Enumerating Trillion Subgraphs On Distributed Systems

HA-MYUNG PARK, KAIST
FRANCESCO SILVESTRI, University of Padova
RASMUS PAGH, IT University of Copenhagen
CHIN-WAN CHUNG, Chongqing University of Technology and KAIST
SUNG-HYON MYAENG, KAIST
U KANG, Seoul National University

How can we find patterns from an enormous graph with billions of vertices and edges? The subgraph enumeration, which is to find patterns from a graph, is an important task for graph data analysis with many applications, including analyzing the social network evolution, measuring the significance of motifs in biological networks, observing the dynamics of Internet, and so on. Especially, the triangle enumeration, a special case of the subgraph enumeration, where the pattern is a triangle, has many applications such as identifying suspicious users in social networks, detecting web spams, and finding communities. However, recent networks are so large that most of the previous algorithms fail to process them. Recently, several MapReduce algorithms have been proposed to address such large networks; however, they suffer from the massive shuffled data resulting in a very long processing time.

In this article, we propose scalable methods for enumerating trillion subgraphs on distributed systems. We first propose PTE (*Pre-partitioned Triangle Enumeration*), a new distributed algorithm for enumerating triangles in enormous graphs by resolving the structural inefficiency of the previous MapReduce algorithms. PTE enumerates trillions of triangles in a billion scale graph by decreasing three factors: the amount of shuffled data, total work, and network read. We also propose PSE (*Pre-partitioned Subgraph Enumeration*), a generalized version of PTE for enumerating subgraphs that match an arbitrary query graph. Experimental results show that PTE provides 79 times faster performance than recent distributed algorithms on real-world graphs, and succeeds in enumerating more than 3 *trillion* triangles on the ClueWeb12 graph with 6.3 billion vertices and 72 billion edges. Furthermore, PSE successfully enumerates 265 trillion clique subgraphs with 4 vertices from a subdomain hyperlink network, showing 47 times faster performance than the state of the art distributed subgraph enumeration algorithm.

CCS Concepts: • **Information systems** → Data mining; • **Theory of computation** → **Parallel algorithms**; **Distributed algorithms**; *MapReduce algorithms*; *Graph algorithms analysis*;

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. R0190-15-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development). The ICT at Seoul National University provides research facilities for this study. The Institute of Engineering Research at Seoul National University provided research facilities for this work. Chin-Wan Chung was supported in part by 2018 Seed Money Project of Chongqing Liangjiang KAIST International Program, Chongqing University of Technology, and in part by Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2017jcyjAX0089). Francesco Silvestri was partially supported by project SID2017 of the University of Padova. Ha-Myung Park is currently affiliated with Seoul National University since March 2018.

Authors' addresses: H.-M. Park, C.-W. Chung, and S.-H. Myaeng; emails: {hamyung.park, chung_cw, myaeng}@kaist.ac.kr; F. Silvestri; email: silvestri@dei.unipd.it; R. Pagh; email: pagh@itu.dk; U. Kang (corresponding author); email: ukang@snu.ac.kr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.

1556-4681/2018/10-ART71 $15.00
https://doi.org/10.1145/3237191

Additional Key Words and Phrases: Triangle enumeration, subgraph enumeration, big data, graph algorithm, scalable algorithm, distributed algorithm, network analysis

ACM Reference format:
Ha-Myung Park, Francesco Silvestri, Rasmus Pagh, Chin-Wan Chung, Sung-Hyon Myaeng, and U Kang. 2018. Enumerating Trillion Subgraphs On Distributed Systems. *ACM Trans. Knowl. Discov. Data* 12, 6, Article 71 (October 2018), 30 pages.
https://doi.org/10.1145/3237191

1 INTRODUCTION

How can we find patterns from an enormous graph with billions of vertices and edges? The problem of *subgraph enumeration* is to discover every subgraph that match a given query graph from a large graph one by one. Subgraph enumeration is a very important task for graph data analysis with many applications, including analyzing the social network evolution [19], measuring the significance of motifs in biological networks [14], and observing the dynamics of Internet [13]. If the query graph is a triangle, which is a graph of three vertices connected to each other, we call the problem *triangle enumeration*. Triangle enumeration itself has abundant applications of anomaly detection such as detecting suspicious accounts like advertisers or fake users in social networks [21, 47], uncovering hidden thematic layers on the web [10], discovering roles [6], detecting web spams [4], finding communities [5, 39], and so on. A challenge in subgraph enumeration is handling big real-world networks, such as social networks and WWW, which have millions or billions of vertices and edges. For example, Facebook and Twitter have 1.86 billion[1] and 313 million active users,[2] respectively, and there exist at least 1 trillion unique URLs are on the web.[3]

Even recently proposed algorithms, however, fail to enumerate subgraphs from such large graphs. The algorithms have been proposed in different ways: I/O efficient algorithms [16, 27, 30], distributed memory algorithms [2, 12, 42], and MapReduce algorithms [1, 7, 28, 33, 35, 38, 44, 45]. These algorithms have a limited scalability. The I/O efficient algorithms use only a single machine, and thus they cannot process a graph exceeding the external memory space of the machine. The distributed memory algorithms use multiple machines but cannot process a graph whose intermediate data exceed the capacity of distributed-memory. The state of the art MapReduce algorithm [35] for triangle enumeration, named CTTP, significantly increases the size of a processable dataset; CTTP reduces the amount of intermediate data in a MapReduce round by dividing the entire task into several sub-tasks and processing them in separate MapReduce rounds. Even CTTP, however, takes a very long time to process an enormous graph because, in every round, it reads the entire dataset and shuffles a lot of edges. Indeed, shuffling a large amount of data in a short time interval causes network congestion and heavy I/O to disks, which decrease the scalability and the fault tolerance, and prolong the running time significantly. Thus, it is desirable to shrink the amount of shuffled data.

In this article, we propose scalable methods for enumerating trillion subgraphs on distributed systems. We first propose PTE (*Pre-partitioned Triangle Enumeration*), a new distributed algorithm for enumerating triangles in an enormous graph by resolving the structural inefficiency of the previous MapReduce algorithms. PTE uses the same vertex coloring technique as CTTP (or TTP [33]) to divide the entire task into sub-tasks, but avoids the massive intermediate data problem by pre-partitioning the graph in advance. After that we propose PSE (*Pre-partitioned*

[1] http://newsroom.fb.com/company-info.
[2] https://about.twitter.com/company.
[3] http://googleblog.blogspot.kr/2008/07/we-knew-web-was-big.html.

Subgraph Enumeration), a generalized version of PTE for enumerating subgraphs that match an arbitrary query graph. We show that PTE and PSE successfully enumerate trillions of subgraphs including triangles in a billion scale graph by decreasing three factors: the amount of shuffled data, total work, and network read. The main contributions of this article are summarized as follows:

- We propose PTE, a new distributed algorithm for enumerating triangles in an enormous graph, which is designed to minimize the amount of shuffled data, total work, and network read.
- We propose PSE, a generalized version of PTE for enumerating subgraphs that match an arbitrary query graph. PSE inherits the advantages of PTE mentioned above.
- We prove the efficiency of the proposed algorithms: PTE operates in $O(|E|)$ shuffled data, $O(|E|^{3/2}/\sqrt{M})$ network read, and $O(|E|^{3/2})$ total work, the worst case optimal, where $|E|$ is the number of edges of a graph and M is the available memory size of a machine. PSE requires $O(|E|)$ shuffled data and $O(|E|(|V_q|\sqrt{|E|/M}-1)^{|V_q|-2})$ network read when $P \leq \sum_{k=1}^{|V_q|}\binom{|V_q|\sqrt{|E|/M}}{k}$, where $|V_q|$ is the number of vertices in a query graph and P is the number of processors. Otherwise, PSE requires $O(P|E|)$ network read.
- Our algorithms are experimentally evaluated using large real-world networks. The results demonstrate that PTE outperforms the best previous distributed algorithms by up to 79 times (see Figure 9). Moreover, PTE successfully enumerates more than 3 trillion triangles in the ClueWeb12 graph containing 6.3 billion vertices and 72 billion edges. Previous algorithms, such as GraphLab, GraphX, CTTP, and TwinTwig fail to process the graph because of massive intermediate data. Furthermore, PSE successfully enumerates 265 trillions of clique subgraphs with 4 vertices from a subdomain hyperlink network, showing 47 times faster performance than the state of the art distributed subgraph enumeration algorithm.

The codes and datasets used in this article are provided in http://datalab.snu.ac.kr/pse. This article is an extension of the original conference paper [31] and generalizes the triangle enumeration algorithm proposed in the conference paper for subgraph enumeration. The remaining part of the article is organized as follows. In Section 2, we review previous studies related to the triangle and subgraph enumeration. In Section 3, we formally define the problem and introduce important concepts and notations used in this article. We introduce the details of our algorithms in Section 4. The experimental results are given in Section 5. Finally, we make conclusions in Section 6. The symbols frequently used in this article are summarized in Table 1.

2 RELATED WORK

In this section, we discuss related works. We first describe several triangle enumeration algorithms to handle large graphs, including recent MapReduce algorithms related to our work. We also outline the MapReduce model and emphasize the importance of shrinking the amount of shuffled data in improving the performance. After that, we describe VF2 [8], the state-of-the-art in-memory algorithm for subgraph enumeration, which PSE uses as a module. Then, we introduce distributed algorithms for subgraph enumeration.

2.1 I/O Efficient Triangle Algorithms

Recently, several triangle enumeration algorithms have been proposed in I/O efficient ways to handle graphs that do not fit into the main memory [16, 27, 30]. Hu et al. [16] propose Massive Graph Triangulation (MGT), which buffers a certain number of edges in the memory and finds all triangles containing one of these edges by traversing every vertex. Pagh and Silvestri [30] propose a cache oblivious algorithm which colors the vertices of a graph hierarchically so that it

Table 1. Table of Symbols

Symbol	Definition
$G = (V, E)$	Simple graph with the set V of vertices and the set E of edges.
u, v, n	Vertices.
i, j, k	Vertex colors.
(u, v)	Edge between u and v where $u \prec v$.
(u, v, n)	Triangle with vertices u, v, and n where $u \prec v \prec n$.
$(i, j, k), (i, j)$	Subproblems.
$d(u)$	Degree (number of neighbors) of u.
$id(u)$	Vertex number of u, a unique identifier.
$\prec$	Total order on V. $u \prec v$ means u precedes v.
ρ	Number of vertex colors.
ξ	Coloring function: $V \to \{0, \ldots, \rho - 1\}$. $\xi(u)$ is the color of vertex u.
E_{ij}	Set of edges (u, v) where $(\xi(u), \xi(v)) = (i, j)$ or (j, i).
$E^{\star}_{ij}$	Set of edges (u, v) where $(\xi(u), \xi(v)) = (i, j)$.
M	Available memory size of a machine.
P	Number of processors in a distributed system.

does not need to know the cache structure of a system. Kim et al. [27] present OPT which is a parallel external-memory algorithm exploiting the features of a solid-state drive. DUALSIM [26] is the state-of-the-art parallel external-memory algorithm for enumerating subgraphs that match an arbitrary query graph; DUALSIM is generalized from OPT.

These algorithms, however, cannot process a graph exceeding the external memory space of a single machine. Moreover, these algorithms cannot output all triangles if a graph has too many triangles; for example, the ClueWeb12 graph has 3 trillion triangles requiring 70 Terabytes of storage, and the SubDomain graph has 266 trillion clique subgraphs of 4 vertices (see Section 5). We note that single machine algorithms are collaborators of our algorithms rather than competitors; a single machine algorithm is used as a module of PTE and PSE.

2.2 Distributed-Memory Triangle Algorithms

The triangle enumeration problem has been recently targeted in the distributed-memory model which assumes a multi-processor system where each processor has its own memory. We call a processor with a memory *a machine*. Arifuzzaman et al. [2] propose a distributed-memory algorithm based on *Message Passing Interface* (MPI). The algorithm divides a graph into overlapping subgraphs and finds triangles in each subgraph in parallel. GraphLab-PowerGraph (or GraphLab) [12], which is an MPI-based distributed graph computation framework, provides an implementation for triangle enumeration. GraphLab copies each vertex and its outgoing edges γ times on average to multiple machines where γ is determined by the characteristic of the input graph and the number of machines. Since $\gamma|E|$ data are replicated in total, GraphLab fails when $\gamma|E|/P \geq M$, where P is the number of processors and M is the available memory size of a machine. GraphX, a graph computation library for Spark, also provides an implementation of the same algorithm as in GraphLab; thus it has the same limitation in scalability. PDTL (Parallel and distributed triangle listing) [11], a parallel and distributed extension of MGT, shows the impressive speed but has limited scalability: (1) every machine must hold a copy of the entire graph, (2) a part of PDTL runs on a single machine, which can be a performance bottleneck, and (3) it stores entire triangles in a single machine. In summary, all the previous distributed memory algorithms are limited in handling large graphs.

2.3 MapReduce

MapReduce [9] is a programming model supporting parallel and distributed computation to process large data. MapReduce is highly scalable and easy to use, and thus has been used for various important graph mining and data mining tasks, such as radius calculation [24, 25, 34], graph queries [23], triangle counting [21, 33, 35], visualization [20], connected components [32], and tensor decomposition [17, 18, 22, 36, 40, 43]. A MapReduce round transforms an input set of key-value pairs to an output set of key-value pairs by three steps: it first transforms each pair of the input to a set of new pairs by a user-defined function (*map step*), groups the pairs by keys so that all values with the same key are aggregated together (*shuffle step*), and processes the aggregated pairs for each key using another user-defined function to output a new result set of key-value pairs (*reduce step*).

The amount of shuffled data significantly affects the performance of a MapReduce task because shuffling includes heavy tasks of writing, sorting, and reading the data [15]. In detail, each map worker buffers the pairs from the map step in memory (collect). The buffered pairs are partitioned into R regions and written to local disks periodically, where R is the number of reduce workers (spill). Each reduce worker remotely reads the buffered data from the local disks of the map workers via a network (shuffle). When a reduce worker has read all the pairs for its partition, it sorts the pairs by keys so that all values with the same key are grouped together (merge and sort). Because of such heavy I/O and network traffic, a large amount of shuffled data decreases the performance significantly. Thus, it is desirable to shrink the amount of shuffled data as much as possible.

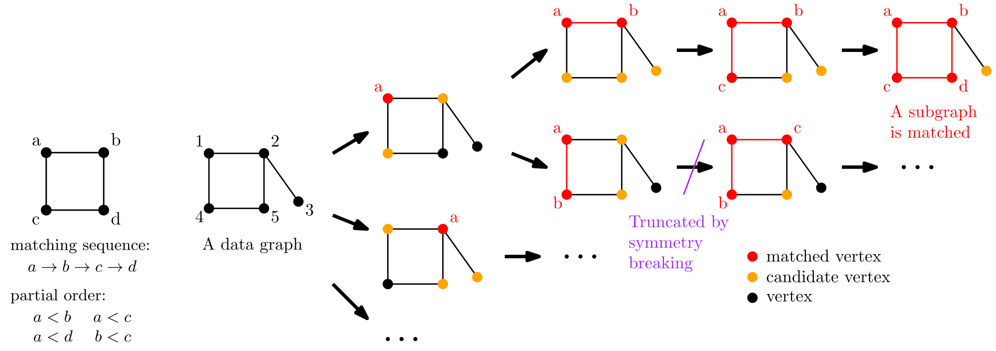
2.4 MapReduce Triangle Algorithms

Several triangle computation algorithms have been designed in MapReduce. We review the algorithms in terms of the amount of shuffled data. The first MapReduce algorithm, which is proposed by Cohen [7], is a variant of node-iterator [41], a well-known sequential algorithm. It shuffles $O(|E|^{3/2})$ length-2 paths (also known as wedges) in a graph. Suri and Vassilvitskii [45] reduce the amount of shuffled data to $O(|E|^{3/2}/\sqrt{M})$ by proposing a graph partitioning based algorithm *Graph Partition* (GP). Considering types of triangles, Park and Chung [33] improve GP by a constant factor in their algorithm, *Triangle Type Partition* (TTP). That is, TTP also shuffles $O(|E|^{3/2}/\sqrt{M})$ data during the process. Aforementioned algorithms cause an out-of-space error when the size of shuffled data is larger than the total available space. Park et al. [35] avoid the out-of-space error by introducing a multi-round algorithm, namely Colored TTP (CTTP). CTTP limits the shuffled data size of a round, and thus significantly increases the size of processable data. However, CTTP still shuffles the same amount of data as TTP does, that is, $O(|E|^{3/2}/\sqrt{M})$. Note that our proposed algorithm in this article shrinks the amount of shuffled data to $O(|E|)$, improving the performance significantly.

The MapReduce algorithms above can be implemented on general distributed systems, using a distributed join algorithm like the one proposed in [3]. Even on general distributed systems, however, they still suffer from the massive shuffled data problem. They join huge amount of data as they shuffle in MapReduce, and the join operation is as expensive as shuffling.

2.5 VF2: A Single Machine Subgraph Enumeration Algorithm

VF2 is an in-memory subgraph enumeration algorithm. VF2 matches query vertices to the vertices of a data graph in a matching sequence. While the matching sequence in VF2 can be arbitrary, our proposed subgraph enumeration algorithm PSE (Section 4.5) uses a topologically sorted sequence of query vertices' partial order that is determined by a process called *symmetry breaking* [14], to remove duplicate outputs and improve the performance. In the data graph, the neighbors of already



(a) A query graph (b) Search states of the graph matching process.

Fig. 1. An example of PSE's matching process for a square query graph based on VF2 and symmetry breaking.

matched vertices are candidates for matching the next query vertex. VF2's feasibility rules truncate some candidates that have no chance to match the query in the future. Figure 1 shows an example of VF2's matching process. The query is a square graph with matching sequence $a \to b \to c \to d$. The data graph has five vertices labeled with numbers from 1 to 5. The vertices are in a total order according to the numbers. VF2 first matches the first query vertex a to any vertex; in this example, a is matched to vertex 1 and the matched vertex is marked red. The neighbors (vertices 2 and 4) of vertex 1 become candidates, which are orange-colored. The next query vertex b is matched to one of the candidates; in this example, we match vertex 2 first. Then, the neighbors (vertices 3 and 5) of vertex 2 are added as candidates. After that, query vertex c is matched to vertex 4. We note that the query vertex c is not matched to candidates 3 and 5 by a feasibility rule that states c and a are connected. By matching query vertex d to query vertex 5, VF2 finds a subgraph that matches the query graph. If the current state has no more candidates to match, VF2 then goes back to the previous state and visits another candidate. This process continues until all search states are visited. Symmetry breaking prevents duplicate output; after matching query vertices a and b to vertices 1 and 4, respectively, symmetry breaking stops matching c to 2 due to the partial order $b < c$; the vertex matched to c must be later than the vertex matched to b in the total order of vertices. As a result, subgraph $(1, 4, 2, 5)$, a duplicate of $(1, 2, 4, 5)$, is not output.

PSE, the proposed subgraph enumeration algorithm, uses VF2 as a module to enumerate subgraphs in each subproblem. ESCAPE [37] is another single machine algorithm; however, the algorithm aims to count all subgraphs with five vertices rather than enumerating subgraphs with a specific graph pattern, and thus does not solve the exact problem that PSE solves.

2.6 Distributed Subgraph Enumeration Algorithms

Several subgraph enumeration algorithms have been proposed based on join techniques. Given a query graph q, they decompose it into a set $\{q_1, \ldots, q_k\}$ of sub-queries. They find the matches $\{G(q_1), \ldots, G(q_k)\}$ of the sub-queries from a data graph G and join them to get the matches $G(q)$ of the original query q. The performance highly depends on which sub-queries the query is decomposed into. Two basic sub-queries are an edge and a star, which is a tree graph of depth 1. We call the methods using the basic sub-queries EdgeJoin [38] and StarJoin [44], respectively. Both methods, however, have problems when they process very large data graphs. EdgeJoin requires a lot of iterative join operations as many times as $|E_q| - 1$, where $|E_q|$ is the number of edges in the query graph q; as a join operation requires one MapReduce round, EdgeJoin performs a lot of MapReduce rounds. On the other hand, StarJoin generates massive intermediate results for high order stars. For example, if a sub-query q_i is a star with $|E_{q_i}|$ edges and a data graph has a high degree vertex with k incident edges, StarJoin generates at least $\binom{k}{|E_{q_i}|}$ matches as an intermediate result. MultiwayJoin [1] makes up for the weak point of EdgeJoin; this method performs all join operations using only a single MapReduce round. MultiwayJoin can be efficient when the size of a query graph is small (e.g., a triangle), but this method generates a tremendous amount of intermediate data during the shuffle step if a query graph has many nodes. The state of the art distributed algorithm for subgraph enumeration is TwinTwigJoin [28]. This algorithm is a special type of StarJoin where the maximum number of edges in a star (sub-query) is limited to 2. By limiting the number of edges, TwinTwigJoin reduces the amount of intermediate data from StarJoin, and requires fewer join operations than EdgeJoin does.

3 PRELIMINARIES

In this section, we define the problem that we are going to solve, introducing several terms and notations formally. We also describe two previously introduced major algorithms for distributed triangle enumeration, which are closely related to both PTE and PSE.

3.1 Problem Definition

A simple graph is an undirected graph that contains no duplicate edges or loops, where a loop is an edge both of whose endpoints are the same vertex. A triangle is a set of three vertices fully connected to each other. We define the problem of triangle enumeration as follows:

Definition 3.1 (Triangle Enumeration). Given a simple graph $G = (V, E)$, the problem of *triangle enumeration* is to discover every triangle in G.

Now we define subgraph enumeration, for which triangle enumeration is a special case with a triangle as the query graph.

Definition 3.2 (Graph Isomorphism). Two graph $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, if and only if $|V_1| = |V_2|$, $|E_1| = |E_2|$, and there exists a mapping function $\zeta : V_1 \rightarrow V_2$ such that $(\zeta(u), \zeta(v)) \in E_2$ for every $(u, v) \in E_1$.

Definition 3.3 (Subgraph Enumeration). Given a simple graph $G = (V, E)$ and a query graph $q = (V_q, E_q)$, the problem of *subgraph enumer ation* is to discover every subgraph s of G such that s is isomorphic to the query graph q.

For simplicity, we call a subgraph isomorphic to a given query graph q a *match*. Note that we do not require an algorithm to retain or emit each triangle or subgraph into any memory system, but to call a local function `enum(·)` with the triangle or the subgraph as the parameter. In other words, an algorithm does not have to keep all subgraphs in the memory at once, which can cause an out-of-space error.

On a vertex set V, we define a total order to uniquely express an edge or a triangle.

Definition 3.4 (Total Order on V). The order of two vertices u and v is determined as follows:

- $u \prec v$ if $d(u) < d(v)$ or $(d(u) = d(v)$ and $id(u) < id(v))$,

where $d(u)$ is the degree, and $id(u)$ is the unique identifier of a vertex u.

We denote by (u, v) an edge between two vertices u and v, and by (u, v, n) a triangle consisting of three vertices u, v, and n. Unless otherwise noted, the vertices in an edge (u, v) have the order of $u \prec v$, and we presume it has a *direction*, from u to v, even though the graph is undirected. Similarly, the vertices in a triangle (u, v, n) also has the order of $u \prec v \prec n$, and we give each edge a name to simplify the description as follows (see Figure 2):

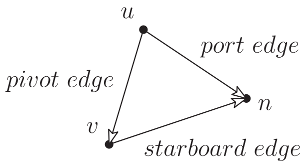


Fig. 2. A triangle with directions by the total order on the three vertices.

Definition 3.5. For a triangle (u, v, n), where the vertices are in the order of $u \prec v \prec n$, we call (u, v) *pivot edge*, (u, n) *port edge*, and (v, n) *starboard edge*.[4]

3.2 Triangle Enumeration in TTP and CTTP

Park and Chung [33] propose a MapReduce algorithm, named TTP, for enumerating triangles. We introduce TTP briefly because of its relevance to our work, and we show that it shuffles a huge amount of data, $O(|E|^{3/2}/\sqrt{M})$.

TTP divides the entire problem into subproblems and solves them independently using multiple machines. To divide the problem, TTP first colors each vertex with one of $\rho = O(\sqrt{|E|/M})$ colors randomly by a hash function $\xi : V \to \{0, \ldots, \rho - 1\}$. Let E_{ij} with $i, j \in \{0, \ldots, \rho - 1\}$ and $i \le j$ be the set $\{(u, v) \in E \mid i = \min(\xi(u), \xi(v)) \text{ and } j = \max(\xi(u), \xi(v))\}$. A triangle is classified as *type-1* if all the vertices in the triangle have the same color, *type-2* if there are exactly two vertices with the same color, and *type-3* if no vertices have the same color. TTP divides the entire problem into $\binom{\rho}{2} + \binom{\rho}{3}$ subproblems of two types:

- (i, j) **subproblem**, with $i, j \in \{0, \ldots, \rho - 1\}$ and $i < j$, is to enumerate triangles in an edge-induced subgraph on $E'_{ij} = E_{ij} \cup E_{ii} \cup E_{jj}$, together with any vertices that are their endpoints. It finds every triangle of type-1 and type-2, where the vertices are colored with i and j. There are $\binom{\rho}{2}$ subproblems of this type.
- (i, j, k) **subproblem**, with $i, j, k \in \{0, \ldots, \rho - 1\}$ and $i < j < k$, is to enumerate triangles in an edge-induced subgraph on $E'_{ijk} = E_{ij} \cup E_{ik} \cup E_{jk}$. It finds every triangle of type-3, where the vertices are colored with i, j, and k. There are $\binom{\rho}{3}$ subproblems of this type.

Each map task of TTP gets an edge $e \in E$ and emits key-value pairs $\langle (i, j); e \rangle$ and $\langle (i, j, k); e \rangle$ for every E'_{ij} and E'_{ijk} containing e, respectively. Thus, each reduce task gets a pair $\langle (i, j); E'_{ij} \rangle$ or $\langle (i, j, k); E'_{ijk} \rangle$, and finds all triangles in the edge-induced subgraph. For each edge, a map task emits $\rho - 1$ key-value pairs (Lemma 2 in [33]); that is, $O(|E|\rho) = O(|E|^{3/2}/\sqrt{M})$ pairs are shuffled in total. TTP fails to process a graph when the shuffled data size is larger than the total available space. CTTP [35] avoids the failure by dividing the tasks into multiple rounds and limiting the shuffled data size of a round. However, CTTP still shuffles exactly the same pairs as TTP; hence CTTP also suffers from the massive shuffled data resulting in a very long running time.

4 PROPOSED METHOD

In this section, we propose scalable methods for enumerating trillion subgraphs on distributed systems. Before considering arbitrary query graphs, we first focus on triangles, the simplest non-trivial subgraphs, to simplify the problem. After that we extend the proposed method to handle general subgraphs as well. There are several challenges in designing an efficient and scalable distributed algorithm for triangle enumeration.

[4] Port and starboard are nautical terms for left and right, respectively.

ALGORITHM 1: Graph Partitioning

```
/* $\psi$ is a meaningless dummy key                                  */
Map      : input $\langle \psi; (u, v) \in E \rangle$
emit $\langle (\xi(u), \xi(v)); (u, v) \rangle$
Reduce   : input $\langle (i, j); E^{\star}_{ij} \rangle$
emit $E^{\star}_{ij}$ to a distributed storage
```

(1) *Minimize shuffled data.* Massive data are shuffled for dividing the problem into subproblems by the previous algorithms. How can we minimize the amount of shuffled data?
(2) *Minimize redundant computation.* The previous algorithms contain several kinds of redundant operations (details in Section 4.2). How can we remove the redundancy?
(3) *Minimize network read.* In previous algorithms, each subproblem reads necessary sets of edges via network, and the amount of network read is determined by the number of vertex colors. How can we decrease the number of vertex colors to minimize network read?

We have the following main ideas to address the above challenges, which are described in detail in later subsections.

(1) *Separating graph partitioning from dividing the problem* decreases the amount of shuffled data to $O(|E|)$ from $O(|E|^{3/2}/\sqrt{M})$ of the previous MapReduce algorithms (Section 4.1).
(2) *Considering the color-direction of edges* removes redundant operations and minimizes computations (Section 4.2).
(3) *Carefully scheduling triangle computations* in subproblems reduces the amount of network read by decreasing the number of vertex colors (Section 4.3).

In the following, we first describe PTE_{BASE}, which exploits pre-partitioning to decrease the shuffled data (Section 4.1). Then, we describe PTE_{CD} to explain how to minimize redundant computation in PTE_{BASE} (Section 4.2). After that, we propose our desired method PTE_{SC} reducing the amount of network read in PTE_{CD} (Section 4.3), and provide theoretical analysis of the methods (Section 4.4). We then describe how to generalize PTE for enumerating subgraphs that match an arbitrary query graph by proposing PSE (Section 4.5). Implementation issues are discussed in the end (Section 4.6). Note that although we describe our methods using MapReduce primitives for simplicity, the methods are general enough to be implemented in any distributed framework (discussions in Section 4.6 and experimental comparisons in Section 5.2.4).

4.1 PTE_{BASE}: Pre-partitioned Triangle Enumeration

In this section, we propose PTE_{BASE}, which rectifies the massive shuffled data problem of previous MapReduce algorithms. The main idea is partitioning an input graph into sets of edges before generating subgraphs, and storing the sets in a distributed storage like *Hadoop Distributed File System* (HDFS) of Hadoop, or *Resilient Distributed Dataset* (RDD) of Spark. We observe that the subproblems of TTP require each set E_{ij} of edges as a unit. It implies that if each edge set E_{ij} is directly accessible from a distributed storage, we do not need to shuffle the edges as TTP does. Consequently, we partition the input graph into $\rho + \binom{\rho}{2} = \frac{\rho(\rho+1)}{2}$ sets of edges according to the vertex colors in each edge; ρ and $\binom{\rho}{2}$ are for E_{ij} when $i = j$ and $i < j$, respectively. Each edge $(u, v) \in E_{ij}$ keeps the order $u \prec v$. Each vertex is colored by a coloring function ξ which is randomly chosen from a pairwise independent family of functions [46]. The pairwise independence of ξ guarantees that edges are evenly distributed. PTE_{BASE} sets ρ to $\lceil\sqrt{6|E|/M}\rceil$ to fit the three edge sets for an

ALGORITHM 2: Triangle Enumeration (PTE_{BASE})

```
   /* $\psi$ is a meaningless dummy key */
   Map : input $\langle \psi; problem = (i, j)$ or $(i, j, k) \rangle$
1  initialize $E'$
2  if $problem$ is of type $(i, j)$ then
       /* $E_{ij} = E^{\star}_{ij} \cup E^{\star}_{ji}$, and $E_{ii} = E^{\star}_{ii}$ */
3      read $E_{ij}$, $E_{ii}$, $E_{jj}$
4      $E' \leftarrow E_{ij} \cup E_{ii} \cup E_{jj}$
5  else if $problem$ is of type $(i, j, k)$ then
       /* $E_{ij} = E^{\star}_{ij} \cup E^{\star}_{ji}$, $E_{ik} = E^{\star}_{ik} \cup E^{\star}_{ki}$, and $E_{jk} = E^{\star}_{jk} \cup E^{\star}_{kj}$ */
6      read $E_{ij}$, $E_{ik}$, $E_{jk}$
7      $E' \leftarrow E_{ij} \cup E_{ik} \cup E_{jk}$
8  enumerateTriangles($E'$)

   /* enumerate triangles in the edge-induced subgraph on $E$ */
9  Function enumerateTriangles($E$)
10     foreach $(u, v) \in E$ do
11         foreach $n \in \{n_u | (u, n_u) \in E\} \cap \{n_v | (v, n_v) \in E\}$ do
12             if $\xi(u) = \xi(v) = \xi(n)$ then
13                 if $(\xi(u) = i$ and $i + 1 \equiv j \mod \rho)$ or $(\xi(u) = j$ and $j + 1 \equiv i \mod \rho)$ then
14                     enum($(u, v, n)$)
15             else
16                 enum($(u, v, n)$)
```

(i, j, k) subproblem into the memory of size M in a processor: the expected size of an edge set is $2|E|/\rho^2$, and the sum $6|E|/\rho^2$ of the size of the three edge sets should be less than or equal to the memory size M.

After the graph partitioning, PTE_{BASE} reads edge sets and finds triangles in each subproblem. In each (i, j) subproblem, it reads E_{ij}, E_{ii}, and E_{jj}, and enumerates triangles in the union of the edge sets. In each (i, j, k) subproblem, similarly, it reads E_{ij}, E_{ik}, and E_{jk}, and enumerates triangles in the union of the edge sets. The edge sets are read from a distributed storage via a network, and the total amount of network read is $O(|E|\rho)$ (see Section 3.2). Note that the network read is different from the data shuffle; the data shuffle is a much heavier task since it requires data collecting and writing in senders, data transfer via a network, and data merging and sorting in receivers (see Section 2.3). However, the network read contains data transfer via a network only.

PTE_{BASE} is described in Algorithms 1 and 2. The graph partitioning is done by a pair of map and reduce steps (Algorithm 1). In the map step, PTE_{BASE} transforms each edge (u, v) into a pair $\langle (\xi(u), \xi(v)); (u, v) \rangle$ (line 1). The edges of the pairs are aggregated by the keys; and for each key (i, j), a reduce task receives $E^{\star}_{ij}$ and emits it to a separate file in a distributed storage (line 2), where $E^{\star}_{ij}$ is $\{(u, v) \in E | (\xi(u), \xi(v)) = (i, j)\}$. Note that $E^{\star}_{ij} \cup E^{\star}_{ji} = E_{ij}$. Thanks to the pre-partitioned edge sets, the triangle enumeration is done by a single map step (see Algorithm 2). Each map task reads edge sets needed to solve a subproblem (i, j) or (i, j, k) (lines 3 and 6), makes the union of the edge sets (lines 4 and 7), and enumerates triangles with a sequential algorithm `enumerateTriangles` (line 8). Although any sequential algorithm for triangle enumeration can

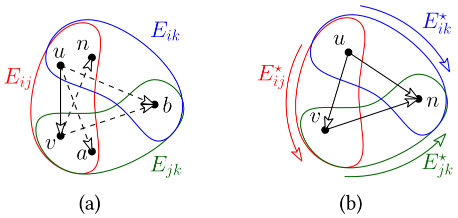


Fig. 3. (a) An example of finding type-3 triangles containing an edge $(u, v) \in E_{ij}$ in an (i, j, k) subproblem. PTE_{BASE} finds the triangle (u, v, b) by intersecting u's neighbor set $\{v, a, b\}$ and v's neighbor set $\{n, b\}$. However, it is unnecessary to consider the edges in E_{ij} since the other two edges of a type-3 triangle containing (u, v) must be in E_{ik} and E_{jk}, respectively. (b) `enumerateTrianglesCD`$(E^{\star}_{ij},\ E^{\star}_{ik},\ E^{\star}_{jk})$ in PTE_{CD} finds every triangle whose pivot edge, port edge, and starboard edge have the same color-directions as those of $E^{\star}_{ij}$, $E^{\star}_{ik}$, and $E^{\star}_{jk}$, respectively. The arrows denote the color-directions.

be used for `enumerateTriangles`, we use CompactForward [29], one of the best sequential algorithms, with a modification (lines 9–16); while the original CompactForward algorithm sorts the neighbors of each vertices by degree, we skip this sorting procedure because the edges are already ordered by the degrees of their vertices. Given a set E' of edges, `enumerateTriangles` runs in $O(|E'|^{3/2})$ total work, the same as that of CompactForward. Note that we propose a specialized algorithm to reduce the total work in Section 4.2. Note also that although every type-1 triangle appears $\rho - 1$ times, PTE_{BASE} emits the triangle only once: for each type-1 triangle of color i, PTE_{BASE} emits the triangle if and only if $i + 1 \equiv j \mod \rho$ given a subproblem (i, j) or (j, i) (lines 12–14). Anyway, PTE_{BASE} still computes a type-1 triangle multiple times redundantly. We completely eliminate the redundant computation in Section 4.2.

4.2 *PTE$_{CD}$*: Reducing the Total Work

PTE_{CD} improves on PTE_{BASE} to minimize the amount of computations by exploiting *color-direction*. We first give an example (Figure 3(a)) to show that the function `enumerateTriangles` in Algorithm 2 performs redundant operations. Let us consider finding type-3 triangles containing an edge $(u, v) \in E_{ij}$ in an (i, j, k) subproblem. `enumerateTriangles` finds such triangles by intersecting the two outgoing neighbor sets of u and v. In Figure 3(a), the neighbor sets are $\{v, a, b\}$ and $\{n, b\}$, and we find the triangle (u, v, b). However, it is unnecessary to consider edges in E_{ij} (that is, (u, v), (u, a), (v, n)) since the other two edges of a type-3 triangle containing (u, v) must be in E_{ik} and E_{jk}, respectively. The redundant operations can be removed by intersecting u's neighbors only in E_{ik} and v's neighbors only in E_{jk} instead of looking at all the neighbors of u and v. In the figure, the two neighbor sets are both $\{b\}$; and we find the same triangle (u, v, b). PTE_{CD} removes the redundant operations by adopting a new function `enumerateTrianglesCD` (lines 11–14 in Algorithm 3). We define the *color-direction* of an edge (u, v) to be from $\xi(u)$ to $\xi(v)$; and we also define the *color-direction* of $E^{\star}_{ij}$ to be from i to j. Then, `enumerateTrianglesCD`$(E^{\star}_{ij}, E^{\star}_{ik}, E^{\star}_{jk})$ finds every triangle whose pivot edge, port edge, and starboard edge have the same color-directions as those of $E^{\star}_{ij}$, $E^{\star}_{ik}$, and $E^{\star}_{jk}$, respectively (see Figure 3(b)). Note that the algorithm does not look at any edges in E_{ij} for the intersection in the example of Figure 3(a) by separating the input edge sets.

Redundant operations of another type appear in (i, j) subproblems. PTE_{BASE} outputs a type-1 triangle exactly once but still computes it multiple times: *type*-1 triangles with a color i appears $\rho - 1$ times in (i, j) or (j, i) subproblems for $j \in \{0, \ldots, \rho - 1\} \setminus \{i\}$. PTE_{CD} resolves the duplicate

ALGORITHM 3: Triangle Enumeration (PTE_{CD})

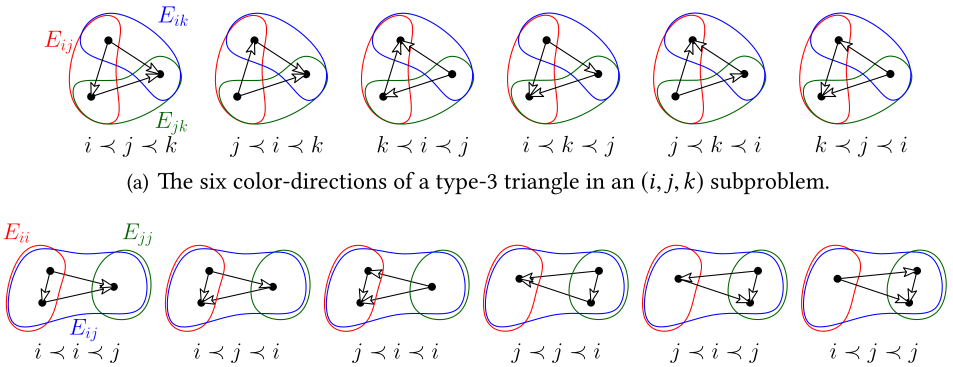
```
    /* $\psi$ is a meaningless dummy key                                                      */
    Map      : input $\langle \psi; problem = (i,j)$ or $(i,j,k)\rangle$
1   if $problem$ is of type $(i,j)$ then
2       read $E^\star_{ij}, E^\star_{ji}, E^\star_{ii}, E^\star_{jj}$
        /* enumerate Type-1 triangles                                                          */
3       if $i+1=j$ then
4           enumerateTrianglesCD($E^\star_{ii}, E^\star_{ii}, E^\star_{ii}$)
5       else if $j=\rho-1$ and $i=0$ then
6           enumerateTrianglesCD($E^\star_{jj}, E^\star_{jj}, E^\star_{jj}$)

        /* enumerate Type-2 triangles                                                          */
7       foreach $(x,y,z) \in \{(i,i,j),(i,j,i),(j,i,i),(i,j,j),(j,i,j),(j,j,i)\}$ do
8           enumerateTrianglesCD($E^\star_{xy}, E^\star_{xz}, E^\star_{yz}$)

9   else if $problem$ is of type $(i,j,k)$ then
10      enumerateType3Triangles($(i,j,k)$)

    /* enumerate every triangle $(u,v,n)$ such that $\xi(u)=i$, $\xi(v)=j$ and $\xi(n)=k$     */
11  Function enumerateTrianglesCD($E^\star_{ij}, E^\star_{ik}, E^\star_{jk}$)
12      foreach $(u,v) \in E^\star_{ij}$ do
13          foreach $n \in \{n_u|(u,n_u) \in E^\star_{ik}\} \cap \{n_v|(v,n_v) \in E^\star_{jk}\}$ do
14              enum($(u,v,n)$)

15  Function enumerateType3Triangles($i, j, k$)
16      read $E^\star_{ij}, E^\star_{ik}, E^\star_{ji}, E^\star_{jk}, E^\star_{ki}, E^\star_{kj}$
17      foreach $(x,y,z) \in \{(i,j,k),(i,k,j),(j,i,k),(j,k,i),(k,i,j),(k,j,i)\}$ do
18          enumerateTrianglesCD($E^\star_{xy}, E^\star_{xz}, E^\star_{yz}$)
```

computation by performing enumerateTrianglesCD($E^\star_{ii}$, $E^\star_{ii}$, $E^\star_{ii}$) exactly once for each vertex color i, thereby making every type-1 triangle appears only once.

Algorithm 3 shows PTE_{CD} using the new function enumerateTrianglesCD. To find type-3 triangles in each (i,j,k) subproblem, PTE_{CD} calls the function enumerateTrianglesCD 6 times for all possible color-directions (lines 10, 15–18) (see Figure 4(a)). To find type-2 triangles in each (i,j) subproblem, similarly, PTE_{CD} calls enumerateTrianglesCD 6 times (lines 7–8) (see Figure 4(b)). For type-1 triangles whose vertices have a color i, PTE_{CD} performs enumerateTrianglesCD($E^\star_{ii}$, $E^\star_{ii}$, $E^\star_{ii}$) only if $i+1 \equiv j \mod \rho$ given a subproblem (i,j) or (j,i) so that enumerateTrianglesCD($E^\star_{ii}$, $E^\star_{ii}$, $E^\star_{ii}$) operates exactly once (lines 3–6). As a result, the algorithm emits every triangle exactly once.

Removing the two types of redundant operations, PTE_{CD} decreases the number of operations for intersecting neighbor sets by more than $2-\frac{2}{\rho}$ times from PTE_{BASE} in expectation. As we will see in Section 5.2.1, PTE_{CD} decreases the operations by up to 6.83× than PTE_{BASE} on real-world graphs.

Theorem 4.1. *PTE_{CD} decreases the number of operations for intersecting neighbor sets by more than $2-\frac{2}{\rho}$ times compared to PTE_{BASE} in expectation.*



(a) The six color-directions of a type-3 triangle in an (i, j, k) subproblem.

(b) The six color-directions of a type-2 triangle in an (i, j) subproblem.

Fig. 4. The color-directions of a triangle according to its type. The function `enumerateTrianglesCD` is called for each color-direction; that is, PTE_{CD} calls it 6 times for type-3 triangles and type-2 triangles, respectively.

Proof. To intersect the sets of neighbors of u and v in an edge (u, v) such that $\xi(u) = i$, $\xi(v) = j$ and $i \neq j$, the function `enumerateTriangles` in PTE_{BASE} performs $d^\star_{ij}(u) + d^\star_{ik}(u) + d^\star_{ji}(v) + d^\star_{jk}(v)$ operations while `enumerateTrianglesCD` in PTE_{CD} performs $d^\star_{ik}(u) + d^\star_{jk}(v)$ operations for each color $k \in \{0, \ldots, \rho - 1\} \setminus \{i, j\}$ where $d^\star_{ij}(u)$ is the number of u's neighbors in $E^\star_{ij}$. Thus, PTE_{BASE} performs $(\rho - 2) \times (d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v))$ additional operations compared to PTE_{CD} for each $(u, v) \in E^{\text{out}}$ where E^{out} is the set of edges $(u, v) \in E$ such that $\xi(u) \neq \xi(v)$; that is,

$$(\rho - 2) \times \sum_{(u,v) \in E^{\text{out}}} \left(d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v) + O(1) \right) \tag{1}$$

We put $O(1)$ because an operation is necessary for checking the existence of the neighbors but $d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v)$ can be smaller than 1. Given an edge (u, v) such that $\xi(u) = \xi(v) = i$, PTE_{BASE} performs $d^\star_{ii}(u) + d^\star_{ij}(u) + d^\star_{ii}(v) + d^\star_{ij}(v)$ operations for each color $j \in \{0, \ldots, \rho - 1\} \setminus \{i\}$; meanwhile, PTE_{CD} performs $d^\star_{ij}(u) + d^\star_{ij}(v)$ operations for each color $j \in \{0, \ldots, \rho - 1\}$. Thus, PTE_{BASE} performs $(\rho - 2) \times (d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v))$ more operations than PTE_{CD} for each $(u, v) \in E^{in}$ where E^{in} is $E \setminus E^{\text{out}}$; that is,

$$(\rho - 2) \times \sum_{(u,v) \in E^{\text{in}}} \left(d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v) + O(1) \right) \tag{2}$$

We add $O(1)$ by the same reason as in Equation (1). Then, the total number of *additional* operations performed by PTE_{BASE}, compared to PTE_{CD}, is the sum of (1) and (2):

$$(\rho - 2) \times \sum_{(u,v) \in E} \left(d^\star_{\xi(u)\xi(v)}(u) + d^\star_{\xi(v)\xi(u)}(v) + O(1) \right) \tag{3}$$

The expected value of $d^\star_{\xi(u)\xi(v)}(u)$ is $d^\star(u)/\rho$ where $d^\star(u)$ is the number of neighbors v of u such that $u \prec v$, since the coloring function ξ is randomly chosen from a pairwise independent family of functions. Thus, (3) becomes as follows:

$$\frac{(\rho - 2)}{\rho} \times \sum_{(u,v) \in E} \left(d^\star(u) + d^\star(v) + O(\rho) \right) \tag{4}$$

	Δ_{ijk}	Δ_{ikj}	Δ_{jik}	Δ_{jki}	Δ_{kij}	Δ_{kji}
$E^\star_{ij}$	✓	✓	▨	▨	✓	
$E^\star_{ik}$	✓	✓	✓			
$E^\star_{ji}$			✓	✓	▨	✓
$E^\star_{jk}$	✓	▨	✓	✓		
$E^\star_{ki}$				✓	✓	✓
$E^\star_{kj}$		✓	▨	▨	✓	✓

Fig. 5. The schedule of data loading for an (i, j, k) subproblem. PTE_{SC} enumerates triangles in columns one by one from left to right. Each check-mark (✓) shows the relations between edge sets and triangle types. Each shaded area denotes when the edge set is in the memory, when we restrict to read an edge set only once in a subproblem.

Meanwhile, the number of operations by PTE_{CD} is $\sum_{(u,v)\in E} (d^\star(u) + d^\star(v) + O(\rho))$ as we will see in Theorem 4.5. Thus, PTE_{CD} reduces the number of operations by $2 - \frac{2}{\rho}$ times from PTE_{BASE} in expectation. □

4.3 *PTE$_{SC}$*: Reducing the Network Read

PTE_{SC} further improves on PTE_{CD} to reduce the amount of network read by *scheduling* calls of the function `enumerateTrianglesCD`. Reading each $E^\star_{ij}$ in $\rho - 1$ subproblems, PTE_{CD} (as well as PTE_{BASE}) reads $O(|E|\rho)$ data via a network in total. For example, $E^\star_{01}$ is read in every $(0, 1, k)$ subproblem for $2 \leq k < \rho$, and the $(0, 1)$ subproblem. It implies that the amount of network read depends on ρ, the number of vertex colors. PTE_{BASE} and PTE_{CD} set ρ to $\lceil\sqrt{6|E|/M}\rceil$ as mentioned in Section 4.1. In PTE_{SC}, we reduce it to $\lceil\sqrt{5|E|/M}\rceil$ by setting the sequence of triangle computation as in Figure 5 that represents the schedule of data loading for an (i, j, k) subproblem. We denote by Δ_{ijk} the set of triangles enumerated by `enumerateTrianglesCD(`$E^\star_{ij}, E^\star_{ik}, E^\star_{jk}$`)`. PTE_{CD} handles the triangle sets one by one from left to right in the figure. The check-marks (✓) show the relations between edge sets and triangle sets. For example, $E^\star_{ij}$, $E^\star_{ik}$, and $E^\star_{jk}$ should be retained in the memory together to enumerate Δ_{ijk}. When we restrict to read an edge set only once in a subproblem, the shaded areas in Figure 5 represent when the edge sets are in the memory. For example, $E^\star_{ij}$ is read before Δ_{ijk}, and is released after Δ_{kij}. Then, we can easily see that the maximum number of edge sets retained in the memory together is 5, and it leads to setting ρ to $\lceil\sqrt{5|E|/M}\rceil$. The procedure of type-3 triangle enumeration with the scheduling method is described in Algorithm 4, which replaces the function `enumerateType3Triangles` in Algorithm 3. Note that the number 5 of edge sets loaded in the memory at a time is optimal as shown in the following theorem.

Theorem 4.2. *Given an (i, j, k) subproblem, the maximum number of edge sets retained in the memory at a time cannot be smaller than 5, if each edge set can be read only once.*

Proof. Suppose that there is a schedule to make the maximum number of edge sets retained in the memory at a time less than 5. We first read 4 edge sets in the memory. Then, (1) any edge set in the memory cannot be released until all triangles containing an edge in the set have been enumerated, and (2) we cannot read another edge set until we release one in the memory. Thus, for at least one edge set in the memory, it should be able to process all triangles containing an edge in the edge set without reading an additional edge set. However, it is impossible because enumerating all triangles containing an edge in an edge set requires 5 edge sets but we have only 4 edge sets. Thus, there is no schedule to make the maximum number of edge sets retained in the memory at a time less than 5. □

ALGORITHM 4: Type-3 Triangle Enumeration in PTE_{SC}

```
Function enumerateType3Triangles(i, j, k)
    read $E^{\star}_{ij}$, $E^{\star}_{ik}$, $E^{\star}_{ji}$, $E^{\star}_{jk}$, $E^{\star}_{kj}$
    foreach $(x, y, z) \in \{(i, j, k), (i, k, j), (j, i, k)\}$ do
        enumerateTrianglesCD($E^{\star}_{xy}$, $E^{\star}_{xz}$, $E^{\star}_{yz}$)
    release $E^{\star}_{ik}$
    read $E^{\star}_{ki}$
    foreach $(x, y, z) \in \{(j, k, i), (k, i, j), (k, j, i)\}$ do
        enumerateTrianglesCD($E^{\star}_{xy}$, $E^{\star}_{xz}$, $E^{\star}_{yz}$)
```

For example, the triangles in Δ_{ijk}, Δ_{ikj}, and Δ_{kij}, which are related to an edge set $E^{\star}_{ij}$, require $E^{\star}_{ij}$, $E^{\star}_{ik}$, $E^{\star}_{jk}$, $E^{\star}_{kj}$, and $E^{\star}_{ki}$.

4.4 Analysis

In this section, we analyze the proposed algorithm in terms of the amount of shuffled data, network read, and total work. We first prove the claimed amount of shuffled data generated by the graph partitioning in Algorithm 1.

Theorem 4.3. *The amount of shuffled data for partitioning a graph is $O(|E|)$, where $|E|$ is the number of edges in the graph.*

Proof. The pairs emitted from the map operation is exactly the data to be shuffled. For each edge, a map task emits one pair; accordingly, the amount of shuffled data is the number $|E|$ of edges in the graph. □

We emphasize that while the previous MapReduce algorithms shuffle $O(|E|^{3/2}/\sqrt{M})$ data, we reduce it to be $O(|E|)$. Instead of data shuffle requiring heavy disk I/O, network read, and massive intermediate data, we only require the same amount of network read, bounded by $O(|E|^{3/2}/\sqrt{M})$.

Theorem 4.4. *PTE requires $O(|E|^{3/2}/\sqrt{M})$ network read.*

Proof. We first show that every $E^{\star}_{ij}$ for $(i, j) \in \{0, \dots, \rho - 1\}^2$ are read $\rho - 1$ times. It is clear that $E^{\star}_{ij}$ such that $i = j$ is read $\rho - 1$ times in (i, k) or (k, i) subproblems for $k \in \{0, \dots, \rho - 1\} \setminus \{i\}$. We now consider $E^{\star}_{ij}$ for $i \neq j$. Without loss of generality, we assume $i < j$. Then, $E^{\star}_{ij}$ is read $\rho - 2$ times in (i, j, k) or (i, k, j) or (k, i, j) subproblems where $k \in \{0, \dots, \rho - 1\} \setminus \{i, j\}$, and once in an (i, j) subproblem; $\rho - 1$ times in total. The total amount of data read by PTE is as follows:

$$(\rho - 1) \sum_{i=0}^{\rho-1} \sum_{j=0}^{\rho-1} |E^{\star}_{ij}| = |E|(\rho - 1) = |E| \left(\sqrt{\frac{5|E|}{M}} - 1 \right) = O\left(\frac{|E|^{3/2}}{\sqrt{M}} \right),$$

where $|E^{\star}_{ij}|$ is the number of edges in $E^{\star}_{ij}$. □

Finally, we prove the claimed total work of the proposed algorithm.

Theorem 4.5. *PTE requires $O(|E|^{3/2})$ total work.*

Proof. Intersecting two sets requires comparisons as many times as the number of elements in the two sets. Accordingly, the number of operations performed by `enumerateTrianglesCD`$(E^\star_{ij}, E^\star_{ik}, E^\star_{jk})$ is

$$\sum_{(u,v)\in E^\star_{ij}} \left(d^\star_{ik}(u) + d^\star_{jk}(v) + O(1)\right),$$

where $d^\star_{ik}(u)$ is the number of u's neighbors in $E^\star_{ik}$. We put $O(1)$ since $d^\star_{ik}(u) + d^\star_{jk}(v)$ can be smaller than 1. PTE_{SC} (as well as PTE_{CD}) calls `enumerateTrianglesCD` for every possible triple $(i, j, k) \in \{0, \ldots, \rho - 1\}^3$; thus the total number of operations is as follows:

$$\begin{aligned}
&\sum_{i=0}^{\rho-1}\sum_{j=0}^{\rho-1}\sum_{k=0}^{\rho-1}\sum_{(u,v)\in E^\star_{ij}} \left(d^\star_{ik}(u) + d^\star_{jk}(v) + O(1)\right)\\
&= \sum_{(u,v)\in E} \left(d^\star(u) + d^\star(v) + O(\rho)\right)\\
&= O(|E|\rho) + \sum_{(u,v)\in E} \left(d^\star(u) + d^\star(v)\right).
\end{aligned}$$

The left term $O(|E|\rho)$ is $O(|E|^{3/2}/\sqrt{M})$ for checking all edges in each subproblem, which occurs also in PTE_{BASE}. The right summation is the number of operations for intersecting neighbors, and it is $O(|E|^{3/2})$ when the vertices are ordered by Definition 3.4 because the maximum value of $d^\star(u)$ for every $u \in V$ is $2\sqrt{|E|}$ as proved in [41]. □

Note that it is the worst case optimal and the same as one of the best sequential algorithms [29].

4.5 Generalization for Enumerating Arbitrary Graph Patterns

This section proposes PSE, a distributed algorithm that enumerates all subgraphs that match an arbitrary query graph, as well as a triangle, by generalizing PTE.

As in a triangle enumeration problem, if vertices are labeled with colors, a subgraph enumeration problem can be divided into subproblems that are independent of each other. PSE first partitions the graph into $\rho + \binom{\rho}{2}$ edge sets like PTE, where ρ is the number of vertex colors; each edge set is E_{ij} for $(i, j) \in \{0, \ldots, \rho - 1\}^2$ such that $i \le j$. We describe how to set the value ρ soon. Given a query graph q with $|V_q|$ vertices, the subgraph enumeration problem is divided into $\sum_{k=1}^{|V_q|} \binom{\rho}{k}$ subproblems; each subproblem $(\tau_1, \ldots, \tau_k)$ is to enumerate every subgraph whose vertices have exactly the colors in $\{\tau_1, \ldots, \tau_k\}$ where $\tau_1 < \cdots < \tau_k$. For example, when $|V_q| = 3$ and $\rho = 4$, the subproblems are (0), (1), (2), (3), (0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3), (0, 1, 2), (0, 1, 3), (0, 2, 3), and (1, 2, 3). Once all the subproblems are solved, we can enumerate all subgraphs that match the query graph, without duplicates. A subgraph that matches the query graph appears in one and only one subproblem related to the color set that contains only the vertex colors of the subgraph. In other words, the correctness of the algorithm follows since each subgraph with a color set $\{\tau_1, \ldots, \tau_k\}$ is emitted only in the subproblem $(\tau_1, \ldots, \tau_k)$. We explicitly note the correctness in the following theorem.

Theorem 4.6. *PSE enumerates exactly one instance of each subgraph that matches a query graph if there exists a method that correctly solves a given subproblem $(\tau_1, \ldots, \tau_k)$ for any $k \in \{1, \ldots, |V_q|\}$.*

The ρ value is determined so that all the edge sets of a subproblem fit into the main memory of a processor, and the parallelism is maximized. A subproblem $(\tau_1, \ldots, \tau_k)$ requires an edge set E_{ij} for each color pair $(i, j) \in \{\tau_1, \ldots, \tau_k\}^2$ such that $i \le j$, and thus the number of required edge sets for the subproblem is $k + \binom{k}{2}$, where k is for the case that the two colors are the same and $\binom{k}{2}$ is for the other cases. The $k + \binom{k}{2}$ edge sets should fit into the memory of size M in a processor, that is,

$$k \times \frac{|E|}{\rho^2} + \binom{k}{2} \times \frac{2|E|}{\rho^2} = \frac{k^2|E|}{\rho^2} \leq \frac{|V_q|^2|E|}{\rho^2} \leq M,$$

as the expected size of an edge set E_{ij} is $|E|/\rho^2$ if $i = j$ and $2|E|/\rho^2$ if $i \neq j$. Accordingly, ρ should be larger than $|V_q|\sqrt{|E|/M}$. At the same time, ρ should be determined so that the parallelism is maximized. Otherwise, if the number of subproblems is less than the number P of processors, some processors are not used while others are busy. Therefore, ρ should be set to satisfy the following expression:

$$P \leq \sum_{k=1}^{|V_q|} \binom{\rho}{k}.$$

When ρ' is the minimum integer value satisfying this expression, the number ρ of vertex colors is determined to be larger than both $|V_q|\sqrt{|E|/M}$ and ρ'. At the same time, we should choose ρ as small as possible to minimize the amount of network read as we will discuss in Theorem 4.7. That is,

$$\rho = \left\lceil \max(|V_q|\sqrt{|E|/M}, \rho') \right\rceil. \tag{5}$$

Reducing the Network Read. For a subproblem $(\tau_1, \ldots, \tau_k)$, if $k = |V_q|$, no match from the subproblem contains *intra-edge*, whose two end vertices have the same color; this is because each vertex within the match has a distinct color and the data graph G is a simple graph without a self-loop. Using this feature, PSE reduces the amount of network read by reading E_{ij} only if $i \neq j$ when it solves a subproblem with $k = |V_q|$. That is, in this case, PSE requires $\binom{|V_q|}{2}$ edge sets instead of $|V_q| + \binom{|V_q|}{2}$ edge sets where each edge set is E_{ij} for $(i, j) \in \{\tau_1, \ldots, \tau_k\}^2$ such that $i < j$.

An edge set E_{ij} is required by every subproblem whose color set contains both i and j. PSE reduces the amount of network read by solving subproblems sharing the same edge sets on the same processor as much as possible. We say that a subproblem A *dominates* another subproblem B if the edge sets required by A are sufficient to solve B. We note that every subproblem with $k < |V_q| - 1$ colors is dominated by one or more subproblems with $|V_q| - 1$ colors. Accordingly, PSE groups the subproblems with $k \leq |V_q| - 1$ colors into $\binom{\rho}{|V_q|-1}$ groups, and processes subproblems in the same group on the same processor. Meanwhile, each subproblem with $k = |V_q|$ colors forms a separate group with only this subproblem. When $\rho = 4$ and $|V_q| = 3$, the problem is divided into $\binom{\rho}{|V_q|-1} + \binom{\rho}{|V_q|}$ subproblem groups, and one possible example is as follows: $\{(0, 1), (0)\}$, $\{(0, 2), (2)\}$, $\{(0, 3), (3)\}$, $\{(1, 2), (1)\}$, $\{(1, 3)\}$, $\{(2, 3)\}$, $\{(0, 1, 2)\}$, $\{(0, 1, 3)\}$, $\{(0, 2, 3)\}$, and $\{(1, 2, 3)\}$. The subproblems can be grouped in several ways. In this example, subproblem (0) is grouped with subproblem $(0, 1)$, but may also be grouped with $(0, 2)$ or $(0, 3)$. It is important to evenly distribute subproblems to groups for the parallelism. We describe how PSE groups the subproblems after the following theorem on the total amount of network read occurred by PSE.

THEOREM 4.7. *PSE requires at most* $\binom{\rho-1}{|V_q|-2} \times |E|$ *network read.*

PROOF. An edge set E_{ij} can be classified into two cases: $i = j$ and $i \neq j$. We first consider an edge set E_{ii} whose two colors are the same. We remind that the edge set E_{ii} is not necessary for the subproblems with $|V_q|$ colors as mentioned already. In other words, no subproblem group with $|V_q|$ colors requires the edge set E_{ii}. The edge set E_{ii} is required by subproblem groups with $|V_q| - 1$ colors including the color i; there are $\binom{\rho-1}{|V_q|-2}$ such groups. Thus, every edge set E_{ii} is read by

$\binom{\rho-1}{|V_q|-2}$ subproblem groups, incurring the following amount of network read:

$$\sum_{i=0}^{\rho-1} |E_{ii}| \times \binom{\rho-1}{|V_q|-2}. \tag{6}$$

We now consider an edge set E_{ij} such that $i \neq j$. Every subproblem group having the colors i and j together requires the edge set E_{ij}. There are $\binom{\rho-2}{|V_q|-2}$ such groups with $|V_q|$ colors and $\binom{\rho-2}{|V_q|-3}$ groups with $|V_q|-1$ colors. Thus, every edge E_{ij} with $i \neq j$ is read by $\binom{\rho-2}{|V_q|-2} + \binom{\rho-2}{|V_q|-3}$ subproblem groups, and the sum is $\binom{\rho-1}{|V_q|-2}$ according to the Pascal's rule.

$$\sum_{i=0}^{\rho-2} \sum_{j=i+1}^{\rho-1} |E_{ij}| \times \binom{\rho-1}{|V_q|-2}. \tag{7}$$

As shown in Equations (6) and (7), an edge set E_{ij} is always required by $\binom{\rho-1}{V_q-2}$ subproblem groups regardless of whether $i = j$ or $i \neq j$. The union of all edge sets is exactly the original edge set E. Thus, the amount of network read is

$$(6) + (7) = \binom{\rho-1}{|V_q|-2} \times |E|,$$

and the theorem follows. □

By applying Equation (5) to Theorem 4.7, the amount of network read becomes $O(|E|(|V_q|\sqrt{|E|/M}-1)^{|V_q|-2})$ if $P \leq \sum_{k=1}^{|V_q|} \binom{|V_q|\sqrt{|E|/M}}{k}$, i.e., $\rho = |V_q|\sqrt{|E|/M}$. If $P > \sum_{k=1}^{|V_q|} \binom{|V_q|\sqrt{|E|/M}}{k}$, i.e., $\rho = \rho'$, PSE requires $O(P|E|)$ network read as $\binom{\rho'-1}{|V_q|-2} \leq \sum_{k=1}^{|V_q|} \binom{\rho'}{k} = O(P)$.

Ensuring the Parallelism. It is crucial to evenly distribute subproblems to groups for the parallelism. However, because each subproblem can belong to different restricted groups, it is easy to distribute unevenly if we are not careful. PSE ensures that all the groups have a similar number of subproblems. To accomplish this goal, PSE assigns the subproblems to the groups as follows. PSE first assigns a subproblem with $|V_q|-1$ or $|V_q|$ colors to each group. Each remaining subproblem is assigned to a *dominating* group; if a subproblem in a group A dominates a subproblem B, we say that the group A dominates the subproblem B. For example, group $\{(0,1),(0)\}$ dominates subproblems (0), (1), and $(0,1)$. We note that subproblems with $|V_q|$ colors do not dominate any other subproblems, and thus no subproblem is assigned to a group with $|V_q|$ colors. All groups are initially marked as *not full*.

(1) Among groups that are not full, PSE selects a *min-group*, which is a group with the least number of subproblems. If there are two or more min-groups, PSE selects one randomly. If the selected group dominates no remaining subproblem, we mark the group full and do (1) again.
(2) Among unassigned subproblems dominated by the selected group, PSE selects one dominated by the least number of min-groups. If two or more subproblems are in the tie, PSE randomly selects one of them.
(3) PSE assigns the selected subproblem to the selected group and repeats (1) and (2) until all the subproblems are assigned.

We consider an example with $|V_q| = 3$ and $\rho = 4$. Let us assume that there are six groups $\{(0,1),(0)\}$, $\{(0,2)\}$, $\{(0,3)\}$, $\{(1,2)\}$, $\{(1,3),(3)\}$, and $\{(2,3)\}$, and two remaining subproblems (1) and (2); the groups with $|V_q|$ colors are omitted. The min-groups are $\{(0,2)\}$, $\{(0,3)\}$, $\{(1,2)\}$, and $\{(2,3)\}$. PSE selects one min-group randomly, and we assume that group $\{(1,2)\}$ is selected in this example. PSE then selects subproblem (1) as it is dominated by only one min-group $\{(1,2)\}$ while subproblem (2) is dominated by three min-groups $\{(0,2)\}$, $\{(1,2)\}$, and $\{(2,3)\}$. Since PSE assigned subproblem (1) to group $\{(1,2)\}$, the subproblem (2) can be assigned to group $\{(0,2)\}$ or $\{(2,3)\}$; the maximum number of subproblems in a group is then 2. If PSE assigned subproblem (2) to group $\{(1,2)\}$, however, subproblem (1) must be assigned to one of the groups $\{(0,1),(0)\}$, $\{(1,2),(2)\}$, and $\{(1,3),(3)\}$; the group with subproblem (1) must have three subproblems. In other words, the subproblems are unevenly distributed.

Solving Subproblems. One advantage of PSE is that it can use any single-machine algorithm for the subgraph enumeration, which has been studied in depth for a long time, to solve each subproblem. If we adopt a single-machine algorithm carelessly, however, some matches may be enumerated multiple times. For example, let us assume that we are solving a subproblem $(\tau_1, \ldots, \tau_k)$ where $k < |V_q|$. We read the edge sets required by the subproblem and find matches using a single-machine algorithm from the edge induced subgraph built from the edge sets. Then, we get all the subgraph matches having color set $\{\tau_1, \ldots, \tau_k\}$. The problem is that the single machine algorithm also finds matches whose color set is a subset of $\{\tau_1, \ldots, \tau_k\}$, and thus some matches are enumerated multiple times. One easy way to workaround this problem is adding a filtering process that checks whether each match has exactly the same colors as the subproblem; if a match does not have any of the colors in the subproblem, the match is not enumerated. This method eliminates duplicate outputs but still performs much redundant computation.

PSE uses the dominance between subproblems to reduce redundant computation; if a subproblem A dominates a subproblem B, a single machine algorithm for the subproblem A also computes all matches of the subproblem B. We note that, in a group, every subproblem is dominated by the subproblem with the largest number of colors in the group. In order to solve all subproblems in a group, PSE executes a single-machine algorithm only once on the edge-sets for the subproblem with the largest number of colors in the group. PSE filters matches that are not of the group's subproblems to prevent duplicate output.

The pseudo-code of PSE is listed in Algorithm 5. PSE consists of a single map function. The input parameters of the map function are a subproblem group g and the query graph q. Let C_g be the set of the subproblems' color sets (line 2), and c be the union of the color sets in C_g (line 3). PSE reads several edge sets for solving the subproblems from a distributed storage (lines 5–12); PSE reads every edge set E_{ij} such that $(i,j) \in c^2$ and $i \leq j$, but if the number $|c|$ of colors in c equals the number $|V_q|$ of vertices in the query graph, PSE does not read edge sets that consist of only a single vertex color. Then, from the union E' of the edge sets, PSE finds all subgraphs that match the query graph q (line 13). Among the subgraphs found, PSE enumerates only the subgraphs whose color set is in C_g, for correctness (lines 14–17). Note that, since this filtering is performed as a pipelined task in distributed systems, the set S of subgraphs does not remain in the memory all at once.

4.6 Implementation

In this section, we discuss practical implementation issues of PTE and PSE. We focus on the most famous distributed computing frameworks, Hadoop and Spark. Note that PTE and PSE can be implemented for any distributed framework that supports map and reduce functionalities.

PTE on Hadoop. We describe how to implement PTE on Hadoop which is the de facto standard of the MapReduce framework. The graph partitioning method (Algorithm 1) of PTE is implemented as a single MapReduce round. The result of the graph partitioning method has a custom output format that stores each edge set as a separate file in HDFS; and thus each edge set is accessible

ALGORITHM 5: Subgraph Enumeration (PSE)

```
/* $\psi$ is a meaningless dummy key                                    */
/* $g$ is a subproblem group                                            */
/* $q$ is the query graph                                               */
Map      : input $\langle \psi; g \rangle$, $q = (V_q, E_q)$
initialize $E'$
$C_g \leftarrow \{c_p = \text{color set of } p \mid p \in g\}$
$c \leftarrow \bigcup_{c_p \in C_g} c_p$
Let $|c|$ be the number of colors in $c$
if $|c| = |V_q|$ then
    foreach $(i, j) \in c^2$ such that $i < j$ do
        read $E_{ij}$
        $E' \leftarrow E' \cup E_{ij}$
else
    foreach $(i, j) \in c^2$ such that $i \leq j$ do
        read $E_{ij}$
        $E' \leftarrow E' \cup E_{ij}$
$S \leftarrow$ enumerateSubgraphs($E'$, $q$)
foreach $s \in S$ do
    Let $c_s$ be the color set of $s$
    if $c_s \in C_g$ then
        enum($s$)
```

by the path of the file. Each edge set $E^{\star}_{ij}$ is stored in an adjacency list format where the source vertex color is i and the destination vertex color is j so that for a vertex u of the color i, we can directly access its neighbors v such that $u \prec v$. The triangle enumeration method (Algorithms 2 or 3) of PTE is implemented as a single map step where each map task processes an (i, j) or (i, j, k) subproblem. For this purpose, we generate a text file where each line is (i, j) or (i, j, k), and make each map task read a line and solve the subproblem.

PSE on Hadoop. From an algorithmic point of view, the graph partitioning of PSE is identical to that of PTE. However, we implement the graph partitioning of PSE slightly differently to improve the performance. While we only need to access succeeding ($\succ$) neighbors of each vertex to enumerate triangles using PTE, we should access preceding ($\prec$) neighbors as well as succeeding neighbors to enumerate subgraphs that match an arbitrary query graph. This is due to the requirement of VF2, the single machine algorithm used in PSE, which visits all neighbors of each vertex. Therefore, creating adjacency lists to directly access all neighbors of each vertex improves the performance. In the map function of the graph partitioning of PTE (Algorithm 1), each edge is emitted only once with its color, i.e., $\langle (\xi(u), \xi(v)); (u, v) \rangle$. For PSE, we emit each edge one more time in the opposite direction, i.e., $\langle (\xi(v), \xi(u)); (v, u) \rangle$, so that each vertex can access both succeeding and preceding neighbors. Each reduce function with key (i, j) receives edge set E_{ij} if $i \leq j$ or E_{ji} if $i > j$, and stores it in the format of a directed adjacency list A_{ij} or A_{ji}, respectively, where all j-color neighbors of each i-color vertex are directly accessible in A_{ij}. The subgraph enumeration method (Algorithm 5) is implemented as a single map step where each map task processes

Table 2. A Summary of Datasets

Dataset (Abbreviation)	Vertices	Edges
Skitter (SK)[5]	1.7M	11M
Youtube (YT)[6]	3.2M	12M
LiveJournal (LJ)[5]	4.8M	43M
Orkut (OK)[5]	3.1M	117M
Twitter (TWT)[7]	42M	1.2B
Friendster (FS)[5]	66M	1.8B
SubDomain (SD)[8]	101M	1.9B
YahooWeb (YW)[9]	1.4B	6.4B
ClueWeb09 (CW09)[10]	4.8B	7.9B
ClueWeb12 (CW12)[11]	6.3B	72B

a subproblem group. For this purpose, we generate a seed file that contains a list of subproblem groups. Each map task reads and solves a subproblem group.

PTE and PSE on Spark. We describe how to implement PTE and PSE on Spark, which is another popular distributed computing framework. The reduce operation of the graph partitioning method (Algorithm 1) is replaced by a pair of *partitionBy* and *foreachPartition* operations of general RDD. The partitionBy operation uses a custom partitioner that partitions edges according to their vertex colors. For each partition, the foreachPartition operation stores an edge set as an adjacency list into HDFS. We generate a new RDD, where each element is a pair of a key of a subproblem group and the list of the subproblems. The new RDD is specially partitioned so that each partition is in charge of a subproblem group.

5 EXPERIMENTS

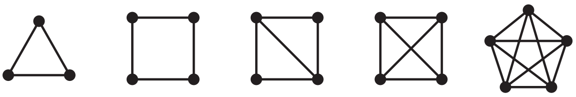
In this section, we experimentally evaluate our algorithms and compare them to recent single machine and distributed algorithms. We aim to answer the following questions.

- **Q1** How much do the three methods of PTE contribute to the performance improvement? (Section 5.2.1)
- **Q2** What is the performance of PSE? (Section 5.2.2)
- **Q3** What about the machine scalability of PTE and PSE? (Section 5.2.3)
- **Q4** How does the performance of PTE and PSE change depending on the underlying distributed framework (MapReduce or Spark)? (Section 5.2.4)

5.1 Setup

5.1.1 Datasets. We use real-world datasets to evaluate the proposed algorithms. The datasets are summarized in Table 2. *Skitter* is an Internet topology graph. *Youtube*, *LiveJournal*, *Orkut*, *Twitter*, and *Friendster* are friendship networks of online social services of the same names, respectively.

[5] http://snap.stanford.edu.
[6] http://konect.uni-koblenz.de.
[7] http://an.kaist.ac.kr/traces/WWW2010.html.
[8] http://webdatacommons.org/hyperlinkgraph.
[9] http://webscope.sandbox.yahoo.com.
[10] http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php?page=Web+Graph.
[11] http://www.lemurproject.org/clueweb12/webgraph.php.



(a) tri (b) sqr (c) sqr-dgn (d) clq4 (e) clq5

Fig. 6. Query graphs.

Table 3. The Number of Subgraphs that Match Each Query in Each Dataset

Dataset	tri	clq4	sqr-dgn	sqr	clq5
SK	28769868	148834439	20522838735	62769198018	1183885507
YT	12323043	29933904	3547559854	9915671458	72636705
LJ	285730264	9933532019	76354588342	51520572777	467429836174
OK	627584181	3221946137	67098889426	127533170575	15766607860
TWT	34824916864	6622234180319	–	–	–
FS	4173724142	8963503263	185191258870	465803364346	21710817218
SD	417761664336	265912212739162	–	–	–
YW	85782928684	5364285380859	–	–	–
CW09	31013037486	–	–	–	–
CW12	3058034046618	–	–	–	–

A dash mark (–) means that the number is not known.

SubDomain is a hyperlink network among domains where an edge exists if there is at least one hyperlink between two subdomains. *YahooWeb*, *ClueWeb09*, and *ClueWeb12* are page level hyperlink networks on the Web. Each dataset is preprocessed to be a simple graph. We reorder the vertices in each edge (u, v) to be $u \prec v$ using an algorithm in [7]. These tasks are done in $O(E)$.

5.1.2 *Query Graphs.* The five query graphs in Figure 6 are used in the experiments. The query graphs are a triangle, a square, a square with a diagonal edge, and two cliques with four and five vertices, respectively. We name the query graphs as tri, sqr, sqr-dgn, clq4, and clq5, respectively. The number of subgraphs matching each query graph in each dataset is listed in Table 3. A dash mark (–) means that the number is not known.

5.1.3 *Experimental Environment.* We implement PTE and PSE on Hadoop (open source version of MapReduce) and Spark. Results described in Sections 5.2.1, 5.2.2, and 5.2.3 are from Hadoop implementations; we also describe the Spark results in Section 5.2.4. We compare PTEs and PSE to previous triangle and subgraph enumeration algorithms: CTTP [35], MGT [16], TwinTwig [28], and the triangle counting implementations on GraphLab and GraphX. CTTP is the state of the art MapReduce algorithm. MGT is an I/O efficient external memory algorithm. GraphX is a graph processing API on Spark, a distributed computing framework. GraphLab is another distributed graph processing framework using MPI. TwinTwig is the state of the art MapReduce algorithm for subgraph enumeration. PSE is compared only to TwinTwig because other systems above do not support enumeration of graph patterns other than a triangle. The algorithms are summarized in Table 4.

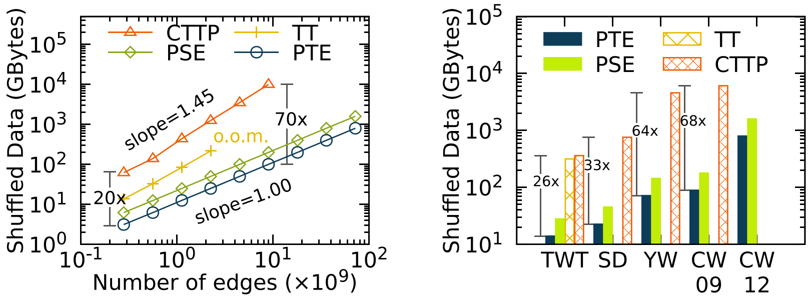
In order to solve subproblems in each machine, PSE uses the VF2 [8] algorithm with a filtering process described in Section 4.5. We note that the performance of PSE can be further improved by using a recent single-machine subgraph enumeration algorithm like DUALSIM [26], which supports only Windows and thus cannot easily be used together with Spark and Hadoop environments.

Table 4. A Summary of Algorithms

Problem	Triangle enumeration					Subgraph enumeration	
Algorithm	PTE	CTTP	MGT	GraphLab	GraphX	PSE	TwinTwig
Property	**Proposed**	MapReduce	External memory	Distributed memory	Distributed memory	**Proposed**	MapReduce

Table 5. Cluster Specification

Hardware		Software
# Machines	20	Hadoop v2.7.3
CPU	Intel Xeon E5-2620v3	Spark v1.5.2
	(hexa-core at 2.4GHz)	GraphLab-PowerGraph v2.2
RAM	32GB	MPICH v3.2



(a) ClueWeb12 with various numbers of edges

(b) Real world graphs

Fig. 7. The shuffled data size of PTE, PSE, TwinTwig (TT), and CTTP (a) on ClueWeb12 with various numbers of edges, and (b) on real-world graphs. PTE shuffles up to 70 times fewer data than CTTP does on ClueWeb12; the gap grows when the data size increases. On real-world graphs, PTE shuffles up to 68 times fewer data than CTTP on ClueWeb09 does.

All experiments were conducted on a cluster with 20 machines where each machine is equipped with an Intel Xeon E5-2620v3 CPU (hexa-core at 2.4GHz) and 32GB RAM. The cluster runs on Hadoop v2.7.3, and consists of 20 worker nodes, one of which serves as a driver node also. The memory size for each worker (a mapper or a reducer) is set to 7GB. Spark v1.5.2 is also installed at the cluster and runs on Hadoop YARN. We operate GraphLab PowerGraph v2.2 on MPICh v3.2 at the same cluster servers. The cluster specification is summarized in Table 5.

5.2 Experimental Results

In this section, we present experimental results to answer the questions listed at the beginning of Section 5.

5.2.1 *Effect of PTE's Three Ideas.*

Effect of Pre-partitioning. We compare PTE and PSE to TwinTwig and CTTP in terms of the amount of shuffled data for triangle enumeration to show the effect of pre-partitioning (Section 4.1). Figure 7(a) shows the results on ClueWeb12 with various numbers of edges. It shows that PTE and PSE shuffle far fewer data than CTTP or TwinTwig, and the difference gets larger as the data size increases; as the number of edges varies from 0.28 billion to 36 billion, the difference

Table 6. The Number of Operations by PTE_{CD} and PTE_{BASE} on Various Graphs

Dataset	PTE_{BASE}	PTE_{CD}	$\frac{PTE_{BASE}}{PTE_{CD}}$
CW12/256	1.5×10^8	2.1×10^7	6.85
CW12/128	6.2×10^8	1.0×10^8	6.22
CW12/64	2.7×10^9	4.6×10^8	5.83
CW12/32	1.2×10^{10}	2.2×10^9	5.10
CW12/16	4.9×10^{10}	1.1×10^{10}	4.44
CW12/8	2.1×10^{11}	5.4×10^{10}	3.90
CW12/4	8.9×10^{11}	2.6×10^{11}	3.47
CW12/2	3.7×10^{12}	1.2×10^{12}	3.18
CW12	1.6×10^{13}	4.9×10^{12}	3.14
TWT	1.1×10^{12}	5.4×10^{11}	2.03
SD	8.3×10^{12}	4.0×10^{12}	2.06
YW	7.4×10^{11}	2.9×10^{11}	2.55
CW09	2.7×10^{11}	6.9×10^{10}	3.94

PTE_{CD} decreases the number of operations by up to 6.85× from PTE_{BASE}.

between PTE and CTTP increases from 20× to 70×. The slopes for PTE, PSE, TT, and CTTP are 1.00, 1.00, 1,32, and 1.47, respectively. They reflect the claimed complexity of shuffled data size, $O(|E|)$ of PTE and $O(|E|^{1.5}/\sqrt{M})$ of CTTP. PSE shuffles twice as much data as PTE does as the map function of PSE emits every edge twice as described in Section 5.1.3. TwinTwig fails to process graphs with more than 4.5 billion edges because of an out of memory error. Figure 7(b) shows the results on real-world graphs; PTE shuffles far fewer data by up to 68× than CTTP does.

Effect of Color-direction. To show the effect of color-direction (Section 4.2), we count the number of all operations in intersecting two neighbor sets (line 13 in Algorithm 3) in Table 6. PTE_{CD} reduces the number of comparisons by up to 6.85× from PTE_{BASE} by the color-direction.

Effect of Scheduling Computation. We also show the effect of scheduling calls of the function `enumerateTrianglesCD` (Section 4.3) by comparing the amount of data read via a network by PTE_{CD} and PTE_{SC} in Table 7. As expected, for every dataset, the ratio $(\frac{PTE_{CD}}{PTE_{SC}})$ is about $1.10 \approx \sqrt{6/5}$.

Running Time Comparison. We now compare the running time of PTEs, PSE, and competitors (CTTP, MGT, GraphLab, GraphX, TwinTwig (TT)) in Figure 8. PTE_{SC} shows the best performance and PTE_{CD} follows it very closely. GraphX does not appear because it fails even with the smallest dataset with 280 million edges because of out-of-memory error. CTTP fails to run within 2 days when edges are more than 9 billion. GraphLab, MGT, and TwinTwig also fail when the number of edges is larger than 600 million, 1.2 billion, and 5 billion, respectively, because of out-of-memory or out-of-range error. The out-of-range error occurs when a vertex id exceeds the range of 32-bit integer limit. Note that, even when MGT can treat vertex ids exceeding the integer range, the performance of MGT would worsen as the graph size increases since MGT performs massive I/O $(O(|E|^2/M))$ when the input data size is large. The slope 1.48 of the PTEs reflects that the total work of them is $O(|E|^{3/2})$ as proved in Theorem 4.5.

Table 7. The Amount of Data Read via a Network on Various Graphs

Dataset	PTE_{CD}	PTE_{SC}	$\frac{PTE_{CD}}{PTE_{SC}}$
CW12/256	13.1GB	13.1GB	1.00
CW12/128	24.8GB	24.8GB	1.00
CW12/64	46.6GB	46.6GB	1.00
CW12/32	89.9GB	89.9GB	1.00
CW12/16	222TB	197GB	1.13
CW12/8	679TB	576GB	1.18
CW12/4	1.9TB	1.7TB	1.12
CW12/2	5.4TB	4.9TB	1.10
CW12	15.6TB	14.2TB	1.10
TWT	39.5GB	39.5GB	1.00
SD	61.0GB	61.0GB	1.00
YW	303GB	298GB	1.09
CW09	798GB	704GB	1.13

The ratio ($\frac{PTE_{CD}}{PTE_{SC}}$) is about $1.10 \approx \sqrt{6/5}$ for all datasets, as expected. When the dataset is small, PTE_{SC} and PTE_{CD} require the same amount of network read, i.e., the ratio is 1, since they use the same number of vertex colors.

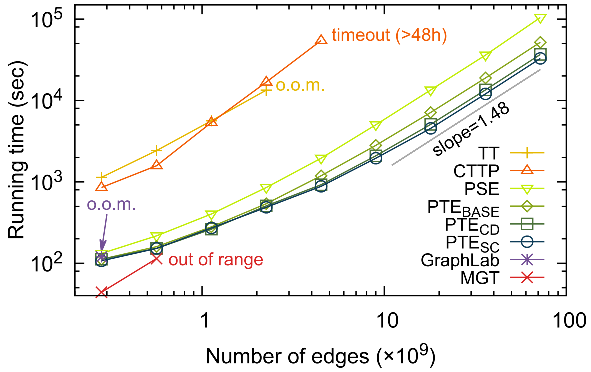


Fig. 8. The running time on ClueWeb12 with various numbers of edges. o.o.m.: out-of-memory. PTE_{SC} shows the best data scalability; only PTEs and PSE succeed in processing the subgraphs containing more than 9 billion edges. The pre-partitioning (CTTP vs. PTE_{BASE}) significantly reduces the running time while the effect of the scheduling function (PTE_{CD} vs. PTE_{SC}) is relatively insignificant.

Figure 9 shows the running time of various algorithms on real world datasets. PTE_{SC} shows the best performances outperforming CTTP and MGT by up to 79 times and 8 times, respectively. Only the proposed algorithms succeed in processing ClueWeb12 with 6.3 billion vertices and 72 billion edges while all other algorithms fail to process the graph.

5.2.2 Varying Queries. For each query in Figure 6, we compare the running time of PSE and TwinTwig, the state of the art MapReduce algorithm for subgraph enumeration; the results are shown in Figure 10. Four graphs are used: Skitter (SK), Youtube (YT), LiveJournal (LJ), and Orkut (OK). Other graphs with many vertices and edges are not displayed here because TwinTwig failed

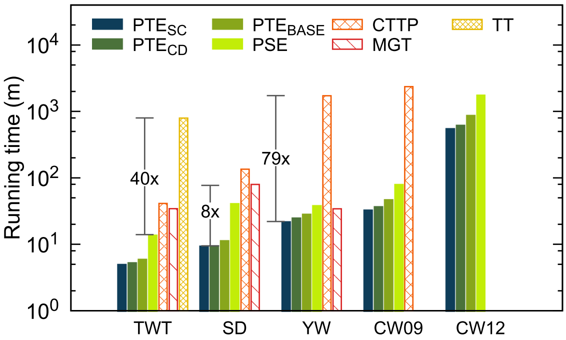
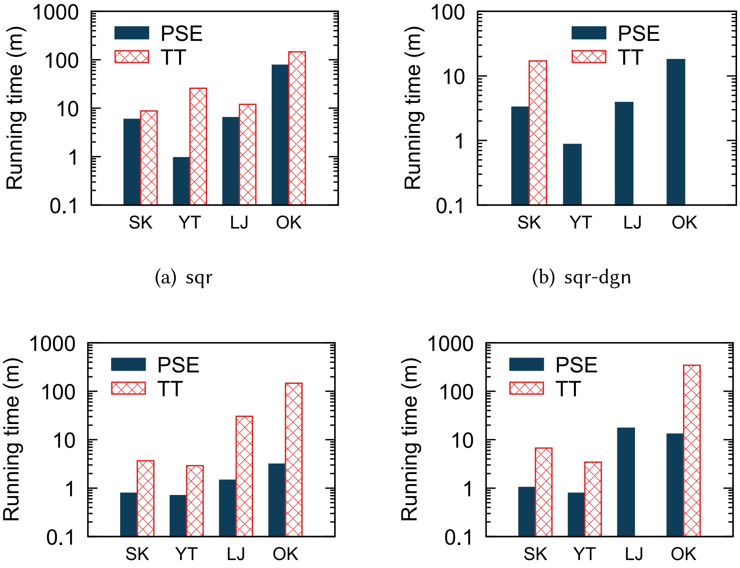


Fig. 9. The running time of proposed methods (PTE_{SC}, PTE_{CD}, PTE_{BASE}, and PSE) and competitors (CTTP, MGT, GraphLab, and TwinTwig (TT)) on real-world datasets (log scale). GraphX is not shown since it failed to process any of the datasets. Missing methods for some datasets mean they failed to run on the datasets. PTE_{SC} shows the best performances outperforming CTTP and MGT by up to 79 times and 8 times, respectively. Only the proposed algorithms succeed in processing the ClueWeb12 graph containing 6.3 billion vertices and 72 billion edges.



(a) sqr

(b) sqr-dgn

(c) clq4

(d) clq5

Fig. 10. The running time of PSE and TwinTwig (TT) for 4 queries: sqr, sqr-dgn, clq4, and clq5. Four graphs are used: Skitter (SK), Youtube (YT), LiveJournal (LJ), and Orkut. PSE shows better performance than TT for all queries, up to 47 times. TT failed to enumerate sqr-dgn pattern in YT, LJ and OK datasets, and clq5 in LJ dataset because of out-of-memory error.

to process them. In all cases, PSE overwhelms TwinTwig; PSE shows about 47 times faster performance when enumerating the query graph clq5 in the data graph LJ. The performance gap of the two algorithms is relatively insignificant for the query graph sqr compared to the other query graphs. Note that the pruning technique of the single machine algorithm used in PSE is relatively not effective for the query sqr, compared to other queries. The single machine algorithm exploits

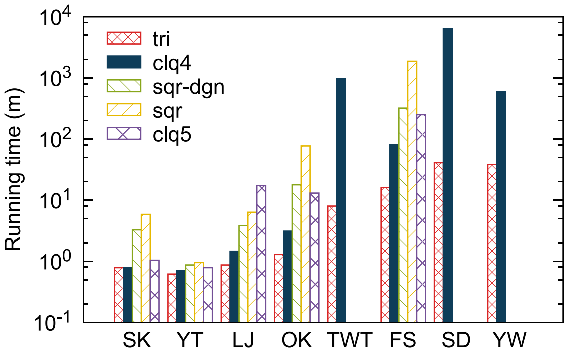
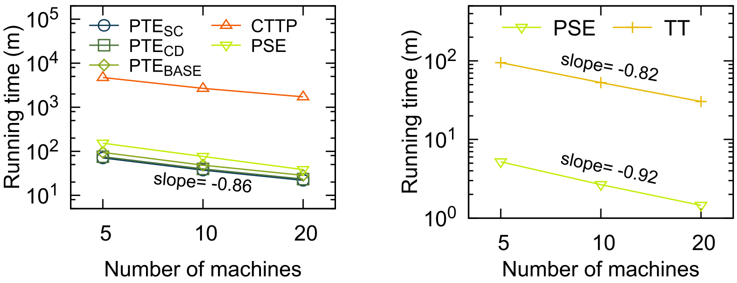


Fig. 11. The running time of PSE on various datasets. The running time tends to be proportional to the number of found subgraphs.



(a) tri, YW

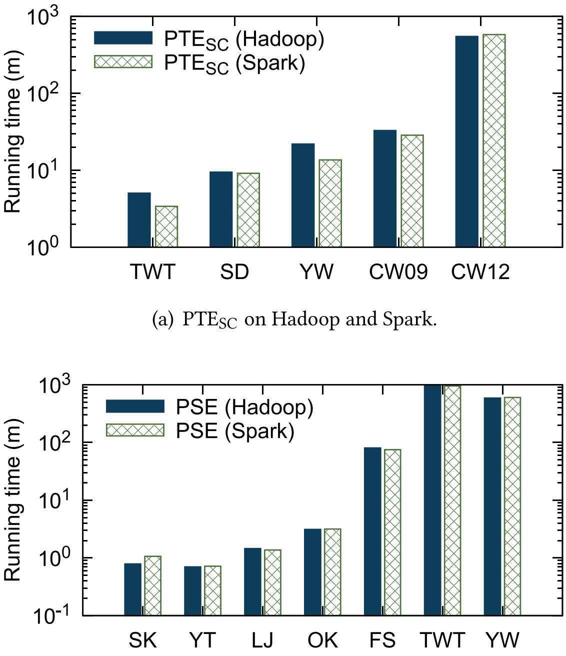
(b) clq4, LJ

Fig. 12. Machine scalability of PTEs, PSE, CTTP, and TwinTwig (TT) for (a) triangle enumeration on YahooWeb (YW) and (b) subgraph enumeration with clq4 graph pattern on LiveJournal (LJ). In (a), GraphLab, GraphX, and TT are excluded because they failed to process YW. PTEs and PSE show strong scalability with exponents −0.86 and −0.92 in triangle enumeration and subgraph enumeration, respectively.

the partial order of a query graph to prune computations. It indicates that the more partial orders of a query graph are, the more the single machine algorithm prunes computations. However, the query sqr has relatively fewer partial orders than other queries do; thus, PSE using the single machine algorithm takes more time when the query is sqr, compared to other queries. Meanwhile, enumerating sqr is easier than in other query graphs for TwinTwig since TwinTwig caches 2-length paths of the input graph as intermediate data and finds subgraphs matching the desired query while sqr subgraphs can be found by joining 2-length paths only once. Nevertheless, PSE shows better performance than TwinTwig even for the query sqr because TwinTwig generates a large amount of intermediate data as much as the number of 2-length paths in an input graph while PSE generates intermediate data only as much as the number of edges in the input graph.

Figure 11 shows the running time of PSE on various datasets. The number of found subgraphs in each graph is listed in Table 3. All the queries in Figure 6 are used. The result shows that the running time tends to be proportional to the number of found subgraphs.

5.2.3 Machine Scalability. We evaluate the machine scalability of PTEs and PSE by measuring the running time of them and competitors varying the number of machines from 5 to 20. Figure 12 shows the running time of PTEs, PSE, CTTP, and TwinTwig (TT) for triangle enumeration on YahooWeb (YW), and subgraph enumeration with the clq4 graph query on LiveJournal (LJ). Note that GraphX, GraphLab, and TT are omitted in Figure 12(a) because they fail to process YW on



(a) PTE$_{SC}$ on Hadoop and Spark.

(b) PSE on Hadoop and Spark. Query clq4 is used.

Fig. 13. The running time of PTE_{SC} and PSE on Hadoop and Spark. The running times do not differ significantly depending on underlying systems.

our cluster. Every version of PTEs and PSE shows strong scalability: the slopes −0.86 of the PTEs and −0.92 of PSE are very close to the ideal value −1. It means that the running time decreases $2^{0.86} = 1.82$ and $2^{0.92} = 1.89$ times, respectively, as the number of machines is doubled.

5.2.4 PTE and PSE on Spark. We implement PTE_{SC} and PSE on Spark as well as on Hadoop to show that PTE and PSE are general enough to be implemented in any distributed system supporting the map and reduce functionality. We compare the running time of implementations on Hadoop and Spark in Figure 13. The result indicates that Spark implementations do not show a better performance than Hadoop implementations even though Spark implementations are able to use a distributed memory as well as disks. This is because the RDD for the pre-partitioned edge sets cannot remain in distributed memory; Spark drives the RDD from distributed memory to distributed disks to free up memory for PTE and PSE to solve subproblems. PTE and PSE require a large amount of memory to solve subproblems, and thus, they consume memory as much as possible. In other words, PTE and PSE for Spark hardly exploit the distributed memory, like the Hadoop versions.

6 CONCLUSION

In this article, we propose PTE, a scalable distributed algorithm for enumerating triangles in very large graphs, and generalize PTE to PSE for enumerating subgraphs that match an arbitrary query graph. We carefully design PTE and PSE so that they minimize the amount of shuffled data, total work, and network read. PTE and PSE show the best performances in real-world data: they outperform the state-of-the-art scalable distributed algorithms by up to 79 times and 47 times, respectively. PTE is the only algorithm that successfully enumerates more than 3 trillion triangles in

ClueWeb12 graph with 72 billion edges while all other algorithms including GraphLab, GraphX, MGT, and CTTP fail. Moreover, PSE succeeds in enumerating 265 trillion clique graph with 4 vertices in a subdomain hyperlink network with 1.9 billion edges while the state of the art distributed subgraph enumeration algorithm TwinTwig fails on the data.

REFERENCES

[1] Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. 2013. Enumerating subgraph instances using map-reduce. In *ICDE*. 62–73.

[2] Shaikh Arifuzzaman, Maleq Khan, and Madhav V. Marathe. 2013. PATRIC: A parallel algorithm for counting triangles in massive networks. In *CIKM*.

[3] Claude Barthels, Gustavo Alonso, Torsten Hoefler, Timo Schneider, and Ingo Müller. 2017. Distributed join algorithms on thousands of cores. *PVLDB* 10, 5 (2017), 517–528.

[4] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting.*TKDD* 4, 3 (2010), 13:1–13:28.

[5] Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. 2011. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E* 83, 5 (2011), 056119.

[6] Bin-Hui Chou and Einoshin Suzuki. 2010. Discovering community-oriented roles of nodes in a social network. In *DaWaK*. 52–64.

[7] Jonathan Cohen. 2009. Graph twiddling in a mapreduce world. *CiSE* 11, 4 (2009), 29–41.

[8] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 10 (2004), 1367–1372.

[9] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *OSDI*. 137–150.

[10] Jean-Pierre Eckmann and Elisha Moses. 2002. Curvature of co-links uncovers hidden thematic layers in the world wide web. *PNAS* 99, 9 (2002), 5825–5829.

[11] Ilias Giechaskiel, George Panagopoulos, and Eiko Yoneki. 2015. PDTL: Parallel and distributed triangle listing for massive graphs. In *ICPP*.

[12] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed graph-parallel computation on natural graphs. In *OSDI*. 17–30.

[13] Enrico Gregori, Luciano Lenzini, and Simone Mainardi. 2013. Parallel k-clique community detection on large-scale networks. *IEEE Trans. Parallel Distrib. Syst.* 24, 8 (2013), 1651–1660.

[14] Joshua A. Grochow and Manolis Kellis. 2007. Network motif discovery using subgraph enumeration and symmetry-breaking. In *RECOMB*. 92–106.

[15] Herodotos Herodotou. 2011. Hadoop performance models. Technical Report CS-2011-05. Duke University. http://www.cs.duke.edu/starfish/files/hadoop-models.pdf.

[16] Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. 2013. Massive graph triangulation. In *SIGMOD*. 325–336.

[17] ByungSoo Jeon, Inah Jeon, Lee Sael, and U. Kang. 2016. SCouT: Scalable coupled matrix-tensor factorization – algorithm and discoveries. In *ICDE*.

[18] Inah Jeon, Evangelos E. Papalexakis, Christos Faloutsos, Lee Sael, and U. Kang. 2016. Mining billion-scale tensors: Algorithms and discoveries. *VLDB J.* 25, 4 (2016), 519–544.

[19] Sanjay Ram Kairam, Dan J. Wang, and Jure Leskovec. 2012. The life and death of online groups: Predicting group growth and longevity. In *WSDM*. 673–682.

[20] U. Kang, Jay-Yoon Lee, Danai Koutra, and Christos Faloutsos. 2014. Net-ray: Visualizing and mining billion-scale graphs. In *PAKDD*.

[21] U. Kang, Brendan Meeder, Evangelos E. Papalexakis, and Christos Faloutsos. 2014. HEigen: Spectral analysis for billion-scale graphs.*TKDE* (2014), 350–362.

[22] U. Kang, Evangelos E. Papalexakis, Abhay Harpale, and Christos Faloutsos. 2012. GigaTensor: Scaling tensor analysis up by 100 times – algorithms and discoveries. In *KDD*. 316–324.

[23] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. 2012. GBASE: An efficient analysis platform for large graphs. *VLDB J.* 21, 5 (2012), 637–650.

[24] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. 2011. PEGASUS: Mining peta-scale graphs. *Knowl. Inf. Syst.* 27, 2 (2011), 303–325.

[25] U. Kang, Charalampos E. Tsourakakis, and Faloutsos Faloutsos. 2009. PEGASUS: A peta-scale graph mining system – implementation and observations. *ICDM* (2009).

[26] Hyeonji Kim, Juneyoung Lee, Sourav S. Bhowmick, Wook-Shin Han, Jeong-Hoon Lee, Seongyun Ko, and Moath H. A. Jarrah. 2016. DUALSIM: Parallel subgraph enumeration in a massive graph on a single machine. In *SIGMOD*. 1231–1245.

[27] Jinha Kim, Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, and Hwanjo Yu. 2014. OPT: A new framework for overlapped and parallel triangulation in large-scale graphs. In *SIGMOD*. 637–648.

[28] Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable subgraph enumeration in mapreduce. *PVLDB* 8, 10 (2015), 974–985.

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

[30] Rasmus Pagh and Francesco Silvestri. 2014. The input/output complexity of triangle enumeration. In *PODS*. 224–233.

[31] Ha-Myung Park, Sung-Hyon Myaeng, and U. Kang. 2016. PTE: Enumerating trillion triangles on distributed systems. In *KDD*. 1115–1124.

[32] Ha-Myung Park, Namyong Park, Sung-Hyon Myaeng, and U. Kang. 2016. Partition aware connected component computation in distributed systems. In *ICDM 2016, December 12-15, Barcelona, Spain*. 420–429.

[33] Ha-Myung Park and Chin-Wan Chung. 2013. An efficient mapreduce algorithm for counting triangles in a very large graph. In *CIKM*. 539–548.

[34] Ha-Myung Park, Chiwan Park, and U. Kang. 2018. PegasusN: A scalable and versatile graph mining system. In *AAAI*.

[35] Ha-Myung Park, Francesco Silvestri, U. Kang, and Rasmus Pagh. 2014. MapReduce triangle enumeration with guarantees. In *CIKM*. 1739–1748.

[36] Namyong Park, ByungSoo Jeon, Jungwoo Lee, and U. Kang. 2016. BIGtensor: Mining billion-scale tensor made easy. In *CIKM, Indianapolis, IN, USA, October 24–28*. 2457–2460.

[37] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *WWW*. 1431–1440.

[38] Todd Plantenga. 2013. Inexact subgraph isomorphism in MapReduce. *J. Parallel Distrib. Comput.* 73, 2 (2013), 164–175.

[39] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. 2004. Defining and identifying communities in networks. *PNAS* 101, 9 (2004), 2658–2663.

[40] Lee Sael, Inah Jeon, and U. Kang. 2015. Scalable tensor mining. *Big Data Research* 2, 2 (2015), 82–86. DOI: http://dx.doi.org/10.1016/j.bdr.2015.01.004 Visions on Big Data.

[41] Thomas Schank. 2007. *Algorithmic Aspects of Triangle-Based Network Analysis*. Ph.D. thesis. University Karlsruhe.

[42] Yingxia Shao, Bin Cui, Lei Chen, Lin Ma, Junjie Yao, and Ning Xu. 2014. Parallel subgraph listing in a large-scale graph. In *SIGMOD*. 625–636.

[43] Kijung Shin, Lee Sael, and U. Kang. 2017. Fully scalable methods for distributed tensor factorization. *IEEE Trans. Knowl. Data Eng.* 29, 1 (2017), 100–113.

[44] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient subgraph matching on billion node graphs. *PVLDB* 5, 9 (2012), 788–799.

[45] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *WWW*. 607–614.

[46] Mark N. Wegman and Larry Carter. 1981. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.* 22, 3 (1981), 265–279.

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

Received June 2017; revised February 2018; accepted July 2018