IEICE TRANS. INF. & SYST., VOL.E102–D, NO.10 OCTOBER 2019

PAPER

An Efficient Parallel Triangle Enumeration on the MapReduce Framework*

Hongyeon KIM[†a)], ***Nonmember and*** **Jun-Ki MIN**[†b)], ***Member***

SUMMARY A triangle enumerating problem is one of fundamental problems of graph data. Although several triangle enumerating algorithms based on MapReduce have been proposed, they still suffer from generating a lot of intermediate data. In this paper, we propose the efficient MapReduce algorithms to enumerate every triangle in the massive graph based on a vertex partition. Since a triangle is composed of an edge and a wedge, our algorithms check the existence of an edge connecting the end-nodes of each wedge. To generate every triangle from a graph in parallel, we first split a graph into several vertex partitions and group the edges and wedges in the graph for each pair of vertex partitions. Then, we form the triangles appearing in each group. Furthermore, to enhance the performance of our algorithm, we remove the duplicated wedges existing in several groups. Our experimental evaluation shows the performance of our proposed algorithm is better than that of the state-of-the-art algorithm in diverse environments.
key words: *triangle enumeration, vertex partition, graph data, MapReduce*

1. Introduction

A *triangle enumeration problem* is to generate every triangle in a graph G consisting of a set of edges E and a set of vertices V. A triangle is composed of three vertices that are connected by three edges. The triangle enumeration problem is interesting in diverse applications such as k-truss [17], dense neighborhood graph [18] and clustering coefficient [19]. A k-truss is a largest subgraph in a graph such that each edge belongs to at least $k-2$ triangles in the subgraph where k is an integer greater than 2. Similar to a k-truss, a dense neighborhood graph with a threshold λ is a subgraph such that every vertex pair shares at least λ common neighbors. In addition, as a well known measurement, a clustering coefficient quantifies how well connected are the neighbors of a vertex in a graph. Thus, the triangle enumeration problem is important to obtain the meaningful results from a graph.

As the well-known traditional algorithms to enumerate every triangle in a graph, there are *node-iterator* and *edge-iterator* [15] algorithms. Given a graph, the *node-iterator* algorithm investigates every vertex and checks whether each pair of neighbors for each vertex is connected by an edge or not. Meanwhile, the *edge-iterator* algorithm investigates every edge and identifies the common neighbors of each edge's end-nodes. However, since both traditional algorithms are running on the single machine having the limited memory, it is hard to enumerate triangles in enormous graphs such as Social Network Services (SNS) and World Wide Web (WWW) [3], [6], [8].

In recent years, many parallel algorithms for triangle enumeration utilizing MapReduce [2], [4], [10]–[13], [21] are proposed in order to alleviate the problem of the traditional algorithms. As a programming model to handle the massive data set, MapReduce [5] distributes the massive data set across several machines in a cluster. The MapReduce algorithms for triangle enumeration can be categorized as follows: Partitioning graphs [16] and Iterating multi-rounds [12], [21]. The partitioning algorithms divide a given graph into several subgraphs and enumerate the triangles for every subgraph in parallel. Meanwhile, the multi-rounds algorithms consisting of several MapReduce rounds divide a graph into several subgraphs each of which can be handled in each MapReduce round and enumerate the triangles in each subgraph.

Obviously, these parallel algorithms are more efficient and scalable than the traditional serial algorithms since these parallel algorithms use a cluster consisting of several machines. However, the MapReduce algorithms still suffer from a critical problem such that a lot of intermediate results are generated during parallel execution. These intermediate results are not only sorted and merged but also written to and read from the disks of the machines through the network. Such massive intermediate results cause the excessive disk I/O and the network congestion resulting in the performance degradation of the parallel algorithms. Thus, in this paper, we propose novel MapReduce algorithms, called *PBTE* and *EPBTE*, based on a vertex partition to enumerate every triangle in a massive graph efficiently.

To explain the features of our work, we first present a basic parallel algorithm, called *BTE*. In *BTE*, every wedge and edge in a graph are generated where a wedge is a length-2 path between two vertices. Since a triangle is composed of a wedge and an edge, we check whether the end-nodes of each wedge are connected by an edge.

To improve the performance of triangle enumeration, we devised *PBTE* in which the vertex set of a graph is split into several disjoint vertex partitions. Then, since end-nodes of a wedge (and an edge) belongs to different vertex partitions, we group the wedges (and the edges) according to each pair of vertex partitions. For each group containing the

Manuscript received December 11, 2018.
Manuscript revised April 27, 2019.
Manuscript publicized July 11, 2019.
[†]The authors are with Korea Univ. of Tech. & Edu., Korea.
*This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Science and ICT (2019R1F1A1062511).
a) E-mail: zenweird@koreatech.ac.kr
b) E-mail: jkmin@koreatech.ac.kr (Corresponding author)
DOI: 10.1587/transinf.2018EDP7421

wedges and edges associated with each pair of vertex partitions, we generate the triangles in parallel. However, in *PBTE*, the end-nodes of each wedge appear in several pairs of vertex partitions redundantly. Thus, the size of intermediate results becomes large and the performance of *PBTE* may be degraded. To alleviate such a problem, we implement the enhanced parallel algorithm, called *EPBTE* in which we minimize the size of intermediate results storing in the distributed file system. In *EPBTE*, we separate the end-nodes of wedges into several groups according to vertex partitions rather than vertex partition pairs and store them into separate files. Then, for each pair of vertex partitions, we can generate the wedges by merging the end-nodes stored in the corresponding two files.

To show the efficiency and scalability of our proposed algorithms *PBTE* and *EPBTE*, we implemented them and compared with the state-of-the-art algorithm *PTE* [11] and the multi-rounds algorithm *CTTP* [12]. Our experimental results demonstrate that the performances of our algorithms are superior to the state-of-the-art algorithms.

Contributions. Our main contributions are summarized as follows:

- We propose the efficient MapReduce algorithms *PBTE* and *EPBTE* to enumerate every triangle in a graph, which split the graph into disjoint vertex partitions and evaluate each pair of vertex partitions to enumerate the triangles appearing in the pair of vertex partitions.
- We present how to compute the number of vertex partitions with respect to the available memory space of each machine participated in the MapReduce framework for our algorithms since the performances of *PBTE* and *EPBTE* are affected by the number of vertex partitions.
- We empirically evaluated the performances of our proposed algorithms *PBTE* and *EPBTE* with the real-life data sets to show the efficiency and scalability of them.

The rest of this paper is organized as follows: Section 2 contains the properties of triangles and the details of MapReduce. In Sect. 3, we review various MapReduce algorithms for triangle enumeration. We describe the intuition and the details of our algorithm in Sect. 4. In Sect. 5, we show the results of our experiments. Lastly, we summarize our work in Sect. 6.

2. Preliminaries

In this section, we first explain the properties of the triangles in a graph and next briefly present MapReduce.

2.1 Properties of Triangles

Let $G = (V, E)$ be an undirected and unweighted simple graph where V is a set of vertices and E is a set of edges such that $E \subseteq V \times V$. For a pair of vertices u and v in V, if there exists an edge $e = (u, v)$ in E, we say u is *adjacent to* v. For each $u \in V$, $N(u) = \{v \in V : (u, v) \in E\}$ is a set of

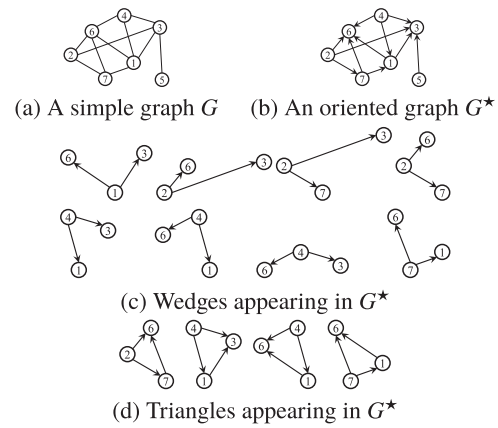

(a) A simple graph G
(b) An oriented graph $G^\star$
(c) Wedges appearing in $G^\star$
(d) Triangles appearing in $G^\star$

Fig. 1 An example for an oriented graph $G^\star$

u's *neighbors* where a neighbor of u is a vertex adjacent to u and the *degree* of a vertex u, denoted by $deg(u)$, is defined as $|N(u)|$.

A *triangle*, denoted by Δ_{uvw}, in a graph G is a complete subgraph consisting of three vertices u, v and w. In other words, when a triangle Δ_{uvw} appears in G, E contains three edges $e = (u, v)$, $e' = (u, w)$ and $e'' = (v, w)$. Given a graph G, a set of all triangles in G is denoted by $\Delta(G)$ and the number of triangles in $\Delta(G)$ is denoted by $|\Delta(G)|$. Meanwhile, a *wedge*, denoted by ω_{uvw}, is a subgraph of three vertices u, v and w in V such that there are two edges $e = (u, v)$ and $e'(u, w)$ in E. A wedge ω_{uvw} can also be seen as a 2-length path whose the end-nodes are v and w. For a wedge ω_{uvw} such that two edges $e = (u, v)$ and $e' = (u, w)$ exist in E, if an edge $e'' = (v, w)$ belongs to E, a triangle Δ_{uvw} is formed.

Similar to the most related literatures [7], [21], we enumerate every triangle appearing in an oriented graph $G^\star = (V, E^\star)$ of a simple graph G since it is convenient to work with $G^\star$ rather than G. Given an oriented graph $G^\star = (V, E^\star)$ of G, $E^\star$ is a set of these directed edges. To explain an oriented graph $G^\star$ of G, we first define a *total order* $\prec$ on the vertex set V of G. For a pair of vertices u and v in V, if $(deg(u) < deg(v))$ or $(deg(u) = deg(v)$ and $id(u) < id(v))$, u precedes v, denoted by $u \prec v$, where $id(u)$ is the unique identifier of a vertex u in V. For each edge $e = (u, v)$ in E, if $u \prec v$, there is a directed edge from u to v in $E^\star$ of $G^\star$.

To represent the neighbors of a vertex $u \in V$ of $G^\star$ which are preceded by u, we define these neighbors of u as follows:

Definition 1: Given an oriented graph $G^\star = (V, E^\star)$ of a simple graph $G = (V, E)$, the set of *out-neighbors* of a vertex $u \in V$ is defined as $N^+(u) = \{v : (u, v) \in E^\star \text{ where } u \prec v\}$.

For instance, given a simple graph G plotted in Fig. 1 (a), the oriented version $G^\star = (V, E^\star)$ of G is shown in Fig. 1 (b) where the vertex set V consists of 7 vertices and the directed edge set $E^\star$ is composed of 11 directed edges. The integer number at each vertex of G and $G^\star$ shown in Fig. 1 (a) and (b) represents the identifier of the vertex. The edge $(1, 4)$ in Fig. 1 (a) becomes the directed edge $(4, 1)$ in

Fig. 1 (b) since $deg(4) < deg(1)$. In addition, there is a directed edge from a vertex 1 to a vertex 6 in Fig. 1 (b) since $deg(1) = deg(6)$ and $id(1) < id(6)$. Moreover, the out-neighbor set of a vertex 4 is $N^+(4) = \{1, 3, 6\}$.

Given an oriented graph $G^\star = (V, E^\star)$ of a simple graph G, we define a triangle enumeration problem on $G^\star$ as follows:

Definition 2: Let a set of every triangle appearing in a simple graph G be $\Delta(G)$. The problem of triangle enumeration is to discover every triangle in $\Delta(G)$ one by one by exploring the oriented graph $G^\star$ of G.

For an oriented graph $G^\star$ of a simple graph G, we now define a wedge and a triangle, referred to as $\omega^\star_{uvw}$ and $\Delta^\star_{uvw}$, respectively. For each wedge ω_{uvw} in G, there always exists the unique total order among the vertices u, v and w. Assume that $u \prec v \prec w$. Then, there always exist a pair of directed edges (u, v) and (u, w) in $E^\star$ which forms a wedge $\omega^\star_{uvw}$ in $G^\star$. Similarly, for each triangle Δ_{uvw} in G, there always exists a triangle $\Delta^\star_{uvw}$ in $G^\star$ consisting of three directed edges (u, v), (u, w) and (v, w) where $u \prec v \prec w$. Moreover, for a triangle $\Delta^\star_{uvw}$ or a wedge $\omega^\star_{uvw}$ in $G^\star$, a vertex u is called a *cone vertex* and a directed edge (v, w) in $\Delta^\star_{uvw}$ is called a *pivot edge*, respectively.

For example, an oriented graph $G^\star$ in Fig. 1 (b) has eight wedges $\omega^\star_{136}$, $\omega^\star_{236}$, $\omega^\star_{237}$, $\omega^\star_{267}$, $\omega^\star_{413}$, $\omega^\star_{416}$, $\omega^\star_{436}$ and $\omega^\star_{716}$ as shown in Fig. 1 (c) as well as four triangles $\Delta^\star_{267}$, $\Delta^\star_{413}$, $\Delta^\star_{416}$ and $\Delta^\star_{716}$ as shown in Fig. 1 (d). In addition, the vertex 2 is the cone vertex of $\Delta^\star_{267}$, $\omega^\star_{236}$, $\omega^\star_{237}$ and $\omega^\star_{267}$ as well as the directed edge $(7, 6)$ is the pivot edge of $\Delta^\star_{267}$.

2.2 MapReduce

Google developed the MapReduce [5] that enables the users to easily develop large scale distributed applications. MapReduce is a distributed and parallel processing model as well as execution environment in the shared-nothing cluster of multiple commodity machines. The machines participated in the MapReduce framework are classified as a single master and several slaves according to their roles. Slaves process the data in parallel as well as a master manages the processing for data in the slaves, the status of network and the distributed data to prevent the system faults, the violation for data integrity and so on. Hadoop [1] is implemented in the OpenSource community as the MapReduce framework. In Hadoop, using the Hadoop Distributed File System (HDFS), a large sized file is initially partitioned into several fragments, called *chunk*, and stored in several machines redundantly for reliability.

MapReduce consists of three phase such as map, shuffle and reduce phases. In the map phase, a *mapper* (a.k.a. a map instance) created by each machine takes a chunk from the input data file and invokes several map functions. A map function takes a key-value pair $(key_1; value_1)$ as input, executes some computation and may output a set of intermediate key-value pairs $(key_2; value_2)$. The key-value pairs emitted by all map functions are sorted and merged by each key in the shuffle phase. In the reduce phase, a *reducer* (a.k.a. a reduce instance) created by each machine invokes reduce functions with each distinct key. The reduce function takes the list of all values sharing the same key and may output the key-value pairs.

3. Related Works

In this section, we briefly described the several algorithms for triangle enumeration.

In [7], a serial algorithm called MGT (Massive Graph Triangulation) was proposed. MGT enumerates every triangle on single machine. Similar to node-iterator [15], MGT travels all vertices in a memory and checks whether two neighbors of each vertex are adjacent to each other. If two neighbors are adjacent, a triangle can be formed. However, MGT cannot handle the massive graph having a lot of vertices due to a limited memory space of a single machine.

To solve the limited memory problem, MapReduce algorithms for triangle enumeration have been proposed based on the traditional algorithms such as node-iterator, edge-iterator and graph partition. To count the number of triangles in a graph based on MapReduce, Suri and Vassilvitskii proposed an efficient parallel algorithm called GP [16]. GP first splits the vertex set of the graph into ρ partitions and counts the number of triangles for each 3-partition $G_{i,j,k}$ where $0 < i < j < k \leq \rho$. In GP, every triangle is classified into one of three types as follows:

- **Type 1**: Three vertices of a triangle belong to the same partition.
- **Type 2**: Two vertices of a triangle belong to the same partition and the remaining vertex belong to a different partition.
- **Type 3**: Three vertices are in the distinct partitions.

Then, triangles of type 1 and type 2 appear multiple times by evaluating every possible 3-partition. Thus, to reduce the redundant computation of the duplicated triangles in GP, another parallel counting algorithm called TTP [10] was proposed. Similar to GP, TTP divides the vertex set of the graph into ρ partitions. However, in contrast to GP, TTP counts the numbers of triangles of type 1 and type 2 by investigating every 2-partition. Meanwhile, let an edge (u, v) be an outer-edge when u and v are in different partitions. Then, the number of triangles of type 3 is computed by using $3'$-partition which is a subgraph of a 3-partition consisting of outer-edges only. To compute triangles, TTP stores each edge $\rho - 1$ times [10].

Although the above triangle counting algorithms can be adapted to the triangle enumeration algorithm, these algorithms generate the duplicated triangles. Thus, for the triangle enumeration problem, Park et al. [12] proposed the CTTP (Colored TTP) algorithm which consists of multiple MapReduce rounds to reduce the amount of intermediate results required for each round. However, similar to TTP, CTTP also stores each edge $\rho - 1$ times. In order to reduce the intermediate results generated in TTP and CTTP, the pre-

partitioned triangle enumeration algorithm called PTE [11] was proposed recently. PTE splits the graph into ρ partitions and stores the partitions into the separate files in a distributed file system. Since each edge is stored only once, the number of intermediate results generated by PTE is less than that of TTP and CTTP. However, to enumerate the triangles appearing in each $3'$-partition, PTE repeatedly reads the edges stored in the multiple files for $3'$-partitions.

4. Proposed Algorithms

In this section, we propose the parallel algorithms for the triangle enumeration problem running on the MapReduce framework. We first introduce the basic parallel algorithm in order to explain the intuition of our proposed algorithms. We next present the partition based algorithm to generate triangles in a graph efficiently. Furthermore, to reduce the intermediate results generated by the partition based algorithm and thus improve the performance by avoiding the overhead caused by redundant intermediate results, we enhanced the partition based algorithm to a novel algorithm called Enhanced Partitioned Based Triangle Enumeration (abbreviated by *EPBTE*), which eliminates the duplicated intermediate results occurred in the partition based algorithm.

4.1 Basic Algorithm for Triangle Enumeration

We first present a basic MapReduce algorithm, called Basic Triangle Enumeration (abbreviated by *BTE*), for enumerating all triangles in an oriented graph $G^\star$.

In an oriented graph $G^\star$, each triangle $\Delta^\star_{uvw}$ is composed of a wedge $\omega^\star_{uvw}$ and a pivot edge (v, w). In other words, if the end-nodes v and w of $\omega^\star_{uvw}$ are connected by a directed edge (v, w), a triangle $\Delta^\star_{uvw}$ is formed. Thus, we first enumerate every wedge in an oriented graph and then we check whether the end-nodes of each wedge are connected by a directed edge. To generate every wedge in an oriented graph $G^\star$, we develop the following lemma.

Lemma 1: Given the oriented graph $G^\star = (V, E^\star)$ of a simple graph $G = (V, E)$, every pair of vertices v and w in the set of out-neighbors $N^+(u)$ of a vertex $u \in V$ becomes the end-nodes v and w of the wedge $\omega^\star_{uvw}$ whose cone vertex is u.

Proof. (By contradiction) For a wedge $\omega^\star_{uvw}$, assume that two end-nodes v and w do not belong to $N^+(u)$. By definition of a wedge $\omega^\star_{uvw}$ in an oriented graph $G^\star$, a pair of directed edges (u, v) and (u, w) exist in $E^\star$. Thus, v and w should belong to $N^+(u)$. This contradicts the above assumption. Thus, every pair of vertices v and w in $N^+(u)$ of u become the end-nodes v and w of $\omega^\star_{uvw}$ whose a cone vertex is u. ■

By Lemma 1, we can enumerate every wedge by collecting each out-neighbor set $N^+(u)$ of every vertex u and then we can form each triangle in an oriented graph $G^\star$ by checking whether there is a directed edge between every pair of vertices v and w in out-neighbor set $N^+(u)$.

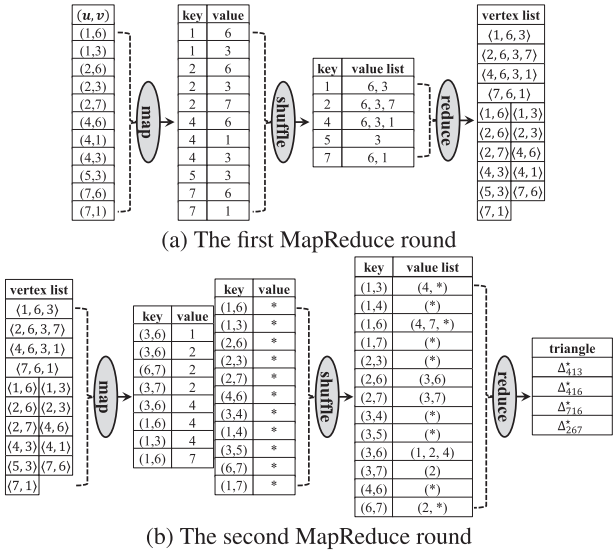
For example, given an oriented graph $G^\star$ in Fig. 1 (b), the out-neighbor set $N^+(4)$ of a vertex 4 consists of three vertices 1, 3 and 6 (i.e., $N^+(4) = \{1, 3, 6\}$). Then, each pair of vertices in $N^+(4)$ becomes the end-nodes of three wedges $\omega^\star_{413}$, $\omega^\star_{416}$ and $\omega^\star_{436}$ by Lemma 1. Then, since there are directed edges $(1, 3)$ and $(1, 6)$ in $G^\star$, triangles $\Delta^\star_{413}$ and $\Delta^\star_{416}$ are generated.

For the out-neighbor set $N^+(u)$ of a vertex u, the number of wedges whose cone vertex is u becomes $\binom{|N^+(u)|}{2} = |N^+(u)| \cdot (|N^+(u)| - 1)/2$. However, the number of out-neighbors of u (i.e., $|N^+(u)|$) is at most the degree of u (i.e., $deg(u)$). In addition, as mentioned in Sect. 2, to build an oriented graph $G^\star$ of a graph G, we define the *total order* on the vertex set V of G such that, when there is a directed edge (u, v) in $E^\star$ of $G^\star$, the degree of u is at most that of v (i.e., $deg(u) \leq deg(v)$). Thus, each vertex in V having high degree has a small number of out-neighbors. Furthermore, for a wedge $\omega^\star_{uvw}$ consisting of a pair of directed edges (u, v) and (u, w), $deg(u)$ is at most $min(deg(v), deg(w))$. Thus, a vertex having high degree tends to not be a cone vertex of a wedge in an oriented graph. Therefore, the number of out-neighbors for every vertex in V of $G^\star$ is relatively small compared to the degree of every vertex in V of G.

After collecting the out-neighbor set $N^+(u)$ of each vertex u, we check whether there is a directed edge between every pair of vertices in $N^+(u)$ since each pair of vertices in $N^+(u)$ becomes the end-nodes of a wedge whose cone vertex is u by Lemma 1. Then, if the end-nodes of a wedge are adjacent to each other, the wedge and the directed edge are merged to form a triangle.

Our basic algorithm *BTE* consists of two MapReduce rounds for enumerating triangles in an oriented graph. In the first MapReduce round, each map function invoked with a directed edge (u, v) in $E^\star$ of an oriented graph $G^\star$ emits a key-value pair $(u; v)$. During the shuffle phase, the key-value pairs emitted by all map functions are sorted and grouped by each key u. Thus, each reduce function called with each key u taking the out-neighbor set $N^+(u) = \{o_1, \ldots, o_{|N^+(u)|}\}$ of u outputs the vertex u and its $N^+(u)$ as a list $\langle u, o_1, \ldots, o_{|N^+(u)|}\rangle$ when $|N^+(u)| \geq 2$ since u cannot be a cone vertex of a wedge if $|N^+(u)|$ is less than two. Moreover, the reduce function called with u also outputs every directed edge (u, v) as a list $\langle u, v\rangle$ for each out-neighbor $v \in N^+(u)$ since (u, v) can be a pivot edge of another triangle.

In the second round, each map function takes a list generated at the previous round. As mentioned above, since each reduce function of the first round generates a list $\langle u, v\rangle$ for a directed edge (u, v) as well as a list $\langle u, N^+(u)\rangle$ when $|N^+(u)| \geq 2$, it is easy to distinguish between directed edge and out-neighbor set of u according to the number of entries in the list. Note that, by adopting the *total order*, we obtain the oriented graph of a simple graph. However, it is hard to preserve the *total order* among all vertices in $N^+(u)$ since the degree of every vertex is needed to calculate the *total order* among them. Furthermore, to check whether two end-nodes of a wedge are adjacent or not, the direction of the directed edge is irrelevant. Thus, for an out-neighbor set



(a) The first MapReduce round

(b) The second MapReduce round

Fig. 2 The behavior of *BTE* algorithm

$N^+(u)$ of a vertex u, the map function emits every pair of vertices in $N^+(u)$ as a key-value pair $(o_i \in N^+(u), o_j \in N^+(u); u)$ where $id(o_i) < id(o_j)$. In addition, for a directed edge (v, w), the map function simply emits a key-value pair $(v, w; *)$ if $id(v) < id(w)$, $(w, v; *)$ otherwise. In this case, a sign $'*'$ means that (v, w) exists in E of G.

The key-value pairs emitted by all map functions are grouped by their key in the shuffle phase. Thus, the wedges whose end-nodes are v and w as well as the edges (v, w) are grouped during the shuffle phase. After shuffle phase, each reduce function takes a key (v, w) and a value list L. If the value list L contains $'*'$ and $|L|$ is at least two, since $|L| - 1$ wedges whose end-nodes (v, w) are adjacent to each other, the reduce function enumerates $|L| - 1$ triangles each of whose pivot edge is (v, w) and cone vertex belongs to L. Otherwise, the reduce function does not generate any triangle.

The following example illustrates the behavior of *BTE* algorithm.

Example 1: Given an oriented graph consisting of 7 vertices and 11 directed edges in Fig. 1 (b), in the first round, each map function taking a directed edge (u, v) emits $(u; v)$ as a key-value pair. For instance, the map function called with $(1, 6)$ emits a key-value pair $(1; 6)$ as shown in Fig. 2 (a). The key-value pairs having the same keys are grouped by the shuffle phase. Thus, each reduce function takes a vertex u as a key and its out-neighbor set $N^+(u)$ as a value list. Then, each reduce function outputs $\langle u, N^+(u)\rangle$ as well as $\langle u, v\rangle$ where $v \in N^+(u)$. For example, as shown in Fig. 2 (a), the reduce function invoked with a key 1 and a value list $\langle 6, 3\rangle$ outputs $\langle 1, 6, 3\rangle$ as well as $\langle 1, 6\rangle$ and $\langle 1, 3\rangle$.

In the second round, each map function taking a list formed $\langle v, w\rangle$ emits a key-value pair $\langle v, w; *\rangle$. Meanwhile, each map function taking $\langle u, N^+(u)\rangle$ emits every pair of vertices in $N^+(u)$ as a key and u as a value. For instance, as shown in Fig. 2 (b), the map function taking $\langle 1, 6\rangle$ emits $(1, 6; *)$ and the map function with a list $\langle 7, 6, 1\rangle$ emits $(1, 6; 7)$. During shuffle phase, the emitted key-value pairs are grouped by their key. Then, each reduce function takes a key (v, w) and a value list L to generate triangles. For example, the reduce function called with a key $(1, 6)$ and a value list $\langle 4, 7, *\rangle$ outputs two triangles $\Delta^\star_{416}$ and $\Delta^\star_{716}$ since there is an edge $(1, 6)$ and two wedges $\omega^\star_{416}$ and $\omega^\star_{716}$.

4.2 Partition Based Algorithm for Triangle Enumeration

In this section, we introduce a parallel algorithm for the triangle enumeration problem, called Partition Based Triangle Enumeration (abbreviated by *PBTE*), in which every out-neighbor set of each vertex is partitioned with respect to their identifiers.

In our basic algorithm *BTE* consisting of two MapReduce rounds, the reduce function of the second round is invoked for each vertex pair which represent an edge or the end-nodes of a wedge. Thus, due to the function call overhead caused by the large number of edges and wedges, the overall performance of *BTE* is degraded. To improve the performance to generate triangles, in *PBTE*, wedges and edges are split into several groups and then each reduce function generates triangles in each group.

4.2.1 Overview of PBTE

The motivation of *PBTE* is that a set of vertices V of an oriented graph $G^\star = (V, E^\star)$ is split into ρ disjointed partitions $p_1, p_2, \ldots, p_\rho$ and then, for each pair of vertex partitions p_i and p_j with $1 \le i \le j \le \rho$, directed edges (v, w) and wedges $\omega^\star_{uvw}$ are grouped and stored into a file on HDFS where v is in p_i and w is in p_j (or v is in p_j and w is in p_i).

Then, we evaluate the edges and wedges stored in each file to enumerate triangles appearing in each pair of vertex partitions. To explain our partition based parallel algorithm *PBTE*, we first define some notations as follows:

Definition 3: Given an oriented graph $G^\star = (V, E^\star)$ and an integer ρ, a vertex partition for the vertex set V, denoted by p_i, is a subset of V such that $p_i \cap p_j = \emptyset$ where $1 \le i, j \le \rho$ and $\bigcup_{i=1,\ldots,\rho} p_i = V$.

Definition 4: Given an oriented graph $G^\star = (V, E^\star)$ and ρ number of vertex partitions $p_1, p_2, \ldots, p_\rho$ for V, an edge partition, denoted by $E^\star_{i,j}$, is defined as $E^\star_{i,j} = \{(v, w) \in E^\star : (v \in p_i, w \in p_j) \text{ or } (v \in p_j, w \in p_i)\}$ where $1 \le i \le j \le \rho$.

In the above definition, we do not define the edge partition $E^\star_{j,i}$ with $i < j$ since $E^\star_{j,i}$ is equal to $E^\star_{i,j}$.

Definition 5: Given ρ number of vertex partitions p_1, $p_2, \ldots, p_\rho$ for the vertex set V of an oriented graph $G^\star = (V, E^\star)$, the out-neighbor set of a vertex $u \in V$ on a vertex partition p_i with $1 \le i \le \rho$, referred to as $N^+_i(u)$, is defined as $N^+_i(u) = \{v : v \in N^+(u) \text{ and } v \in p_i\}$. Furthermore, the out-neighbor set of a vertex $u \in V$ on a pair of vertex partitions p_i and p_j with $1 \le i \le j \le \rho$, denoted by $N^+_{i,j}(u)$, is

Function PBTE $(G^\star = (V, E^\star), \rho)$
$G^\star = (V, E^\star)$: an oriented graph of G,
ρ: the number of vertex partitions
begin
1. *PartitionPairs* = RunMapReduce(Partition, $E^\star$, ρ);
2. $\Delta(G)$ = RunMapReduce(Enumeration, *PartitionPairs*);
3. **return** $\Delta(G)$;
end

Fig. 3 The *PBTE* algorithm

defined as $N^+_{i,j}(u) = N^+_i(u) \cup N^+_j(u)$. Then, a wedge partition, denoted as $N^+_{i,j}$, is defined as $\bigcup_{\forall u \in V} N^+_{i,j}(u)$.

By splitting V into ρ vertex partitions, every vertex $u \in V$ belongs to one of ρ vertex partitions. For a pivot edge (v, w) of a triangle $\Delta^\star_{uvw}$, let us assume that v is in p_i and w is in p_j. Then, the pivot edge (v, w) exists in an edge partition $E^\star_{i,j}$. Furthermore, for the wedge $\omega^\star_{uvw}$ composing $\Delta^\star_{uvw}$, the end-nodes v and w of $\omega^\star_{uvw}$ exist in the wedge partition $N^+_{i,j}$ since v is in p_i and w is in p_j. Thus, by investing the edge partition $E^\star_{i,j}$ and wedge partition $N^+_{i,j}$ together, we can construct the triangle $\Delta^\star_{uvw}$. From the above observation, we have the following lemma.

Lemma 2: Given an oriented graph $G^\star = (V, E^\star)$ and ρ number of vertex partitions for a vertex set V, let $P_{i,j}$ be the union of a wedge partition $N^+_{i,j}$ and an edge partition $E^\star_{i,j}$ for a pair of vertex partitions p_i and p_j with $1 \le i \le j \le \rho$. Then, each triangle $\Delta^\star_{uvw}$ in $G^\star$ appears only in a single $P_{i,j}$ where v is in p_i and w is in p_j or v is in p_j and w is in p_i.

Proof. Given an oriented graph $G^\star = (V, E^\star)$, a triangle $\Delta^\star_{uvw}$ in $G^\star$ is composed of a wedge $\omega^\star_{uvw}$ whose cone vertex is u and a pivot edge (v, w). Let us assume that the vertices v and w belong to the vertex partition p_i and p_j, respectively, where $1 \le i \le j \le \rho$. Then, by Definition 4, the pivot edge (v, w) of $\Delta^\star_{uvw}$ appears in a single edge partition $E^\star_{i,j}$ only. Furthermore, by Lemma 1, the end-nodes v and w of the wedge $\omega^\star_{uvw}$ belong to the out-neighbor set $N^+(u)$ of u. Then, by Definition 5, v is in $N^+_i(u)$ and w is in $N^+_j(u)$ according to our assumption. Thus, v and w belong to the wedge partition $N^+_{i,j}$ since $N^+_{i,j}(u) = N^+_i(u) \cup N^+_j(u)$ and $N^+_{i,j} = \bigcup_{\forall u \in V} N^+_{i,j}(u)$. Therefore, a triangle $\Delta^\star_{uvw}$ appears only in a single $P_{i,j}$ composed of $N^+_{i,j}$ and $E^\star_{i,j}$. ■

For each pair of vertex partitions p_i and p_j for the vertex set V with $1 \le i \le j \le \rho$, we denote the union of an edge partition $E^\star_{i,j}$ and a wedge partition $N^+_{i,j}$ as a *partition pair* $P_{i,j}$. By Lemma 2, we can generate every triangle in an oriented graph $G^\star$ by investigating each partition pair without duplications. Thus, we developed the MapReduce algorithm *PBTE* based on Lemma 2. In Fig. 3, we present the pseudo-code of *PBTE* which consists of Partition round (line 1 in Fig. 3) and Enumeration round (line 2 in Fig. 3). We will present the details of Partition round and Enumeration round in Sect. 4.2.2 and Sect. 4.2.3, respectively.

4.2.2 Partition Round of PBTE

The pseudo-code of the Partition round of *PBTE* is presented

Function Partition.map$(key, (u, v))$
key: null,
(u, v): a directed edge in $E^\star$ of $G^\star$ of $P_{i,j}$
begin
Let $h()$ be a hash function returning an integer within $[1, \rho]$.
1. emit$(u; v)$;
2. $i = h(u), j = h(v)$;
3. **if** $(i \le j)$ **then** $E^\star_{i,j} = E^\star_{i,j} \cup \{(u, v)\}$
4. **else** $E^\star_{j,i} = E^\star_{j,i} \cup \{(u, v)\}$
end
Function Partition.reduce$(u; L)$
u: a vertex in V,
L: a value list for the out-neighbor set $N^+(u)$ of u
begin
1. **for** each vertex v in L **do**
2. $\quad i = h(v)$;
3. $\quad N^+_i(u) = N^+_i(u) \cup \{v\}$;
4. **for** $i \in [1, \rho]$ **do**
5. $\quad$ **for** $j \in [i, \rho]$ **do**
6. $\quad\quad$ **if** $N^+_i(u) \ne \emptyset$ and $N^+_j(u) \ne \emptyset$ **then**
7. $\quad\quad\quad N^+_{i,j}(u) = N^+_i(u) \cup N^+_j(u)$;
8. $\quad\quad\quad$ sortOutNegibors$(N^+_{i,j}(u))$;
9. $\quad\quad\quad$ store a list $\langle u, N^+_{i,j}(u)\rangle$ to $N^+_{i,j}$ of $P_{i,j}$;
end

Fig. 4 Partition round of *PBTE*

in Fig. 4. In the Partition round of *PBTE*, we generate every partition pair $P_{i,j}$ each of which consists of a wedge partition $N^+_{i,j}$ and an edge partition $E^\star_{i,j}$ where $1 \le i \le j \le \rho$ based on Definitions 3, 4 and 5.

In the map phase of the Partition round, each map function takes a directed edge (u, v) and emits the key-value pair $(u; v)$ (line 1 of Partition.map in Fig. 4). Let $h(u)$ be a pairwise independent hash function which returns an integer number within $[1, \rho]$ such that the hash value i of a vertex u indicates that u belongs to the vertex partition p_i for the vertex set V. To find the corresponding edge partition for each directed edge (u, v), the map function taking (u, v) applies the hash function $h()$ to each vertex of (u, v) (line 2 of Partition.map). Let $h(u)$ be i and $h(v)$ be j, respectively. Then, by Definition 4, (u, v) belongs to $E^\star_{i,j}$ if $i \le j$. Otherwise, (u, v) belongs to $E^\star_{j,i}$ (lines 3-4 of Partition.map).

During the shuffle phase, all values having the same key are grouped. Thus, each reduce function takes a vertex u as a key with its out-neighbor set $N^+(u)$ as the value list L. To generate every wedge partition $N^+_{i,j}$, the reduce function for a vertex u next splits L representing $N^+(u)$ into $N^+_i(u)$s with $1 \le i \le \rho$ (lines 1-3 of Partition.reduce) and calculate every $N^+_{i,j}(u)$ with $1 \le i \le j \le \rho$ since the wedge partition $N^+_{i,j}$ consists of $N^+_{i,j}(u)(= N^+_i(u) \cup N^+_j(u))$s of every vertex u by Definition 5 (lines 4-9 of Partition.reduce). The out-neighbor set $N^+_i(u)$ of a vertex u is the subset of the out-neighbor set $N^+(u)$ whose elements belong to p_i. At line 2 of Partition.reduce, we already calculate the partition id of each vertex v in L by using the hash function $h()$. Thus, the reduce function records each vertex $v \in L$ to $N^+_i(u)$ where $h(v)$ is i (line 3 of Partition.reduce). Then, for every pair of $N^+_i(u)$ and $N^+_j(u)$ with $1 \le i \le j \le \rho$, the reduce function generates $N^+_{i,j}(u)$ by merging $N^+_i(u)$ and $N^+_j(u)$ (lines 4-7 of

Function Enumeration.map$((i, j)$, value)
(i, j): the id of a partition pair $P_{i,j}$,
value: null
begin
1. **for** each $N^+_{i,j}(u)$ in $N^+_{i,j}$ **do**
2. emit((i,j); $\langle u, N^+_{i,j}(u)\rangle$);

end
Function Enumeration.reduce$((i, j); L)$
(i, j): the id of a partition pair $P_{i,j}$,
L: a value list for the wedge partition $N^+_{i,j}$ of $P_{i,j}$
begin
1. sortEdgeSet($E^\star_{i,j}$);
2. **for** each $N^+_{i,j}(u)$ in L **do**
3. **for** each pair of v and w in $N^+_{i,j}(u)$ with $id(v) < id(w)$ **do**
4. **if** ($h(v) = i$ and $h(w) = j$) or ($h(v) = j$ and $h(w) = i$) **then**
5. **if** binarySearch((v, w), $E^\star_{i,j}$) **then**
6. output($\Delta^\star_{uvw}$);

end

Fig. 5 Enumeration round of *PBTE*

Partition.reduce). If $N^+_i(u)$ or $N^+_j(u)$ is an empty set, it means there is no wedge $\omega^\star_{uvw}$ such that one of its end-nodes belongs to p_i and the other is in p_j. Thus, the triangles whose cone vertices are u do appear in $P_{i,j}$. From this observation, we build $N^+_{i,j}(u)$ only when both $N^+_i(u)$ and $N^+_j(u)$ are not empty (line 6 of Partition.reduce). Since we have to evaluate every vertex pair in $N^+_{i,j}(u)$ at the next round to form a triangle, we sort the vertices in $N^+_{i,j}(u)$ in increasing order by their id to access every pair of vertices in $N^+_{i,j}(u)$ without redundancy (line 8 of Partition.reduce). Then, in the wedge partition $N^+_{i,j}$, the reduce function stores $N^+_{i,j}(u)$ with u as a list to denote that u is the cone vertex for every pair of vertices in $N^+_{i,j}(u)$ (line 9 of Partition.reduce).

4.2.3 Enumeration Round of PBTE

After the Partition round of *PBTE*, the edge partition $E^\star_{i,j}$ and the wedge partition $N^+_{i,j}$ of every partition pair $P_{i,j}$ with $1 \le i \le j \le \rho$ are generated. Thus, at the Enumeration round of *PBTE*, we generate the triangles appearing in each $P_{i,j}$ based on Lemma 2.

The pseudo-code of the Enumeration round is presented in Fig. 5. In the map phase of the Enumeration round, each map function takes the id of a partition pair $P_{i,j}$ as a key. Then, for each $N^+_{i,j}(u)$ kept in $N^+_{i,j}$ stored at the Partition round, the map function takes $N^+_{i,j}(u)$ and then emits the key-value pair $((i, j); \langle u, N^+_{i,j}(u)\rangle)$ where (i, j) denotes the id of the partition pair $P_{i,j}$ (lines 1-2 of Enumeration.map).

During the shuffle phase, all values having the same key are grouped. After that, in the reduce phase of the Enumeration round, $\rho \cdot (\rho + 1)/2$ reduce functions each of which handles a single partition pair $P_{i,j}$ are invoked since there are $\rho + \binom{\rho}{2} = \rho \cdot (\rho + 1)/2$ partition pairs where ρ and $\binom{\rho}{2}$ are for $P_{i,j}$ when $i = j$ and $i < j$, respectively.

At the reduce phase of the Enumeration round, for each pair of vertices v and w in $N^+_{i,j}(u)$ of $N^+_{i,j}$ where $id(v) < id(w)$, if v and w are adjacent to each other, a triangle $\Delta^\star_{uvw}$ is formed. Thus, we have to find the edge (v, w) in $E^\star_{i,j}$ efficiently. To do so, we sort all directed edges $e \in E^\star_{i,j}$ (line 1 of Enumeration.reduce) as follows: Since the direction of a directed edge is irrelevant to check whether a pair of vertices are adjacent, we first transform each directed edge $e = (v, w)$ in $E^\star_{i,j}$ into the edge e' such that $e' = e$ if $id(v) < id(w)$, otherwise $e' = (w, v)$. Thus, in the rest of this section, we regard that every edge $e = (v, w)$ in $E^\star_{i,j}$ satisfies that $id(v) < id(w)$. Then, all edges $e = (v, w)$ in $E^\star_{i,j}$ are sorted in increasing order of $id(v)$ and, in the case of tie, we sort them in increasing order of $id(w)$. For instance, given three directed edges $(1, 6)$, $(1, 4)$ and $(2, 3)$ in $E^\star_{i,j}$, the sorting result becomes $(1, 4)$, $(1, 6)$ and $(2, 3)$.

To form a triangle in $P_{i,j}$, an end-node of a wedge belongs to the vertex partition p_i and the other is in p_j. However, since $N^+_{i,j}(u)$ of $N^+_{i,j}$ is the union of $N^+_i(u)$ and $N^+_j(u)$, some pairs of vertices in $N^+_{i,j}(u)$ are included in a single vertex partition p_i or p_j. Thus, for each pair of vertices v and w in $N^+_{i,j}(u)$ where $id(v) < id(w)$, we first check whether v and w belong to the vertex partitions p_i and p_j, respectively, or p_j and p_i, respectively (lines 2-4 of Enumeration.reduce). Then, if two vertices v and w satisfy the condition, since all edges in $E^\star_{i,j}$ are sorted, we can find the edge for each vertex pair v and w in $N^+_{i,j}(u)$ by performing the binary search with (v, w) (line 5 of Enumeration.reduce). Finally, a triangle $\Delta^\star_{uvw}$ whose cone vertex is u is output when v and w are adjacent (line 6 of Enumeration.reduce).

The following example illustrates the behavior of *PBTE* algorithm.

Example 2: Reconsider the oriented graph $G^\star$ shown in Fig. 1 (b). At the Partition round of PBTE, each map function taking a directed edge (u, v) emits a key-value pair $(u; v)$ as well as keeps (u, v) to the corresponding edge partition $E^\star_{i,j}$ where $h(u) = i$ (or j) and $h(v) = j$ (or i). Let us assume that the number of vertex partitions ρ is 2 and a hash function is $h(u) = (id(u) \bmod \rho) + 1$. Then, the map function taking a directed edge $(2, 6)$ keeps $(2, 6)$ to the edge partition $E^\star_{1,1}$. The other map functions keep 10 directed edges to their edge partitions as shown in Fig. 6 (a) and (b). The key-value pairs emitted by all map functions are grouped by the same keys during the shuffle phase as shown in Fig. 6 (c). Thus, each reduce function taking a key u and $N^+_i(u)$ as a value list L splits L into $N^+_i(u)$s, merges each pair of $N^+_i(u)$ and $N^+_j(u)$ into $N^+_{i,j}(u)$, sorts the vertices in $N^+_{i,j}(u)$ in increasing order by their id and stores each sorted $N^+_{i,j}(u)$ into $N^+_{i,j}$ of the partition pair $P_{i,j}$. For instance, the reduce function taking a key 2 splits $L = \{6, 3, 7\}$ into $N^+_1(2) = \{6\}$ and $N^+_2(2) = \{3, 7\}$ and generates $N^+_{1,2}(2) = \{3, 6, 7\}$ and $N^+_{2,2}(2) = \{3, 7\}$ by merging $N^+_1(2)$ and $N^+_2(2)$. Then, $N^+_{1,2}(2)$ and $N^+_{2,2}(2)$ are stored into $N^+_{1,2}$ and $N^+_{2,2}$, respectively. However, the reduce function taking a key 5 with $L = \{3\}$ does not generate $N^+_{1,2}(5)$ since $N^+_1(5) = \emptyset$. The other reduce functions with 1, 4 and 7 store $N^+_{1,2}(4)$, $N^+_{1,2}(1)$, $N^+_{1,2}(7)$, $N^+_{2,2}(2)$ and $N^+_{2,2}(4)$, respectively, as shown in Fig. 6 (d).

At the Enumeration round of PBTE, each map function takes the id (i, j) of a partition pair $P_{i,j}$ and reads $N^+_{i,j}$. Then, the map function emits the key-value pair $((i, j); \langle u, N^+_{i,j}(u)\rangle)$

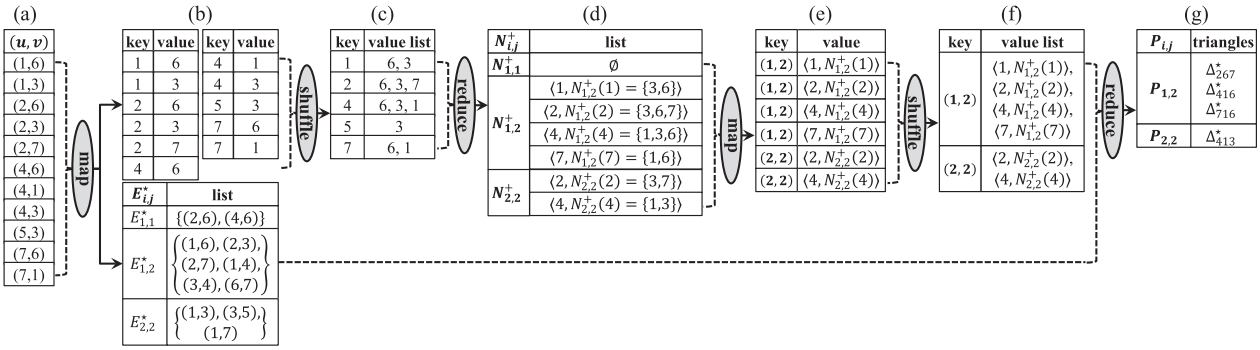


Fig. 6 The behavior of *PBTE* algorithm

for every $N^+_{i,j}(u)$ in $N^+_{i,j}$. For instance, a map function taking $(1,2)$ reads $N^+_{1,2}$. Then, the map function emits $((1,2);\langle 1, N^+_{1,2}(1)\rangle)$, $((1,2);\langle 2, N^+_{1,2}(2)\rangle)$, $((1,2);\langle 4, N^+_{1,2}(4)\rangle)$ and $((1,2);\langle 7, N^+_{1,2}(7)\rangle)$. The other map function with $(2,2)$ emits $((2,2);\langle 2, N^+_{2,2}(2)\rangle)$ and $((2,2);\langle 4, N^+_{2,2}(4)\rangle)$ as shown in Fig. 6 (e). The key-value pairs are grouped by the same keys during the shuffle phase as shown in Fig. 6 (f). Thus, for a partition pair $P_{i,j}$, each reduce function outputs the triangles appearing in $P_{i,j}$. For instance, in Fig. 6 (g), a reduce function for the partition pair $P_{2,2}$ outputs a triangle $\Delta^\star_{413}$ since the vertex pair 1 and 3 in $N^+_{2,2}(4)$ appears in $E^\star_{2,2}$ as an edge $(1,3)$. Similarly, the reduce function for $P_{1,2}$ outputs three triangles $\Delta^\star_{267}$, $\Delta^\star_{416}$ and $\Delta^\star_{716}$.

4.3 Enhanced PBTE Algorithm

In Sect. 4.2, we proposed a *PBTE* algorithm consisting of the Partition and Enumeration rounds. However, each reduce function at the Partition round of *PBTE* invoked with a vertex $u \in V$ and its out-neighbor set $N^+(u)$ stores the vertices in $N^+(u)$ into several partition pairs redundantly. For instance, given an oriented graph $G^\star$ in Fig. 1 (b), as shown in Fig. 6 (d), two vertices 3 and 7 in $N^+_2(2)$ are stored in both $N^+_{1,2}$ and $N^+_{2,2}$. Precisely, the cone vertex u is stored at most $\rho \cdot (\rho + 1)/2$ times since the number of out-neighbor sets $N^+_{i,j}(u)$ generated from $N^+(u)$ is at most $\rho + \binom{\rho}{2} = \rho \cdot (\rho+1)/2$. Furthermore, let a vertex $v \in N^+(u)$ belong to a vertex partition p_i where $1 \le i \le \rho$ (i.e., $v \in N^+_i(u)$). Then, since the out-neighbor set $N^+_{i,j}(u)$ is the union of $N^+_i(u)$ and $N^+_j(u)$ by Definition 5, $N^+_i(u)$ is the subset of $N^+_{a,i}(u)$ where $1 \le a \le i$ and $N^+_{i,b}(u)$ where $i < b \le \rho$. Thus, each vertex $v \in N^+(u)$ is stored at most ρ times. Therefore, each reduce function at the Partition round of *PBTE* stores at most $\rho \cdot (\rho+1)/2 + \rho \cdot |N^+(u)|$ vertices. These numerous duplicated out-neighbors on every partition pair results in the performance degradation at the reduce phase of the Partition round of *PBTE*.

To alleviate the drawback of *PBTE*, we devised an enhanced MapReduce algorithm, called Enhanced PBTE (abbreviated by *EPBTE*). In Fig. 7, we present the pseudo-code of *EPBTE*. Similar to *PBTE*, our enhanced algorithm *EPBTE* consists of two MapReduce rounds: the Partition$_{EN}$

Function EPBTE $(G^\star = (V, E^\star), \rho)$
$G^\star = (V, E^\star)$: an oriented graph of G,
ρ: the number of vertex partitions
begin
1. *Partitions* = RunMapReduce(Partition$_{EN}$, $E^\star$, ρ);
2. $\Delta(G)$ = RunMapReduce(Enumeration$_{EN}$, *Partitions*);
3. **return** $\Delta(G)$;
end

Fig. 7 The *EPBTE* algorithm

(line 1 in Fig. 7) and the Enumeration$_{EN}$ rounds (line 2 in Fig. 7). At Partition$_{EN}$ of *EPBTE*, every directed edge (v, w) in $E^\star$ is stored into an edge partition $E^\star_{i,j}$ like *PBTE*. However, in contrast to the Partition round of *PBTE* in which all out-neighbor sets $N^+_{i,j}(u)$ are stored in the wedge partition $N^+_{i,j}$, we store all out-neighbor sets $N^+_i(u)$ of all vertice u into a single file with $1 \le i \le \rho$.

In the Enumeration$_{EN}$ round of *EPBTE*, for each pair of $N^+_i(u)$ and $N^+_j(u)$ stored in the individual files, we check whether a pair of vertices ($v \in N^+_i(u)$, $w \in N^+_j(u)$) is in $E^\star_{i,j}$ to form a triangle $\Delta^\star_{uvw}$ appeared in a partition pair $P_{i,j}$ by Lemma 2. Before explaining the details of *EPBTE*, we define a set of $N^+_i(u)$s as follows:

Definition 6: Given ρ number of vertex partitions p_1, $p_2, \ldots, p_\rho$ for the vertex set V of an oriented graph $G^\star = (V, E^\star)$, an end-node partition on a vertex partition p_i, referred to as N^+_i, is defined as $N^+_i = \bigcup_{\forall u \in V} N^+_i(u)$.

The details of Partition$_{EN}$ round and Enumeration$_{EN}$ round will be presented in Sect. 4.3.1 and Sect. 4.3.2, respectively.

4.3.1 Partition$_{EN}$ Round of EPBTE

At the Partition$_{EN}$ round of *EPBTE*, based on Definitions 4 and 6, we generate every edge partition $E^\star_{i,j}$ and every end-node partition N^+_i where $1 \le i \le j \le \rho$. The pseudo-code of the Partition$_{EN}$ round is presented in Fig. 8. Since the behavior of Partition$_{EN}$.map is identical to that of Partition.map of *PBTE*, we omit the explanation for Partition$_{EN}$.map.

Each reduce function invoked with a vertex $u \in V$ taking a value list L records each vertex $v \in L$ to $N^+_i(u)$ where $h(v)$ is i (lines 1-3 of Partition$_{EN}$.reduce in Fig. 8).

Function Partition$_{EN}$.map(key, (u, v))
key: null, (u, v): a directed edge in $E^\star$ of $G^\star$ of $P_{i,j}$
begin
Let $h()$ be a hash function returning an integer within $[1, \rho]$.
1. emit(u; v);
2. $i = h(u)$, $j = h(v)$;
3. **if** $(i \le j)$ **then** $E^\star_{i,j} = E^\star_{i,j} \cup \{(u, v)\}$
4. **else** $E^\star_{j,i} = E^\star_{j,i} \cup \{(u, v)\}$

end
Function Partition$_{EN}$.reduce(u; L)
u: a vertex in V,
L: a value list for the out-neighbor set $N^+(u)$ of u
begin
1. **for** each vertex v in L **do**
2. $\quad i = h(v)$;
3. $\quad N^+_i(u) = N^+_i(u) \cup \{v\}$;
4. **for** $i \in [1, \rho]$ **do**
5. $\quad$ **if** $N^+_i(u) \neq \emptyset$ **then**
6. $\quad\quad$ store a list $\langle u, N^+_i(u)\rangle$ to N^+_i;

end

Fig. 8 Partition$_{EN}$ round of *EPBTE*

Then, the reduce function stores $N^+_i(u)$ with u as a list into an end-node partition N^+_i where $1 \le i \le \rho$ (lines 4-6 of Partition$_{EN}$.reduce). If $N^+_i(u)$ is an empty set, it means that there is no wedge $\omega^\star_{uvw}$ such that either of the end-nodes belongs to a vertex partition p_i. Thus, the triangles whose cone vertex u do not appear in $\bigcup_{1\le a\le i} P_{a,i}$ and $\bigcup_{i<b\le\rho} P_{i,b}$. Therefore, the reduce function does not store $N^+_i(u)$ if $|N^+_i(u)| = 0$.

Note that, in both algorithms *PBTE* and *EPBTE*, since each directed edge is stored only once in the proper edge partition $E^\star_{i,j}$, the total number of stored edges becomes $|E^\star|$. However, in contrast to the Partition round of *PBTE*, the reduce function of the Partition$_{EN}$ round of *EPBTE* stores each out-neighbor set $N^+_i(u)$ with the cone vertex u into the end-node partition N^+_i on the vertex partitions p_i by splitting $N^+(u)$. Since the number of out-neighbor sets $N^+_i(u)$ is at most ρ, the cone vertex u is stored at most ρ times. Furthermore, since $N^+(u) = \bigcup_{1\le i\le\rho} N^+_i(u)$, the reduce function stores $|N^+(u)|$ out-neighbors. Therefore, the number of the vertices ($=\rho + |N^+(u)|$) stored at the Partition$_{EN}$ round of *EPBTE* is significantly smaller than that ($=\rho \cdot (\rho + 1)/2 + \rho \cdot |N^+(u)|$) at the Partition round of *PBTE*.

4.3.2 Enumeration$_{EN}$ Round of EPBTE

At the Enumeration$_{EN}$ round of *EPBTE*, for every vertex u, we first combine $N^+_i(u) \in N^+_i$ and $N^+_j(u) \in N^+_j$ having the same cone vertex u in order to generate every wedge $\omega^\star_{uvw}$ where v in $N^+_i(u)$ and w in $N^+_j(u)$. And then, we check whether the end-nodes v and w of $\omega^\star_{uvw}$ are adjacent to generate a triangle $\Delta^\star_{uvw}$.

Given a pair of end-node partitions N^+_i and N^+_j, we have to look up every pair of $\langle u, N^+_i(u)\rangle$ and $\langle u, N^+_j(u)\rangle$ stored in N^+_i and N^+_j respectively. This process is identical to the equi-join operation with the candidate key as a join attribute since a cone vertex u appears only once in N^+_i and N^+_j respectively. In our implementation, we find every such pair in a sort-merge join fashion [9] since the nested loop join operation is too slow and the hash join operation consumes much memory to maintain a hash table. On the contrary of other join operations, the sort-merge join operation is relatively efficient. Furthermore, the sort-merge join operation requires the space for a pair of entries coming from two operands during join processing when the candidate key is a join attribute. This results in the minimum memory usage.

Function Enumeration$_{EN}$.map(i; $value$)
i: the id of a vertex partition p_i, $value$: null
begin
1. sort N^+_i in increasing order by the id of cone vertex u;
2. store N^+_i to HDFS;

end
Function Enumeration$_{EN}$.reduce((i, j); L)
(i, j): the id of a partition pair $P_{i,j}$, L: null
begin
1. sortEdgeSet($E^\star_{i,j}$);
2. **for** each pair of $N^+_i(u) \in N^+_i$ and $N^+_j(u) \in N^+_j$ **do**
3. $\quad$ **for** each pair of $v \in N^+_i(u)$ and $w \in N^+_j(u)$ **do**
4. $\quad\quad$ **if** $id(v) > id(w)$ **then** $(v', w') = (w, v)$;
5. $\quad\quad$ **else** $(v', w') = (v, w)$;
6. $\quad\quad$ **if** binarySearch((v', w'), $E^\star_{i,j}$) **then**
7. $\quad\quad\quad$ output($\Delta^\star_{uvw}$);

end

Fig. 9 Enumeration$_{EN}$ round of *EPBTE*

The pseudo-code of the Enumeration$_{EN}$ round is presented in Fig. 9. In the map phase, each map function is invoked with the id i for the vertex partition p_i. Then, the map function for p_i sorts N^+_i in increasing order by id of the cone vertex u and stores sorted N^+_i (lines 1-2 of Enumeration$_{EN}$.map) in order to perform a merge join at the reduce phase.

At the reduce phase, each reduce function invoked by the reducer with an integer pair (i, j) which represents the id of the partition pair $P_{i,j}$ first sorts the directed edges in $E^\star_{i,j}$ (line 1 of Enumeration$_{EN}$.reduce). Then, we find every pair of $N^+_i(u)$ and $N^+_j(u)$ having the same cone vertex u in N^+_i and N^+_j (line 2 of Enumeration$_{EN}$.reduce). As mentioned above, we find such pairs scanning N^+_i and N^+_j sequentially following the sort-merge join fashion. For each pair of vertices $v \in N^+_i(u)$ and $w \in N^+_j(u)$, we check whether v and w are adjacent by performing the binary search on $E^\star_{i,j}$ and generate a triangle $\Delta^\star_{uvw}$ when (v, w) are adjacent like *PBTE* (lines 3-7 of Enumeration$_{EN}$.reduce). To conduct binary search on $E^\star_{i,j}$, we swap ids of v and w if $id(v) > id(w)$ since every edge $e = (v, w)$ in $E^\star_{i,j}$ satisfies $id(v) < id(w)$ (lines 4-5 of Enumeration$_{EN}$.reduce).

The following example illustrates the behavior of *EPBTE* algorithm.

Example 3: Reconsider the oriented graph $G^\star$ shown in Fig. 1 (b). Since the behaviors of the map and shuffle phases of the Partition$_{EN}$ round of *EPBTE* is identical to those of the Partition round of *PBTE*, we omit the explanation to Figs. 10 (a), (b) and (c). Then, each reduce function taking a key u and a value list L splits L into $N^+_i(u)$s and stores $N^+_i(u)$s to an end-node partition N^+_i. Let ρ be 2 and a hash function be $h(u) = (id(u) \bmod \rho) + 1$, respectively. Then, as

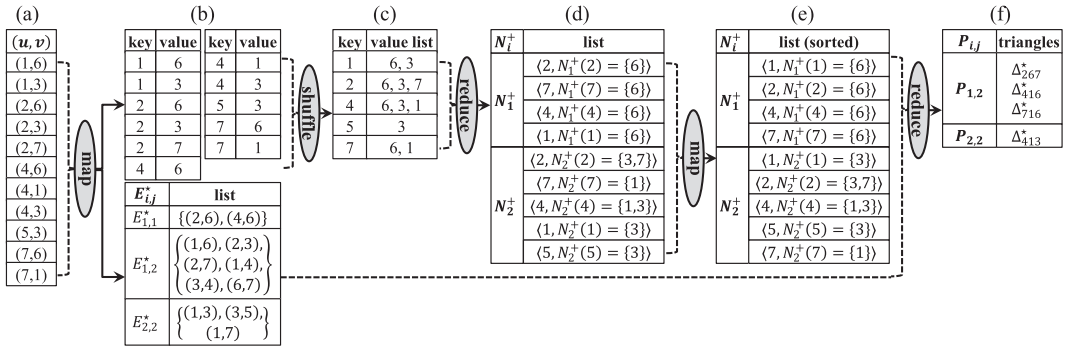


Fig. 10 The behavior of *EPBTE* algorithm

shown in Fig. 10 (d), the reduce function taking a key 2 splits $L = \{6, 3, 7\}$ into $N_1^+(2) = \{6\}$ and $N_2^+(2) = \{3, 7\}$ and then, stores $N_1^+(2)$ and $N_2^+(2)$ into N_1^+ and N_2^+, respectively. The other reduce functions with 1, 4, 5 and 7 also store $N_1^+(1)$, $N_1^+(4)$ and $N_1^+(7)$ into N_1^+ as well as $N_2^+(1)$, $N_2^+(4)$, $N_2^+(5)$ and $N_2^+(7)$ into N_2^+. In the Enumeration$_{EN}$ round, each map function invoked with a vertex partition p_i sorts N_i^+ in increasing order by id of cone vertex and stores the sorted N_i^+. For instance, as shown in Fig. 10 (e), the map function for p_1 sorts N_1^+ and stores N_1^+. After that, each reduce function for a partition pair $P_{i,j}$ outputs the triangles appearing the $P_{i,j}$. For instance, the reduce function for $P_{1,2}$ outputs three triangles $\Delta^\star_{267}$, $\Delta^\star_{416}$ and $\Delta^\star_{716}$ by using N_1^+, N_2^+ and $E^\star_{1,2}$ as well as the reduce function for $P_{1,1}$ outputs a triangle $\Delta^\star_{413}$ by using N_1^+ and $E^\star_{1,1}$ as shown in Fig. 10 (f).

4.4 Analysis of Our Parallel Algorithms

In this section, we provide how to obtain the number of vertex partitions ρ since the performances of *PBTE* and *EPBTE* are affected by ρ. Of particular, the performances at the Enumeration round and Enumeration$_{EN}$ round of both algorithms are mainly affected by ρ since each reduce function at both rounds is invoked with the id (i, j) for each partition pair $P_{i,j}$ where $1 \le i \le j \le \rho$. The number of reducers (and reduce functions) is equal to the number of partition pairs $\rho \cdot (\rho + 1)/2$. Thus, as the number of vertex partitions ρ increases, the performances of both algorithms will be degraded due to the large number of reducers. In contrast, when ρ is too small, the reduce phase at these rounds may not work since each reduce function has to take a large volume of an edge partition $E^\star_{i,j}$ resulting in out of memory. Therefore, we derive the min imum ρ with respect to the memory usage.

Note that the memory usage at the Partition round and Partition$_{EN}$ round of *PBTE* and *EPBTE*, respectively, is not affected by ρ since each map function at both rounds simply takes a directed edge as well as the reduce function takes the out-neighbor set $N^+(u)$ of a vertex u. Let d_M be the maximum degree of vertices in V. Then, the memory usage of each map and reduce functions at these rounds is at most d_M. However, since d_M is much smaller than the memory usage at the Enumeration and Enumeration$_{EN}$ rounds, we can ignore the memory usage at the Partition round and Partition$_{EN}$ rounds.

As described in Figs. 5, each map function of *PBTE*'s Enumeration round simply stores each $N^+_{i,j}(u)$ $(=N_i^+(u) \cup N_j^+(u))$ in $N^+_{i,j}$ with (i, j) one by one. The expected number of vertices in $N^+_{i,j}(u)$ is at most $2 \cdot d_M/\rho$ since $|N_i^+(u)|$ and $|N_j^+(u)|$ stochastically are less than d_M/ρ, respectively. In addition, each map function of *EPBTE*'s Enumeration$_{EN}$ round simply sorts N_i^+ in increasing order by the id of cone vertex. Thus, we can also ignore the memory requirement for each map function at the Enumeration and Enumeration$_{EN}$ rounds.

In the reduce function at the Enumeration round and Enumeration$_{EN}$ round, the space for loading $E^\star_{i,j}$ is required since each reduce function loads an edge partition $E^\star_{i,j}$ to perform sorting and binary search on it efficiently. Thus, in our implementation, the space for $E^\star_{i,j}$ is required. In addition, for each pair of vertices v and w in $N^+_{i,j}(u)$, a binary search on $E^\star_{i,j}$ is performed by the reduce function. Thus, at most $2 \cdot d_M/\rho$ space is required for $N^+_{i,j}(u)$. However, since $|N^+_{i,j}(u)|$ $(=2 \cdot d_M/\rho)$ is much smaller than $|E^\star_{i,j}|$, we only consider the size of $E^\star_{i,j}$ for the memory consumption of the reduce function at the Enumeration round and Enumeration$_{EN}$ round.

Lemma 3: Given an oriented graph $G^\star = (V, E^\star)$ and the number of vertex partitions ρ for the vertex set V, the expected number of directed edges in each edge partition $E^\star_{i,j}$ becomes $2 \cdot |E^\star|/(\rho \cdot (\rho + 1))$.

Proof. Each vertex $u \in V$ is partitioned by a hash function $h()$ which is randomly chosen from a pairwise independent family of functions. The pairwise independence of $h()$ guarantees that directed edges are evenly distributed over edge partitions $E^\star_{i,j}$ with $1 \le i \le j \le \rho$. Since the number of edge partitions is $\rho + \binom{\rho}{2} = (\rho \cdot (\rho + 1))/2$ where ρ and $\binom{\rho}{2}$ are for $P_{i,j}$ when $i = j$ and $i < j$ respectively, the expected number of directed edges in each edge partition $E^\star_{i,j}$ becomes $|E^\star|/((\rho \cdot (\rho + 1))/2) = 2 \cdot |E^\star|/(\rho \cdot (\rho + 1))$. ■

Let M be the number of edges kept in the available memory of each machine participated in a MapReduce

framework. Then, we calculate the proper value of ρ for the reduce phase at the Enumeration round of *PBTE* and the Enumeration$_{EN}$ round of *EPBTE* as follows:

Lemma 4: Given an oriented graph $G^\star = (V, E^\star)$ and the number of edges M kept in available memory space of each machine, the number of vertex partitions ρ for the reduce phase at the Enumeration round and the Enumeration$_{EN}$ round is $\left\lceil \sqrt{\frac{2\cdot|E^\star|}{M} + \frac{1}{4}} - \frac{1}{2} \right\rceil$.

Proof. By Lemma 3, the expected number of directed edges in each edge partition $E^\star_{i,j}$ becomes $2\cdot|E^\star|/(\rho\cdot(\rho+1))$. Then, each reduce function at the Enumeration round of *PBTE* and Enumeration$_{EN}$ round of *EPBTE* has to keep $2\cdot|E^\star|/(\rho\cdot(\rho+1))$ edges for $E^\star_{i,j}$. Thus, under the restriction of the number of edges M kept in memory space, we get

$$\frac{2\cdot|E^\star|}{\rho\cdot(\rho+1)} \le M.$$

Then, we have following inequalities.

$$\frac{2\cdot|E^\star|}{M} \le \rho^2 + \rho$$
$$\frac{2\cdot|E^\star|}{M} \le \left(\rho + \frac{1}{2}\right)^2 - \frac{1}{4}$$
$$\frac{2\cdot|E^\star|}{M} + \frac{1}{4} \le \left(\rho + \frac{1}{2}\right)^2$$
$$\sqrt{\frac{2\cdot|E^\star|}{M} + \frac{1}{4}} \le \rho + \frac{1}{2}$$
$$\sqrt{\frac{2\cdot|E^\star|}{M} + \frac{1}{4}} - \frac{1}{2} \le \rho$$

Therefore, the minimum integer number of ρ becomes $\left\lceil \sqrt{\frac{2\cdot|E^\star|}{M} + \frac{1}{4}} - \frac{1}{2} \right\rceil$. ■

Since the memory usage of each reduce function at the Enumeration round of *PBTE* and the Enumeration$_{EN}$ round of *EPBTE* is dominant, the ρ for *PBTE* and *EPBTE* becomes $\left\lceil \sqrt{\frac{2\cdot|E^\star|}{M} + \frac{1}{4}} - \frac{1}{2} \right\rceil$.

5. Experiments

5.1 Experimental Environments

To show the efficiency and scalability of our algorithms *PBTE* and *EPBTE*, we empirically evaluated the performances of our proposed algorithms by comparing with the previous parallel algorithms [11], [12] including the state-of-the-art algorithm *PTE*.

All experiments were performed on the cluster composed of one master node and 30 slave nodes. The master node is equipped with an Intel Xeon E3-1220 V2 CPU and 16 Gbyte of memory size. Meanwhile, each slave node has an Intel Core i5-3470 CPU, 4 Gbyte of memory size and 1 Tbyte of disk size. Every machine is running on Linux (Ubuntu 12.04.4). In our experiments, the number of machines, denoted as m, is varied from 10 to 30 and the default number of machines is 30. We used Hadoop 2.7.3 for the MapReduce framework implementation obtained from [1]. The implementations of all algorithms presented in Table 1 were compiled by Javac 1.8. Among the implemented algorithms, we used the source codes of *CTTP* [12] and *PTE* [11] obtained from the web pages provided by the authors. In our experiments, we used the base version of PTE, *PTE*$_{BASE}$, among three versions of *PTE*.

Table 1 Implemented triangle enumeration algorithms

Algo.	Description
CTTP	Multi-rounds parallel algorithm proposed in [12].
PTE	State-of-the-art algorithm proposed in [11].
BTE	Our basic parallel algorithm.
PBTE	Our partition based parallel algorithm.
EPBTE	Our enhanced parallel algorithm.

Table 2 Data sets

Data set	$\lvert V\rvert$	$\lvert E^\star\rvert$	$\lvert\Delta(G^\star)\rvert$	Link density *(d)*	Size
LiveJournal	4.8M	69M	286M	$3.0\cdot10^{-6}$	1 G
Orkut	3.0M	111M	628M	$1.2\cdot10^{-5}$	1.7 G
Brain	0.7M	171M	24B	$3.5\cdot10^{-4}$	2.4 G
BrainLarge	0.7M	267M	42B	$5.4\cdot10^{-4}$	3.8 G

Data Sets. To evaluate the proposed algorithms, we used the real-life data sets: *LiveJournal*, *Orkut*, *Brain* and *BrainLarge*. The *LiveJournal* data set is the graph for a free on-line community [16] and the *Orkut* data set is the graph for a free on-line social network [20]. Both data sets *Brain* and *BrainLarge* are the networks for the collections of brain networks [14]. The statistics of data sets are summarized in Table 2. In Table 2, the link density d of each data set is calculated using the equation $|E^\star|/(|V|\cdot(|V|-1))$. As shown in Table 2, *LiveJournal* and *Orkut* contain small number of edges compared to number of vertices whereas *Brain* contains the largest number of edges. In other words, the link densities of *LiveJournal* and *Orkut* are much smaller than that of *Brain*.

The default data set used in our experiments is *Brain* since it contains moderate number of edges. By using algorithm proposed in [4], we preprocessed each data set to obtain the corresponding oriented graph. In our experiments, we ran all algorithms three times and report the average running time of each algorithm. In addition, we do not report the running times which exceed 3 hours.

5.2 Experimental Results

In this section, we present our experimental results on the real-life data sets.

Varying the number of vertex partitions ρ: We plotted the running times of *PBTE* and *EPBTE* for varying the number of vertex partitions ρ from 6 to 20 on the default data set *Brain* in Fig. 11.

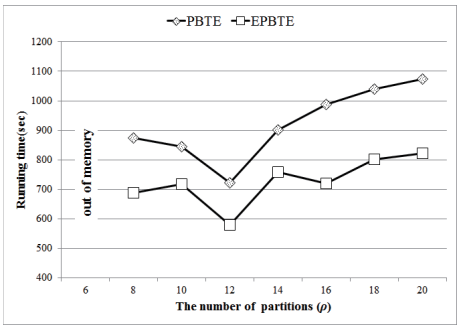


Fig. 11 The running times of *PBTE* and *EPBTE* varying the number of vertex partitions ρ on *Brain* data set

Table 3 Comparison between the actual ρ for our proposed algorithms and the value of ρ calculated by Lemma 4 on every data set

Data sets	ρ of PBTE	ρ of EPBTE	Calculated ρ
LiveJournal	8	8	7
Orkut	8	9	9
Brain	12	12	11
BrainLarge	15	15	14

When $\rho = 6$, the out of memory violation occurs in *PBTE* and *EPBTE* due to the large size of $E^{\star}_{i,j}$ processed at the Enumeration round and Enumeration$_{EN}$ round. At the Enumeration round of *PBTE* and the Enumeration$_{EN}$ round of *EPBTE*, as the number of vertex partitions ρ increases, the size of each edge partition $E^{\star}_{i,j}$ handled by each reduce function decreases. Thus, the running times of *PBTE* and *EPBTE* become improved as ρ increases from 8 to 12. However, when ρ becomes greater than 12, the performances of both algorithms become degraded due to the large number of the reduce functions at the Enumeration round and Enumeration$_{EN}$ round. Thus, when the number of vertex partitions ρ is 12, the performances of *PBTE* and *EPBTE* are the best on the default data set *Brain*.

Although we also evaluated the running times of *PBTE* and *EPBTE* on the other data sets varying ρ, we do not show the results of the other data sets because the patterns are very similar. Instead, we report the actual ρ of *PBTE* and *EPBTE* showing the best performance on each data set in our experiment and the calculated ρ obtained by Lemma 4 in Table 3. To compute ρ by Lemma 4, we set the available memory space to 350Mbyte since, although the maximum heap memory size provided by the java virtual machine running on each slave node is about 500Mbyte, we have to preserve some space for keeping environment variables.

In this experiment, to compute ρ by Lemma 4, we set M be 2.91M obtained by letting the size of an edge be 120 byte which is the double of vertex size (60 byte) used in the source code of *PTE*. For example, for a data set *Orkut* including 111M directed edges, the ρ becomes $\left\lceil \sqrt{\frac{2\cdot 111\cdot 1024\cdot 1024}{2.91\cdot 1024\cdot 1024} + \frac{1}{4}} - \frac{1}{2} \right\rceil = 9$. As reported in Table 3, for each data set, the actual ρ of each algorithm is slightly different from the calculated ρ. This result indicates that the value of ρ obtained from Lemma 4 is sufficient to use as the number of vertex partitions.

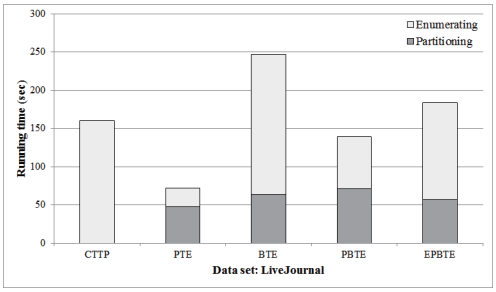
Varying the data sets: We next evaluated the execution time of each algorithm on all data sets. In this experiment, for *PBTE* and *EPBTE*, we used the calculated ρ as the number of vertex partitions. In Figs. 12(a)-(d), we plot the execution time of each algorithm except *CTTP* by splitting the partitioning round time and the enumerating round time on each data set since *CTTP* does not distinguish between the partitioning round and enumerating round.

As shown in Fig. 12(a)-(d), the execution time of each algorithm increases with increasing the size of data set. Our basic algorithm *BTE* shows the worst performance since each reduce function at the second round is invoked with each edge or the end-nodes of each wedge. In particular, the running times of *BTE* on two data sets *Brain* and *BrainLarge* exceed 3 hours since both data sets contain a large number of edges. In addition, the performance of *CTTP* is worse than those of *PTE*, *PBTE* and *EPBTE* since *CTTP* iterates many MapReduce rounds.

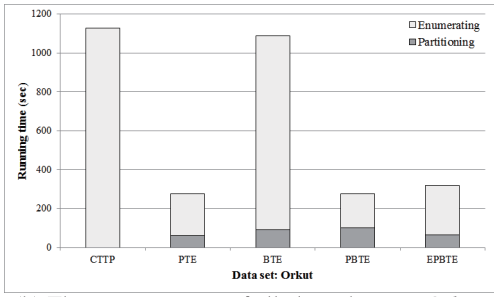
As shown in Figs. 12(a)-(d), on each data set, the partitioning rounds of *PTE*, *PBTE* and *EPBTE* show the similar performance. However, the execution times of all algorithms' enumerating rounds are different. This means that an effective technique for identifying triangles in each partition is required to solve the triangle enumeration problem efficiently.

At the enumerating rounds, our algorithms *PBTE* and *EPBTE* sort each edge partition $E^{\star}_{i,j}$ and perform a binary search with $(v, w) \in N^{+}_{i,j}(u)$ on $E^{\star}_{i,j}$ whereas *PTE* simply evaluates whether the end-nodes of every edge in each partition have a common neighbor. Thus, when the number of edges is small compared to the number of vertices (i.e., the link density is small), *PTE* shows the better performance than our proposed algorithms *PBTE* and *EPBTE*. Therefore, for *LiveJournal* and *Orkut* data sets, the execution time *PTE* is smaller than those of *PBTE* and *EPBTE* as shown in Fig. 12 (a) and (b) since the overhead for sorting $E^{\star}_{i,j}$ is larger than its gain. Meanwhile, when the link density becomes large (i.e., *Brain* and *BrainLarge* data sets), the gain of binary search compensates the overhead for sorting $E^{\star}_{i,j}$ since the number of vertex pair in $N^{+}_{i,j}(u)$ also becomes large. Thus, the performances of our proposed algorithms are better than that of *PTE*.

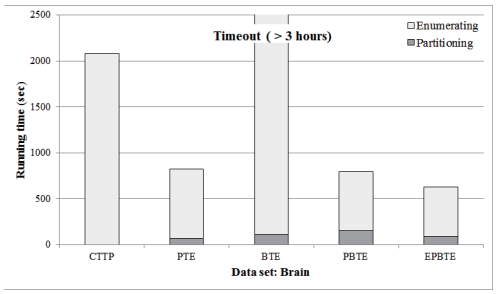
Since both *LiveJournal* and *Orkut* data sets contain small number of edges, the number of wedge partitions $N^{+}_{i,j}$ derived from $|E^{\star}|$ is small and the size of each wedge partition is also small. In this case, *PBTE* storing each wedge partition $N^{+}_{i,j}$ is more efficient than *EPBTE* storing the end-node partitions N^{+}_{i} and N^{+}_{j} separately and performing join them as shown in Figs. 12 (a) and (b). Meanwhile, when the number of edges becomes large (i.e., *Brain* and *BrainLarge* data sets), the number of wedge partitions also is large and *PBTE* stores numerous duplicated out-neighbors into several wedge partitions as mentioned in Sect. 4.3. Thus, the performance of *EPBTE* becomes better than that of *PBTE*



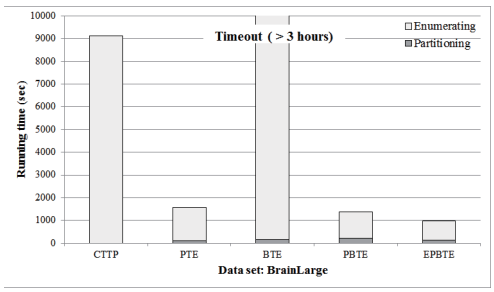
(a) The running times of all algorithms on *LiveJournal*



(b) The running times of all algorithms on *Orkut*



(c) The running times of all algorithms on *Brain*



(d) The running times of all algorithms on *BrainLarge*

Fig. 12 The running times of all algorithms on each data set

Table 4 The sorting overhead over execution time of *EPBTE*

Data sets	Sorting (sec)	Execution (sec)	Ratio
LiveJournal	14.32	184.56	7.76%
Orkut	22.11	319.68	6.92%
Brain	36.16	628.43	5.75%
BrainLarge	38.25	984.59	3.88%

as shown in Figs. 12 (c) and (d).

In order to show the sorting overhead over execution time of *EPBTE*, we measured the cumulative time of sorting $E^{\star}_{i,j}$ on each machine since several reduce functions invoking the procedure sortEdgeSet($E^{\star}_{i,j}$) are executed on each machine. In Table 4, we reported the maximum among the cumulative sorting times of all machines as well as the execution time and the ratio of sorting time to execution time. As reported in Table 4, the sorting times of the data sets with the high link density such as *Brain* and *BrainLarge* are greater than those of the data set with low link density (i.e., *LiveJournal* and *Orkut*) since the numbers of edges in *Brain* and *BrainLarge* is larger than those in *LiveJournal* and *Orkut*. However, the ratio of sorting time to execution time tends to decrease as the link density becomes large. This result confirms the above analysis such that the gain of binary search compensates the overhead for sorting $E^{\star}_{i,j}$ when the link density becomes large.

Varying the number of machines m**:** In order to evaluate four algorithms *CTTP*, *PTE*, *PBTE* and *EPBTE* with regard to machine scalability, we run the algorithms on the default data set *Brain* varying the number of machines m from 10 to 30. As shown in Fig. 13, the running time of each algorithm decreases as the number of machines m increases. Especially, our proposed algorithm *EPBTE* shows the best scalability regardless of the number of machines m.

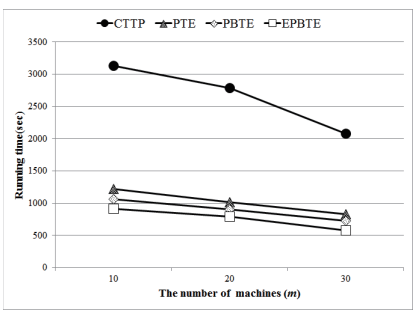


Fig. 13 The running times of *CTTP*, *PTE*, *PBTE* and *EPBTE* varying the number of machines m

6. Conclusion

In this paper, we proposed the parallel algorithms *PBTE* and *EPBTE* for triangle enumeration in the massive graph. Both *PBTE* and *EPBTE* split the graph into several vertex partitions and stores the directed edges and out-neighbor sets for the partition pairs separately to generate the triangles appearing in each partition pair. Especially, *EPBTE* does not store the duplicated out-neighbors in contrast to *PBTE* to improve the performance to generate triangles. Furthermore, we proved that the proper number of vertex partitions ρ with respect to the maximum number of edges M kept in available memory space of each machine becomes $\left\lceil \sqrt{\frac{2\cdot|E^{\star}|}{M} + \frac{1}{4}} - \frac{1}{2} \right\rceil$ for our proposed algorithms. Then, the minimum ρ is sufficient to use as the number of vertex partitions. Our experiments confirm the effectiveness and scalability of our proposed algorithms.

References

[1] Apache, Apache hadoop, http://hadoop.apache.org, 2010.
[2] A. Azad, A. Buluç, and J. Gilbert, "Parallel triangle counting and enumeration using matrix algebra," 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp.804–811, IEEE, 2015.
[3] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," Computer networks, vol.33, no.1, pp.309–320, 2000.
[4] J. Cohen, "Graph twiddling in a mapreduce world," Computing in Science & Engineering, vol.11, no.4, pp.29–41, 2009.
[5] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol.51, no.1, pp.107–113, 2008.
[6] M. Girvan and M.E.J. Newman, "Community structure in social and biological networks," Proceedings of the national academy of sciences, vol.99, no.12, pp.7821–7826, 2002.
[7] X. Hu, Y. Tao, and C.-W. Chung, "Massive graph triangulation," Proceedings of the 2013 ACM SIGMOD international conference on Management of data, pp.325–336, ACM, 2013.
[8] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," Proceedings of the 19th international conference on World wide web, pp.591–600, ACM, 2010.
[9] P. Mishra and M.H. Eich, "Join processing in relational databases," ACM Computing Surveys (CSUR), vol.24, no.1, pp.63–113, 1992.
[10] H.-M. Park and C.-W. Chung, "An efficient mapreduce algorithm for counting triangles in a very large graph," Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pp.539–548, ACM, 2013.
[11] H.-M. Park, S.-H. Myaeng, and U. Kang, "Pte: Enumerating trillion triangles on distributed systems," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.1115–1124, ACM, 2016.
[12] H.-M. Park, F. Silvestri, U. Kang, and R. Pagh, "Mapreduce triangle enumeration with guarantees," Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp.1739–1748, ACM, 2014.
[13] M.K. Rasel, Y. Han, J. Kim, K. Park, N.A. Tu, and Y.-K. Lee, "Itri: Index-based triangle listing in massive graphs," Information Sciences, vol.336, pp.1–20, 2016.
[14] R.A. Rossi and N.K. Ahmed, "The network data repository with interactive graph analytics and visualization," Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015.
[15] T. Schank, "Algorithmic aspects of triangle-based network analysis," Ph.D. thesis, University of Karlsruhe, 2007.
[16] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," Proceedings of the 20th international conference on World wide web, pp.607–614, ACM, 2011.
[17] J. Wang and J. Cheng, "Truss decomposition in massive networks," Proceedings of the VLDB Endowment, vol.5, no.9, pp.812–823, 2012.
[18] N. Wang, J. Zhang, K.-L. Tan, and A.K. Tung, "On triangulation-based dense neighborhood graph discovery," Proceedings of the VLDB Endowment, vol.4, no.2, pp.58–68, 2010.
[19] D.J. Watts and S.H. Strogatz, "Collective dynamics of 'small-world' networks," nature, vol.393, no.6684, pp.440–442, 1998.
[20] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," Knowledge and Information Systems, vol.42, no.1, pp.181–213, 2015.
[21] H. Zhang, Y. Zhu, L. Qin, H. Cheng, and J.X. Yu, "Efficient triangle listing for billion-scale graphs," 2016 IEEE International Conference on Big Data (Big Data), pp.813–822, IEEE, 2016.

Hongyeon Kim was born in 1986. He is currently pursuing the Ph.D. degree in computer science and engineering from Korea University of Technology and Education, Republic of Korea. He received the M.S. degree from the School of Computer Science and Engineering, Korea University of Technology and Education, in 2013. His main research interests include indexing, triangle enumeration, Big data, and MapReduce.

Jun-Ki Min was born in 1972. He received the BS degree in computer science from SoongSil University, in 1995, and the MS and PhD degrees in computer science and electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), in 1998 and 2002, respectively. He is currently a professor with the Korea University of Technology and Education (KoreaTech), Korea. Before that, he was a member of senior researcher in the Electronics and Telecommunications Research Institute (ETRI). He also served as a PC member for PAKDD, DASFAA, VLDB, ICDE, and WWW conferences. He has written and published several articles in international journals and conference proceedings. His current research interests include query processing and optimization, sensor data management, and stream data processing, XML, and parallel query processing.