^6	5.4×10^6	5.3×10^7
1.6×10^7	5.2×10^7	7.6×10^8
2.4×10^7	2.6×10^8	9.7×10^9
3.2×10^7	4.2×10^8	1.6×10^{10}

5.3.2 *Effect of graph properties*

Effect of density. For the experiments, synthetic datasets, which comply with the preferential attachment model[5], are used. The data model enables the control over the number of nodes n and the density of the graph, which are represented as γ. The γ decides the number of edges m when the number of nodes n is determined. Figure 7 shows the experimental results for varying γ and n while m is fixed. In other words, the result shows the influence of the density of the graphs when the input data size is fixed. The experiments are conducted with graphs which have 20 million edges and whose data file size is about 250 Mbyte. For the TTP and GP algorithms, the value of ρ is chosen as 15 and 16 respectively with which the performance of the algorithms is the best. The GP algorithm takes more time for counting triangles than the TTP algorithm on all occasions regardless of γ. The Cohen algorithm is better when γ is lower than aproximately 8 but the performance gradually declines as the density increases. Note that most graphs have γ greater than 8 including what we used. The γ of web-BerkStan, as-Skitter, soc-LiveJournal1, and Twitter are 9.70, 6.47, 8.96, 28.6 respectively. Especially, the γ of Facebook is more than 80. As we mentioned in Section 2, The reason why the running time of Cohen increases as γ increases is that the output size of the first MapReduce phase is proportional to the

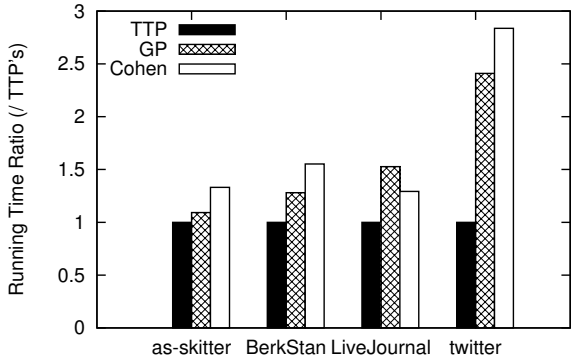


Figure 9: the ratio of the running time with real world datasets

degree of nodes in the algorithm. As γ increases, the number of nodes decreases so that the degree of nodes increases because the number of edges is fixed.

Effect of graph size. For the experiments, twitter data [3] is used. We generated induced subgraphs of the twitter graph on 1, 2, 4, 8, 16, 24 and 32 millions nodes. The generated datasets are listed in Table 3. Figure 8 shows the experimental results for varying graph sizes. The TTP algorithm outperforms other algorithms in most cases. Especially, the performance gap between the algorithms increases as the data size increases. For the TTP and GP algorithms, the value of ρ is chosen as 15 and 16 respectively where the node size is lower than 25 million, and the value is chosen 31 and 32 respectively in the other cases because of the memory buffer overflow.

5.3.3 Results of Real Datasets

We conduct experiments on real datasets to check the proposed algorithm is feasible for real world data. Figure 9, Table 5 and Table 4 show the results. The running time of each algorithm is listed in Table 4. In order to show the speed-up of our algorithm comparing to other algorithms, Figure 9 shows the ratio of the running time. In the figure, the running time of each algorithm is divided by the running time of our algorithm. For all datasets, The TTP algorithm outperforms other algorithms. Especially for twitter dataset, the TTA algorithm is more than twice as fast as the others.

6. CONCLUSION

Counting the number of triangles is a fundamental problem in various domains such as social networks, P2P networks, world wide web, and road networks. In order to solve the problem in enormous graphs, prior researchers presented algorithms that operates on MapReduce framework; Cohen[9] proposed a MapReduce algorithm, called Cohen algorithm, based on node-iterator, and Suri and Vassilvitskii[22] proposed another MapReduce algorithm, namely GP algorithm, based on graph partitioning. However, previous algorithms have weaknesses. The performance of Cohen algorithm is diminished as the average degree of the graph becomes greater. The GP algorithm generates much intermediate data redundantly that cause network overload and prolong the total running time. In this paper, we propose a new MapReduce algorithm based on graph partitioning like as the GP algorithm for counting the number of triangles in enormous graphs. The TTP algorithm substantially reduces the duplication by classifying the type of triangles and processing triangles by the type. The experimental evaluation shows that the TTP algorithm outperforms provious MapReduce algorithms in most cases. Especially, for a twitter dataset, the proposed algorithm is more than twice as fast as existing algorithms. Moreover, the performance gap increases as the graph becomes larger and denser.

Table 4: The running time of all algorithms (min)

Dataset	TTP	GP	Cohen
web-BerkStan	1.31	1.68	2.03
as-skitter	1.62	1.78	2.17
soc-LiveJournal1	3.63	5.55	4.70
twitter	213	514	605

Table 5: The shuffled data sizes of all algorithms (GB)

Dataset	TTP	GP	Cohen
web-BerkStan	2.04	2.60	2.09
as-Skitter	1.10	1.24	2.15
soc-LiveJournal1	21.6	24.4	16.8
Twitter	1300	2140	4340

7. ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea grant funded by the Korean government (MSIP) (No. NRF-2009-0081365).

8. REFERENCES

[1] http://newsroom.fb.com/Key-Facts.
[2] http://snap.stanford.edu/.
[3] http://an.kaist.ac.kr/pub_date.html.
[4] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
[5] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.
[6] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–24. ACM, 2008.
[7] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips. Tolerating the community detection resolution limit with edge weighting. *Physical Review E*, 83:056119, 2011.
[8] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 672–680, 2011.
[9] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[10] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

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

[12] R. Dementiev. *Algorithm engineering for large data sets*. PhD thesis, Doktorarbeit, Universität des Saarlandes, 2006.

[13] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the national academy of sciences*, 99(9):5825–5829, 2002.

[14] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *Proceedings of the 2013 ACM SIGMOD international conference on Management Of data*, pages 325–336, 2013.

[15] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[16] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Ninth IEEE International Conference on Data Mining*, pages 229–238, 2009.

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

[18] B. Menegola. An external memory algorithm for listing triangles. Technical report, Universidade Federal do Rio Grande do Sul, 2010.

[19] J. Myung and S.-g. Lee. Matrix chain multiplication via multi-way join algorithms in mapreduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, pages 53:1–53:5, 2012.

[20] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social networks*, 31(2):155–163, 2009.

[21] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer, 2005.

[22] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011.

[23] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. 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*, pages 837–846, 2009.

[24] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[25] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[26] T. White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[27] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 259–268, 2011.