MapReduce Triangle Enumeration With Guarantees

Ha-Myung Park
KAIST, Korea
hamyung.park@kaist.ac.kr

Francesco Silvestri
Department of Information Engineering
University of Padova, Italy
silvest1@dei.unipd.it

U Kang
KAIST, Korea
ukang@cs.kaist.ac.kr

Rasmus Pagh
IT University of Copenhagen, Denmark
pagh@itu.dk

ABSTRACT

We describe an optimal randomized MapReduce algorithm for the problem of triangle enumeration that requires $O\left(E^{3/2}/(M\sqrt{m})\right)$ rounds, where m denotes the expected memory size of a reducer and M the total available space. This generalizes the well-known vertex partitioning approach proposed in (Suri and Vassilvitskii, 2011) to multiple rounds, significantly increasing the size of the graphs that can be handled on a given system. We also give new theoretical (high probability) bounds on the work needed in each reducer, addressing the "curse of the last reducer". Indeed, our work is the first to give guarantees on the maximum load of each reducer for an arbitrary input graph. Our experimental evaluation shows the scalability of our approach, that it is competitive with existing methods improving the performance by a factor up to 2×, and that it can significantly increase the size of datasets that can be processed.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Design, Experimentation, Algorithms

Keywords

Triangle Enumeration, MapReduce, Graph Mining

1. INTRODUCTION

We are living in a flood of data. These data can be usually represented as graphs such as social networks, interaction networks, road networks and so on. For example, in the most famous social network service Facebook, there are 1.23 billion monthly active users and they form a huge friendship network [12]. Because of the enormity of these networks, it is hard to mine meaningful information from them. Recently, several graph data mining problems have been intensively studied. Especially, the triangle enumeration problem is regarded as one of the fundamental graph mining problems because of its various applications. In a social network, for example, the triangle enumeration problem is used for detecting sybil accounts and measuring content quality [2, 41]. On the web, it is used for finding spam pages and uncovering hidden thematic layers [2, 11]. Also, the triangle enumeration problem is a tool for solving the following problems: *truss decomposition* which finds k-trusses of a graph for all k where a k-truss is a subgraph such that every edge is in (k-2) triangles [36]; *triangular vertex connectivity* which finds all triangularly connected vertex pairs [31]. The importance of these problems and their applications have been introduced in many previous works [6, 8, 11, 37].

Triangle enumeration has been widely studied in the last years, in particular in sequential platforms [17, 28, 7, 5]. However, as the graph size continues to increase, it is effectively impossible to enumerate triangles in massive graphs. One of the methods to deal with such large graphs is to exploit recent parallel programming paradigms. In particular MapReduce [10], and its open source version Hadoop [16], has emerged as a de facto standard framework for processing massive data sets on large scale parallel platforms, such as clusters of commodity PCs. Informally, a MapReduce algorithm transforms an input set of key-value pairs into an output set of key-value pairs in a number of *rounds*, where in each round each pair is first individually transformed into a set of new pairs (*map step*), then grouped by key (*shuffle step*), and finally all values associated with the same key are processed, separately for each key, producing the next new set of key-value pairs (*reduce step*).

The triangle enumeration problem has then been studied in MapReduce [29, 1, 34]. The main objective of these results is to derive efficient MapReduce algorithms requiring a very small number of MapReduce rounds. However, as shown in [1], a MapReduce algorithm using a small number of rounds *must* generate large amount of intermediate data that travel over the network during the shuffle operation. Since the amount of this intermediate data can be much larger than the input size, issues related to the performance of the network and to system failure may arise with massive input graphs. Indeed, the network may be subject to

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2662017.

congestion since a large amount of data is created and sent over the network in a small time interval (i.e., during the shuffle step), reducing the scalability and fault tolerance of the network. It is then desirable to design algorithms that tradeoff round number and decrease the amount of data exchanged during a round, even at the cost of a larger number of rounds. Moreover, we observe that distributing a large computation among different rounds may help to bookkeep the computation and thus to restore it if the system completely fails or if, in the case of cloud services, the computing cost exceeds a given threshold.

In this paper, we address this issue by proposing a new *multi-round* MapReduce randomized algorithm for enumerating all triangles that, by increasing the number of rounds, reduces the maximum amount of intermediate data required during each round. The algorithm exhibits a tradeoff among the number R of rounds, the maximum amount m of memory words required by each reducer, and the total amount M of space required in a round. Specifically, we have $R = O\left(E^{3/2}/(M\sqrt{m})\right)$, where E is the number of edges in the input graph. Our algorithm is based on a vertex partitioning as other previous MapReduce algorithms [34, 1, 29]. However, our partitioning relies on a 4-wise independent hash function which allows us to improve previous analyses that assume a random input graph: we guarantee the claimed results for *any* input graph. Moreover, 4-wise hash functions can be easily and efficiently implemented using the tabulation based technique in [35].

1.1 Contributions

Our main contributions are summarized as follows:

- We propose *Colored Triangle Type Partition* (CTTP), a multi-round MapReduce randomized algorithm for enumerating all triangles in an enormous graph. The algorithm requires $O\left(E^{3/2}/(M\sqrt{m})\right)$ rounds in the worst case. In each round, the total amount of space required by CTTP is M, each mapper uses M/E space, and each reducer uses m space (quantities expressed in memory words). Moreover, the algorithm requires $O\left(E^{3/2}\right)$ total work in expectation, matching the best-known sequential algorithms. The claimed bounds are shown to be optimal.
- We show how the CTTP algorithm can be improved in order to get strong guarantees on the maximum load of each reducer with high probability. Specifically, we show that the aforementioned expected bounds on the space requirements and total work apply with probability at least $1 - 1/E^{\epsilon}$ when $m = \Omega\left(E^{3/4}\sqrt{\log E}\right)$. This result shows how to get rid with high probability of the "curse of the last reducer" [34], that is of an uneven distribution of the total work among the reducers.
- CTTP is experimentally evaluated using several real-world datasets. CTTP is always faster than recent MapReduce algorithms for triangle enumeration, showing up to 2× performance improvement. Experiments also show that there is little overhead in distributing the computation to multiple rounds. Furthermore, by exploiting multiround, we are able to successfully process new datasets where previous algorithms failed because of enormous intermediate data.

Table 1: The notations used in the paper.

Notation	Explanation
E	The number of edges
m	Memory size of a reducer
M	Total data space of all machines
R	The number of rounds
ρ	The number of vertex colors
K	The number of subproblems
K_r	The number of subproblems solved in r-th round
$\xi(x)$	Color of vertex x.
$E_{i,j}$	Set of edges (a, b) such that $min\{\xi(a), \xi(b)\} = i$ and $max\{\xi(a), \xi(b)\} = j$

We remark that the stated bounds apply for each input graph, and do not assume the input to be a random graph as in previous works.

We provide binary file of CTTP and datasets used for the experiments in `http://kdm.kaist.ac.kr/cttp`.

1.2 Paper Organization

The remaining part of the paper is organized as follows. In Section 2, previous works related to the triangulation problem and MapReduce are reviewed. In Section 3 we formalize the problem and describe the MapReduce computational model. In Section 4, the CTTP algorithm is described and analyzed. In Section 5, we show the experimental results. In Section 6, some final comments are provided.

2. RELATED WORK

2.1 Triangle Enumeration

The enumeration problem has been recently targeted in the external memory model, that is on a disk-memory hierarchy where the memory has size m and data are transferred in blocks of size B. In particular, Chu et al. and Hu et al. [7, 17] have proposed deterministic algorithms requiring $O\left(E^2/(mB)\right)$ I/Os, while Pagh and Silvestri [28] proposed an optimal randomized algorithm requiring $O\left(E^{3/2}/(B\sqrt{m})\right)$. Goodrich and Pszona [13] have described an efficient algorithm requiring $O\left((E/B)\log_M E\right)$ I/Os for graphs with constant arboricity, while Chiba and Nishizeki have proposed a general RAM algorithm requiring $O\left(\alpha E\right)$ work for graphs with arboricity α. The relations between listing and other problems have also been widely investigated, see for instance Williams and Williams [39] for a reduction to matrix multiplication, and Jafargholi and Viola [18] for 3SUM/3XOR. Triangle listing in certain classes of random graphs has been addressed recently by Berry et al. [3] to explain the empirically good behavior of simple triangle listing algorithms. For the related problem of counting the number of triangles in a graph, we refer to [24] and references therein.

Triangle enumeration has been explicitly addressed in the MapReduce framework by Afrati et al. [1]. However, the MapReduce algorithms for triangle *counting* proposed by Suri and Vassilvitskii [34] and by Park and Chung [29] can be easily adapted to triangle enumeration. All these algorithms require just $O\left(1\right)$ MapReduce rounds, but replicate each edge $\Theta\left(\sqrt{E/m}\right)$ times on the average, where m denotes the available space of a reducer. All these algorithms rely on a vertex partitioning that split the problem into $(E/m)^{3/2}$ subproblems, each one solved in a reducer of size $O\left(m\right)$ in

a single round. We observe that the vertex partitions proposed by the previous works provide balanced subproblems only assuming an input random graph, and fail for some special graphs such as complete graphs. Finally, we recall that MapReduce algorithms exhibiting tradeoffs between round number, reducer size and total memory have been studied in [4, 30] for sparse and dense matrix multiplication.

2.2 Hadoop and MapReduce

MapReduce is a popular distributed programming framework [10] for processing very large amount of data. MapReduce has several advantages: (a) it allows programming for distributed systems easy; (b) it provides fault-tolerance; (c) it is highly scalable; and (d) it requires a relatively cheap cost to build and maintain a cluster. MapReduce, and its open source version Hadoop [16], have been used for many important graph mining tasks, like radius/diameter [23], graph queries [22], triangle [21], and visualization [20].

3. PRELIMINARIES

3.1 Computational Model

Our algorithms are designed and analyzed in the computational model for MapReduce, named MR(m, M), proposed in [30]. The model is defined in terms of two parameters m and M, characterizing the maximum amount of memory available to a map/reduce function, and the maximum amount of memory available in the whole system, respectively.

An algorithm in this model specifies a sequence of *rounds*. The computation within each round is defined by map and reduce functions whose input and output are multisets of key-value pairs. A pair is denoted as $\langle k; v\rangle$, where k is the key and v the value. Function *map* takes as input one pair and outputs a multiset of new pairs. Function *reduce* receives as input pairs with the same key and outputs a multiset of new pairs. We use the keyword `emit`$(\langle k; v\rangle)$ within a map/reduce function for specifying that the pair $\langle k; v\rangle$ is an output pair.

The r-th round, for each $r \geq 0$, is organized as follows. The input of the round is a multiset I_r of pairs. A new multiset W_r is then generated by applying the map function to each pair in I_r (*map step*); we call *mapper* each application of the map function. Subsequently, pairs in W_r with the same key are grouped together (*shuffle step*). Finally, each group of pairs with the same key is processed by the reduce function, and the multiset O_r containing the output of the applications of the reduce function to each group is the final output of the round (*reduce step*). We denote with *reducer* a single application of the reduce function. Multiset O_r can be the input of the next round $r+1$. Let $m_{k,r}$ be the space required for computing the reduce function on the group defined by key k in round r, and let K_r be the set of distinct keys in W_r. Then, the model requires that $m_{k,r} \leq m$ for each $k \in K_r$ and $r \geq 0$, and that $\sum_{k \in K_r} m_{k,r} \leq M$ for each $r \geq 0$. Similar constraints are required for the map function.

The complexity of an algorithm in the MR(m, M) model is the number of rounds R that it executes in the worst case. The goal is to minimize the round number for given values of m and M. We also define the total work of an algorithm in MR(m, M) as the sum of the work requirement of all mappers and reducers.

3.2 Problem Specification

We consider a simple, undirected graph G (no self loops, no parallel edges) with vertex set V and edge set E. The *enumeration problem* requires to enumerate all triangles within the graph G. We do not require the algorithm to emit a pair for each triangle, but we simply assume that for each triangle (u, v, w) the algorithm calls a local function `enum`$(\cdot)$ with the triangle vertexes as input parameters.

For notational convenience and consistency with earlier papers, whenever the context is clear we use E as a shorthand for the *size* of a set E (and similarly for other sets). We will assume that the elements of V are ordered according to degree, breaking ties among vertices of the same degree arbitrarily. We assume that each edge $\{u, v\}$ requires one memory word, and is initially represented by the pair $\langle \psi; (u, v)\rangle$, where (u, v), with $u < v$, denotes the value and ψ a dummy key. Following [17], for a triangle (v_1, v_2, v_3), with $v_1 < v_2 < v_3$, we call the edge (v_2, v_3) its *pivot edge*, and the vertex v_1 its *cone vertex*. For any integer n, we denote with $[n]$ the set $\{0, \ldots, n-1\}$.

For the sake of simplicity, we assume there are no *very high degree* vertexes, that is, vertexes with degree larger than $\sqrt{Em}$. We observe that triangles with at least one high degree vertex can be enumerated using sorting. Specifically, for each very high degree vertex v, all triangles containing v are found by suitably sorting three times the edge sets E as shown in [28, Lemma 1]. The sorting operation can be carried out using the MapReduce algorithm proposed in [14], which requires $O(1)$ rounds as soon as the reducer size m is larger than $\sqrt{E}$. Since E aggregate space is required for finding all triangles with a given very high degree vertex, M/E very high degree vertexes can be processed in parallel. Thus all triangles with at least one very high degree vertex are enumerated in $O\left(E^{3/2}/(M\sqrt{m})\right)$ rounds, using reducers of size m, constant size mappers, aggregate space M and total work $O\left(E^{3/2}\right)$. We observe that the round complexity and the total work for removing very high degree vertexes are asymptotically negligible compared with the ones of the CTTP algorithm described in the next section.

4. PROPOSED METHOD

In this section we propose *Colored Triangle Type Partition* (CTTP), a MapReduce algorithm for triangle enumeration that exhibits a tradeoff between the number of rounds R, the space m required by each mapper or reducer, and the total aggregate space M. For arbitrary input graph, CTTP requires $R = O\left(E^{3/2}/(M\sqrt{m})\right)$ rounds and guarantees that each mapper and reducer requires space M/E and expected space m, respectively. We then show how to get rid of the "curse of the last reducer" in high probability with minor changes to the CTTP algorithm. As already mentioned we suppose for simplicity that there are no very high degree vertexes (i.e., vertexes with degree larger than $\sqrt{Em}$); we refer to Section 3.2 for an approach that enumerates all triangles with at least one very high degree vertex with the same asymptotic round complexity of the CTTP algorithm.

The CTTP algorithm is based on vertex partitioning as many other previous MapReduce algorithms [34, 1, 29]. CTTP, howerver, partitions vertexes according with a coloring function randomly selected from a 4-wise independent family of functions, as suggested in [28] for triangle enu-

meration in external memory. By just requiring 4-wiseness to the coloring function, our technique overcomes previous analyses that assume a random input graph, and guarantees that the claimed result applies to *any* input graph.

The vertexes are colored with $\rho = \lceil \sqrt{6E/m} \rceil$ colors using a function $\xi : V \to [\rho]$ chosen uniformly at random from a 4-wise independent family of functions. We let $E_{i,j}$, with $i \leq j$ and $i, j \in [\rho]$, denote the set $\{(u,v) \in E | i = \min\{\xi(u), \xi(v)\}$ and $j = \max\{\xi(u), \xi(v)\}\}$. As in [29], a triangle (u, v, w) is classified *type-3* if no two vertexes of the triangle have the same color, *type-2* if there are exactly two vertexes with the same color, and *type-1* if all vertexes have the same color.

The CTTP algorithm decomposes the enumeration problem into $K = \binom{\rho}{3} + \binom{\rho}{2} = \rho(\rho^2 - 1)/6$ subproblems. The subproblems are of two types:

1. (i, j, k)-*subproblem*, with $i < j < k$ and $i, j, k \in [\rho]$: the algorithm finds all type-3 triangles of G where the vertexes are colored with colors i, j, k. It is easy to see that the three edge sets $E_{i,j}, E_{i,k}, E_{k,j}$ suffice for solving the subproblem. There are $\binom{\rho}{3}$ subproblems of this kind.
2. (i, j)-*subproblem*, with $i < j$ and $i, j \in [\rho]$: the algorithm finds all type-1 and type-2 triangles of G where the vertexes are colored with colors i or j. It is easy to see that the three edge sets $E_{i,j}, E_{i,i}, E_{j,j}$ suffice for solving the subproblem. There are $\binom{\rho}{2}$ subproblems of this type.

The CTTP algorithm solves each subproblem within a single reducer, by evenly distributing the K subproblems among $R = \rho E/M$ rounds. The pseudocode of the r-round, for each $0 \leq r < R$ is given in Algorithm 1. The input of each round is a set of pairs $\langle \psi; (u,v) \rangle$ where (u,v) is an edge in E and ψ is a dummy symbol.

In the r-round, CTTP solves the (i, j, k)-subproblem, for each $0 \leq i < j < k < \rho$ with $i + j + k \equiv r \mod R$, in the reducer associated with key (i, j, k), and the (i, j)-subproblem, for each $0 \leq i < j < \rho$ with $i + j \equiv r \mod R$, in the reducer associated with key $(i, j, -1)$. Mappers are responsible for forwarding each edge to the right reducers. For each input pair $\langle \psi; (u,v) \rangle$, the mapper sends the following messages (let $i = \min\{\xi(u), \xi(v)\}$, $j = \max\{\xi(u), \xi(v)\}$):

1. Case $i \neq j$. For each $0 \leq k < \rho$ such that $k \equiv r - i - j \mod R$, $k \neq i$, $k \neq j$, CTTP emits the pair $\langle (k, i, j); (u,v) \rangle$ if $k < i$, or $\langle (i, k, j); (u,v) \rangle$ if $i < k < j$, or $\langle (i, j, k); (u,v) \rangle$ if $j < k$. Also, CTTP emits $\langle (i, j, -1); (u,v) \rangle$ if $i + j \equiv r \mod R$.
2. Case $i = j$. For each $0 \leq k < \rho$ such that $k \equiv r - i \mod R$, $k \neq i$, CTTP emits the pair $\langle (k, i, -1); (u,v) \rangle$ if $k < i$, or $\langle (i, k, -1); (u,v) \rangle$ if $i < k$.

We observe that the proposed distribution of subproblems among rounds guarantees that each mapper emits the same number of pairs (i.e., $\rho/R = \Theta(M/E)$ emitted pairs per mapper in each round). Therefore, the work of the map step can be evenly distributed among the available processing units by the runtime schedule, avoiding that the computation is delayed by a few slow mappers. In contrast, other distributions may require some mappers to emit much more pairs than others. For instance, by lexicographically sorting the triplets/pairs denoting subproblems and then solving each chunk of consecutive E/m subproblems in a round, we get that a few mappers emit up to E/m pairs each, while the remaining mappers emit just a constant number of pairs.

Algorithm 1: Map and reduce functions in the r-th round of CTTP.

Map: input pair $\langle \psi; (u,v) \rangle$
1: $i = \min\{\xi(u), \xi(v)\}$, $j = \max\{\xi(u), \xi(v)\}$
2: **if** $i \neq j$ **then**
3: $k' \leftarrow r - i - j \mod R$
4: **if** $i + j \equiv r \mod R$ **then** emit $\langle (i, j, -1); (u,v) \rangle$
5: **for** $\ell = 0$ to $\rho/R - 1$ **do**
6: $k \leftarrow k' + \ell R$
7: **if** $k < i$ **then** emit $\langle (k, i, j); (u,v) \rangle$
8: **if** $i < k < j$ **then** emit $\langle (i, k, j); (u,v) \rangle$
9: **if** $j < k$ **then** emit $\langle (i, j, k); (u,v) \rangle$
10: **else**
11: $k' \leftarrow r - i \mod R$
12: **for** $\ell = 0$ to $\rho/R - 1$ **do**
13: $k \leftarrow k' + \ell R$
14: **if** $k < i$ **then** emit $\langle (k, i, -1); (u,v) \rangle$
15: **if** $i < k$ **then** emit $\langle (i, k, -1); (u,v) \rangle$

Reduce: input $\langle (i, j, k); E' = E_{i,j} \cup E_{j,k} \cup E_{i,k} \rangle$ if $k \geq 0$, or $\langle (i, j, k); E' = E_{i,j} \cup E_{i,i} \cup E_{j,j} \rangle$ if $k = -1$
1: **if** $k \geq 0$ **then**
2: Find all type-3 triangles in E' with colors i, j, k
3: **else**
4: Find all type-2 triangles in E' with colors i, j
5: Find all type-1 triangles in E' with color i or j.

4.1 Analysis

Let $K = \rho(\rho^2 - 1)/6$ be the total number of subproblems. It is easy to see that $O(K/R)$ subproblems are solved in each round. However, the next lemma shows that exactly K/R subproblems are solved in each round if R is not a multiple of 2 or 3, and that the deviation from K/R is at most $7\rho/(6R)$ otherwise.

LEMMA 1. *Let $K = \rho(\rho^2 - 1)/6$ be the total number of subproblems and let K_r be the number of subproblems solved in the r-th round. Then, $K_r = K/R$ if $R \not\equiv 0 \mod 2$ and $R \not\equiv 0 \mod 3$, and $K/R - 5\rho/(6R) \leq K_r \leq K/R + 7\rho/(6R)$ otherwise.*

PROOF. Let K'_r be the number of (i, j, k)-subproblems solved in a round. We recall that a (i, j, k)-subproblem is solved in the r-th round if the following congruence is verified:

$$i + j + k \equiv r \mod R. \tag{1}$$

It is easy to see that $K'_r = K^*_r/3!$, where K^*_r be the number of triplets (i, j, k) with $i, j, k \in [\rho]$ and $i \neq j \neq k$ (i.e., no assumption on the order) satisfying Equation 1: indeed all and only the 3! permutations of each triplet in K'_r are in K^*_r. For any given values of i and j in $[\rho]$, there are ρ/R values of k that verify the congruence $k \equiv r - i - j \mod R$. Since there are ρ^2 combinations of i, j we get that there are ρ^3/R triplets satisfying Equation 1. However, this number also contains triplets with two or three equal terms. We now count the number of triplets with only two or one distinct values. The number of triplets $(i, i, j), (i, j, i), (j, i, i)$ with $i, j \in [\rho]$ satisfying Equation 1, that is $2i + j \equiv r \mod R$, is ρ^2/R. Note that this number contains also the triplets (i, i, i) satisfying the congruence. The number of triplets (i, i, i) satisfying Equation 1, that is $3i \equiv r \mod R$, depends on the value of R. If $R \not\equiv 0 \mod 3$, then $3i \equiv r \mod R$ has ρ/R solutions for any r. If $R \equiv 0 \mod 3$, then $3i \equiv r \mod R$ has no solution when $r \not\equiv 0 \mod 3$, and $3\rho/R$ solutions when $r \equiv 0 \mod 3$.

By the above arguments, we have that $K'_r = (\rho^3/R - 3(\rho^2/R - \rho/R) - \rho/R)/3! = \binom{\rho}{3}/R$ if $R \not\equiv 0 \mod 3$. On the other hand, we have for $R \equiv 0 \mod 3$ that $K'_r = (\rho^3/R - 3(\rho^2/R - 3\rho/R) - 3\rho/R)/3! = \binom{\rho}{3}/R + 2\rho/(3R)$ if $r \equiv 0 \mod 3$ and $K^*_r = (\rho^3/R - 3\rho^2/R)/3! = \binom{\rho}{3}/R - \rho/(3R)$ if $r \not\equiv 0 \mod 3$.

Consider now the number K''_r of (i,j)-subproblems. Using an argument similar to the previous one we get that $K''_r = \rho(\rho-1)/(2R) = \binom{\rho}{2}/R$ if $R \not\equiv 0 \mod 2$, $K''_r = \rho(\rho-2)/(2R) = \binom{\rho}{2}/R - \rho/(2R)$ if $R \equiv 0 \mod 2$ and $r \equiv 0 \mod 2$, and $K''_r = \rho^2/(2R) = \binom{\rho}{2}/R + \rho/(2R)$ if $R \equiv 0 \mod 2$ and $r \equiv 0 \mod 2$.

The lemma then follows by summing up K'_r and K''_r in the different cases. □

We are now ready to prove the claimed performance of the CTTP algorithm.

THEOREM 1. *The CTTP algorithm correctly enumerates all triangles of the input graph in $(E/M)\left\lceil\sqrt{6E/m}\right\rceil$ rounds. In each round, each mapper and reducer requires space M/E and expected space m, respectively, and the aggregate space is M. The total expected work is $O\left(E^{3/2}\right)$.*

PROOF. It is easy to see that each (i,j,k)-subproblem is evaluated once during round $r = i+j+k \mod R$. Moreover, all edges $E' = E_{i,j} \cup E_{j,k} \cup E_{i,k}$ required for solving the subproblem are available to the reducer. Indeed, for each edge $(u,v) \in E_{i,j}$ the mapper receiving the pair $\langle \psi; (u,v)\rangle$ creates a message $\langle (i,j,k'); (u,v)\rangle$ with $k' = k$ (Line 9 of Algorithm 1). Similarly for edges in $E_{j,k}$ and $E_{i,k}$. An equivalent argument shows that all (i,j)-subproblems are correctly solved.

Since each mapper emits M/E pairs, a mapper clearly requires M/E space. On the other hand, the expected size of each reducer is m since it receives three sets of expected size $m/3$ each. At any time, the aggregate space is M since there are at most M/E copies of each edge.

We now upper bound the work for solving (i,j,k)-subproblems. Let $E_{i,j}$, $E_{j,k}$ and $E_{i,k}$ be the random variables representing the input size of the three sets received by the reducer with key (i,j,k). Using any work-efficient sequential algorithm (see e.g. [17, 28]) within each reducer, we get that the work required by the reducer is $O\left((E_{i,j}+E_{j,k}+E_{i,k})^{3/2}\right) = O\left(E_{i,j}^{3/2}+E_{j,k}^{3/2}+E_{i,k}^{3/2}\right)$. The total work T can thus be upper bounded as follows:

$$T = O\left(\sum_{(i,j,k)}\left(E_{i,j}^{3/2}+E_{j,k}^{3/2}+E_{i,k}^{3/2}\right)\right) = O\left(\rho\sum_{(i,j)}E_{i,j}^{3/2}\right).$$

Let L (resp., S) be the set of pairs (i,j) for which $E_{i,j} > m$ (resp., $E_{i,j} \le m$). Thus

$$T = O\left(\rho\sum_{(i,j)\in S} m^{3/2}\right) + O\left(\rho\sum_{(i,j)\in L}\frac{E_{i,j}^2}{\sqrt{m}}\right).$$

The expected work of T is then

$$\mathbb{E}(T) = O\left(E^{3/2}\right) + O\left(\rho\frac{\mathbb{E}\left(\sum_{(i,j)\in L}E_{i,j}^2\right)}{\sqrt{m}}\right)$$

By [28, Lemma 2], $\mathbb{E}\left(\sum_{(i,j)\in L}E_{i,j}^2\right)$ is upper bounded by $O(Em)$ when the maximum degree of the graph is $\sqrt{Em}$. Thus $\mathbb{E}(T) = O(E^{3/2})$. Since the total work for solving (i,j)-subproblems is negligible, the claim follows. □

Finally, we observe that our algorithm is optimal by deriving a lower bound on the round number required by an algorithm for triangle enumeration. We assume that each edge or vertex requires at least one memory word: that is, at any point in time there can be at most m edges/vertexes in a reducer of size m. This assumption is verified by our algorithm, and is similar to the indivisibility assumption which is usually required for deriving lower bounds on a memory hierarchy.

THEOREM 2. *Any algorithm using reducers of size m and aggregate space M requires, even in the best case, $\lfloor t/(M\sqrt{m})\rfloor$ rounds to enumerate t distinct triangles. The CTTP algorithm is then asymptotically optimal in the worst case.*

PROOF. Without loss of generality, we ignore mappers since they would decrease the lower bound by just a constant factor (in fact, as already noticed in [26], a reduce step can clearly embed the subsequent map step so that a MapReduce computation can be simply seen as a sequence of rounds of reduce steps). A reduce with size m cannot enumerate more than $m^{3/2}$ triangles [28] by the initial assumption on the memory words required by each edge/vertex. Then the maximum number of triangles that can be enumerated in a round is $\sum_{i\in\Gamma} m_i^{3/2}$, where Γ is the set of keys in the round and m_i is the size required by reducer with key i. Since $\sum_{i\in\Gamma} m_i \le M$ and $m_i \le m$, we get that the summation is upper bounded by $M\sqrt{m}$ (i.e., when there are M/m reduces of size m). Then, at least $\lfloor t/(M\sqrt{m})\rfloor$ rounds are required for enumerating t distinct triangles. Since there exists a graph with $t = \Omega\left(E^{3/2}\right)$ (a complete graph with $\sqrt{E}$ vertexes) the worst-case optimality of CTTP follows. □

We observe that the lower bound proposed by Afrati et al. [1] for triangle enumeration in MapReduce does not apply in our settings. Indeed, they show that a one-round algorithm requires in the worst case at least aggregate memory $M = \Omega\left(V^3/\sqrt{m}\right)$, where V is the vertex number. This bound can also be derived as a special case of Theorem 2.

4.2 Getting the High Probability

We now show how the vertex coloring in the CTTP algorithm can be improved in order to get strong guarantees on the maximum load of each reducer. As in the previous section, we assume vertex degree to be not larger than $\sqrt{Em}$. For the sake of simplicity, the results in this section are provided in asymptotic notation, although exact bounds can be derived as in the previous section with simple but tedious derivations.

We use two different coloring techniques for *high degree* vertexes (i.e., with degree in $(\sqrt{E}, \sqrt{Em}]$) and for *low degree* vertexes (i.e., with degree in $[0, \sqrt{E}]$). Low degree vertexes are colored using a coloring function $\zeta : V \to [\rho]$ randomly chosen among a set of $\log E$-wise functions. On the other hand, the coloring of high degree vertexes is deterministically computed as suggested in [32] for subgraph enumeration in external memory. Namely, high degree vertexes are colored using ρ colors in such a way that the sum of the degrees of vertexes with the same color is $\Theta(\sqrt{Em})$. It is easy to see that vertex degree can be computed by sorting the edge set E, while the coloring of the at most $2\sqrt{E}$ high degree vertexes can be computed within a single reducer as soon as $m = \Omega(\sqrt{E})$.

THEOREM 3. *The CTTP algorithm with the above coloring technique enumerates all triangles of the input graph in* $O\left((E/M)\left\lceil\sqrt{E/m}\right\rceil\right)$ *rounds. The algorithm uses* $O(M/E)$ *space per mapper and* $O(M)$ *aggregate total space. Moreover, if* $m = \Omega\left(E^{3/4}\sqrt{\log E}\right)$*, then it holds with probability at least* $1 - 1/E^{\epsilon}$*, for any constant* $\epsilon > 0$*, that each reducer requires* $O(m)$ *space and the total work is* $O\left(E^{3/2}\right)$.

PROOF. In [32, Theorem 3], it is proved that the probability that $E_{i,j} = O(m)$ for any given color pair (i, j) is $1-1/E$ as soon as $m = \tilde{\Omega}\left(\sqrt{E}\right)$ if each low degree vertex is colored by assigning a random color in $[\rho]$ independently and uniformly. The proof relies on a result by [19] for providing strong bounds on the sum of dependent random variables. However, by using a result in [15] instead of the one by [19] within the proof of [32, Theorem 3], it is possible to extend the result to the case low degree vertexes are colored with a $\log E$-wise hash function assuming $m = \Omega\left(E^{3/4}\sqrt{\log E}\right)$.

Consider the edge set E' containing edges adjacent to only low degree vertexes, and let Y_e for $e \in E'$ be an indicator variable set to 1 if e is in $E_{i,j}$ and 0 otherwise. We clearly have $|E_{i,j} \cap E'| = \sum_{e \in E'} Y_e$. Since Y_e and $Y_{e'}$ are dependent if e and e' share a vertex in G and vertexes are colored using $\log E$-wise functions, we use the result by Gradwohl and Yehudayoff [15, Theorem 3.1] for providing a deviation bound on the sum of the dependent random variables Y_e. It follows that $|E_{i,j} \cap E'| = O(m)$ with probability $1 - O\left(\left((E^{3/4}\sqrt{\log E})/m\right)^{\log E}\right)$ (it follows from [15] since we use $\log E$-wise hash functions and a random variable Y_e depends on the at most $2\sqrt{E}$ random variables of adjacent edges).

Consider now the edge set E'' containing edges connecting high degree vertexes of colors i or j to low degree vertexes. By the coloring of high degree vertexes, we have that $E'' = O\left(\sqrt{Em}\right)$. Let Z_e for each $e \in E''$ be an indicator variable set to 1 if the low degree vertex of e has color i or j. We have $|E_{i,j} \cap E''| \leq \sum_{e \in E'} Z_e$ (note that $Z_e = 1$ even if vertexes in e have both color i). Since these variables are dependent, we use again [15, Theorem 3.1], getting that $|E_{i,j} \cap E''| = O(m)$ with probability $1 - O\left(\left((E^{3/4}\sqrt{\log E})/m\right)^{\log E}\right)$ (it follows from [15] since we are using $\log E$-wise hash functions and a random variable Z_e depends on the at most $2\sqrt{E}$ random variables of edges adjacent on the low degree vertex of e).

Finally, consider edges connecting only high degree vertexes. There cannot be more than $O(m)$ of these edges in $E_{i,j}$ since there are at most $\Theta(\sqrt{m})$ high degree vertexes with the same colors.

Therefore, the set $E_{i,j}$ for a given color pair (i, j) has size $O(m)$ with probability $1 - O\left(\left((E^{3/4}\log E)/m\right)^{\log E}\right)$. By applying an union bound, we get that all edge sets $E_{i,j}$ have size $O(m)$ with probability $1 - (E/m)O\left(\left((E^{3/4}\sqrt{\log E})/m\right)^{\log E}\right) \geq 1 - 1/E^{\epsilon}$ by setting $m \geq \alpha E^{3/4}\sqrt{\log E}$ with $\alpha = 2^{\epsilon+1/4}$. Since the size of each reducer if $O(m)$ with probability at least $1 - 1/E^{\epsilon}$, the upper bound on the work complexity easily follows. □

Table 2: The summary of datasets. [1]

Dataset	Nodes	Edges	Triangles	File Size	
				Original	Pre-processed
LiveJournal[2]	4.8 M	4.3 M	285730264	1 GB	679 MB
PhoneCall	30 M	0.2 B	149045937	28 GB	5.6 GB
Twitter[3]	42 M	1.2 B	34824916864	26 GB	23 GB
SubDomain[4]	0.1 B	1.9 B	417761664336	36 GB	36 GB
YahooWeb[5]	1.4 B	6.4 B	85782928684	120 GB	120 GB
ClueWeb09[6]	4.8 B	7.9 B	31013037486	72 GB	153 GB

5. EXPERIMENTS

In this section, we experimentally evaluate the proposed algorithm and compare it to recent MapReduce algorithms for triangulation. We aim to answer the following questions from the experiments.

Q1 How does the number of round affect the performance of CTTP?
Q2 Is CTTP scalable in terms of the size of data and the number of machines?
Q3 Does CTTP work well with real world datasets?

We first introduce the datasets which are used for the experiments, and we specify how to implement the algorithms in Section 5.2. After that, we answer the questions in section 5.3 by presenting the result of the experiments.

5.1 Datasets

We use real-world datasets listed in Table 2 to evaluate the proposed algorithm. They are brought from various sources. LiveJournal is an online community for sharing journals, and the LiveJournal dataset is the friendship network of the community [33]. The PhoneCall dataset is a list of phone call records on December 2007 and on January 2008. It is offered by an anonymous telecommunication service provider. The Twitter dataset is the 'following' network of Twitter, a famous social network service [25]. The SubDomain dataset contains links among subdomains on the Web [27]. The YahooWeb dataset is a page level hyperlink graph [40]. The largest one used in the experiments, the ClueWeb09 dataset is another page level hyperlink graph [38].

Each dataset is preprocessed to be a simple undirected graph. If a pair of vertexes has two edges (a, b) and (b, a), one of them is deleted from the graph. If an edge appears multiple times in a dataset, all of the duplicate edges are removed except one. We also remove all self-loop edges of which source and destination vertexes are the same. This procedure can be simply implemented on MapReduce and the running time is $O(E)$ which is dominated by the running time of the main algorithms. We exclude the preprocessing time when we analyze the running time of all algorithms because they need the same preprocessing before execution.

[1] Links to the datasets are in `http://kdm.kaist.ac.kr/cttp`
[2] `http://snap.stanford.edu/data/soc-LiveJournal1.html`
[3] `http://an.kaist.ac.kr/traces/WWW2010.html`
[4] `http://webdatacommons.org/hyperlinkgraph`
[5] `http://webscope.sandbox.yahoo.com`
[6] `http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php?page=Web+Graph`

5.2 Implementation

We compare the proposed algorithm (CTTP) with the recent MapReduce algorithms (TTP [29] and GP [34]) for triangulation. The proposed algorithm and the compared algorithms are written in Java programming language and they are executed on Hadoop which is regarded as the de facto standard implementation of MapReduce. All experiments are conducted on a Hadoop cluster at KAIST with 40 machines where each machine has 4GB of memory. The default number of reducers is set to 80.

The number ρ of partitions (or colors) for GP and TTP is set to $\sqrt{9E/m}$ and $\sqrt{6E/m}$, respectively, for the following reason. Let us assume that a dataset is divided into ρ partitions. When we suppose that edges are evenly distributed, the probability that vertexes incident to an edge belong to a specific partition is $1/\rho^2$; this kind of edge is called *inner-edge*. Similarly, the probability that the vertexes incident to an edge belong to two specific partitions is $2/\rho^2$; this type of edge is called *outer-edge*. It indicates that the probability that an edge belongs to a subproblem (i, j, k) of GP is $3 \times 1/\rho^2 + 3 \times 2/\rho^2 = 9/\rho^2$ because there are three cases of inner-edge, i, j, and k, and three cases of outer-edge, (i, j), (i, k), and (j, k); thus the expected number of edges in a subproblem of GP is $9E/\rho^2$. All edges in a subproblem should fit in the memory of a reducer, i.e. $9E/\rho^2 \leq m$, and it leads to set ρ to $\sqrt{9E/m}$. TTP processes two types of subproblems, (i, j, k) and (i, j). In contrast with GP, a subproblem (i, j, k) in TTP has no inner-edge, and it indicates that the expected number of edges in a subproblem (i, j, k) is $6E/\rho^2$. The expected number of edges in a subproblem (i, j) is $4E/\rho^2$ which is smaller than a subproblem (i, j, k). Thus, we set ρ to $\sqrt{6E/m}$ for TTP. The number ρ of colors for CTTP is set to $\sqrt{6E/m}$, the same as that of TTP, because CTTP and TTP process exactly the same subproblems.

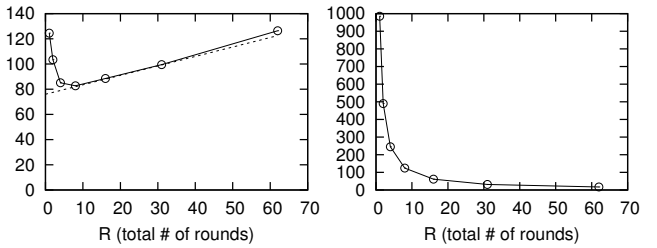
As we mentioned in Section 4, the number R of rounds of CTTP is $O(\rho E/M)$. In our experiments, we suppose a poor condition where the total available space M is only the input size E, i.e. $M = E$, so R is set to ρ in Section 5.3.2, 5.3.3, and 5.3.4. We make an exception if the ρ is smaller than $\sqrt{6r + 1}$ where r is the number of reducers. As already mentioned, the number K of subproblems is $\rho(\rho^2 - 1)/6$. If $R = \rho$, the number K_r of subproblems solved in a round is $(\rho^2 - 1)/6$. If $K_r < r$, CTTP cannot utilize all reducers since several reducers do nothing and just waiting until the other reducers finish. In fact, such situation occurs when the input dataset is so small compared to M. In this case, we set R to 1 as in the previous algorithms.

5.3 Experimental Results

In this section, we present the experimental evaluation answering the questions listed in the beginning of Section 5. The result tells us that 1) the number of rounds affects the performance of CTTP with a small factor, and the additional cost is much smaller than the total cost, 2) CTTP is very scalable in terms of data size, and its machine scalability is close to the optimum, and 3) CTTP works very well with real world datasets, and outperforms the recent algorithms.

5.3.1 Effect of Number of Rounds

In CTTP, the number R of rounds is $O(E^{3/2}/M\sqrt{m})$. While the number E of edges are determined by the input data, the memory space m of a reducer and the total available space M of a cluster is determined by the system. Besides, even in the same system, M changes from time to time. It means that R can vary on the same input data and the same system. Then our question is that how the performance of CTTP is affected by the number of rounds. We, first, set the simplified cost function of CTTP focusing on the number of rounds. After that, we analyze and compare the experimental result with the cost function. In the cost function, we suppose that the disk I/O speed and the network speed are not affected by the amount of processed data. The details of the cost function is described in Section 4 of [9].



(a) Running time (min) (b) Shuffled data size per round (GB)

Figure 1: Effect of the number of rounds. It shows the running time and the shuffled data size per round with various total number of rounds on the Twitter dataset. The dotted line in (a) is the expected result from the cost function of Equation 6. With $R \geq 4$, the result is almost the same as expected. With $R < 4$, however, the result shows much higher running time than our expectation because of the massive intermediate data which deplete the disk I/O performance and network speed. (b) shows the trade-off relationship between the total number of rounds and the size of shuffled data per round.

A MapReduce job of CTTP consists of a map step, a shuffle step, and a reduce step. We let $CostM(\mu, s)$, $CostS(\tau, s)$, and $CostR(\tau, s, K_r)$ be the cost of each step for *a round* where μ and τ are the numbers of mappers and reducers, respectively, s is the input data size of each step, and K_r is the number of subproblems solved in the job. $CostM(\mu, s)$ contains the cost for starting μ mappers and loading the input data into memory. It is formulated as follows:

$$CostM(\mu, s) = startUpCost(\mu) + \frac{s}{\mu} \cdot \frac{1}{D_s} \tag{2}$$

where D_s is disk I/O speed. $CostS(\tau, s)$ is the cost for transferring data between different machines. It is given by:

$$CostS(\tau, s) = \frac{s \cdot D_r}{\tau} \cdot \frac{1}{N_s} \tag{3}$$

where D_r is the ratio of data transferred between different machines and N_s is the network speed. $CostR(\tau, s)$ contains the cost for starting τ reducers and loading the data transferred from the shuffle step, and also contains the plugged in method's original time complexity. It is given by:

$$\begin{aligned} CostR(\tau, s, K_r) =& startUpCost(\tau) + \frac{s}{\tau} \cdot \frac{1}{D_s} + \\ & \frac{K_r}{\tau} \cdot plugInCost(\frac{s}{K_r}) \end{aligned} \tag{4}$$

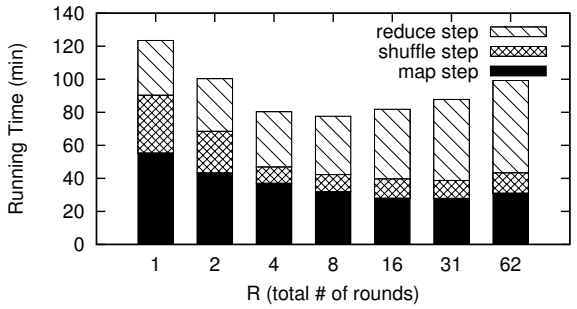


Figure 2: Average running time of each step. When $R < 4$, the running time of map and shuffle step is much longer than in the case of $R \geq 4$ while the running time of reduce step increases linearly as expected.

Then, the total cost of CTTP with R rounds is given by:

$$TotalCost(\mu, \tau, E, \rho, R) = R \times CostM(\mu, E) + R \times CostS(\tau, \frac{(\rho-1)E}{R}) + R \times CostR(\tau, \frac{(\rho-1)E}{R}, \frac{\rho(\rho^2-1)}{6R}) \quad (5)$$

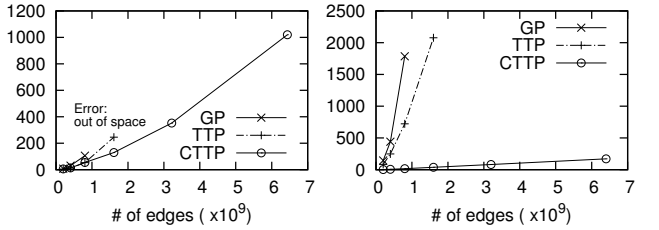
For each round, the whole input data are loaded during the map step. The shuffle step ships $(\rho-1)E/R$ data between different machines, and reducers receive $(\rho-1)E/R$ from the shuffle step. By applying Equations 2, 3 and 4 to Equation 5, we get the following cost:

$$\begin{aligned} &TotalCost(\mu, \tau, E, \rho, R) = \\ &\quad (R-1) \times (CostM(\mu, E) + startUpCost(\tau)) + \\ &\quad CostM(\mu, E) + CostS(\tau, (\rho-1)E) + \\ &\quad CostR(\tau, (\rho-1)E, \frac{\rho(\rho^2-1)}{6}) \end{aligned} \quad (6)$$

It indicates that the number of round affects only the factor $CostM(\mu, E) + startUpCost(\tau)$. In order to measure the running time of the factor, we design a *DoNothingJob* whose mappers load the input data but does not emit any data, and reducers are explicitly executed but do nothing. The cost of the job is exactly the factor of additional cost. The running time of the DoNothingJob on Twitter dataset is measured about 45 seconds on average.

Now we are ready to explain the result in Figure 1. Figure 1(a) shows the running time of CTTP with various round numbers from 1 to 62. If $R > 62$, fewer subproblems than reducers are processed; in this case, some reducer does not work and only wait for other reducers. The dotted line in Figure 1(a) shows the expected result from the cost function of Equation 6. With $R \geq 4$, the experimental result is almost the same as expected. However, the result shows much higher running time than our expectation with $R < 4$. The reason is that CTTP generates massive intermediate data in a round when R is small, so the data are overloaded to mappers in the map step and overburden the network in the shuffle step. Figure 2 explains the phenomenon well. When $R < 4$, the running time of map and shuffle step is much longer than the case of $R \geq 4$ while the running time of reduce step increases linearly as expected. Note that Figure 2 shows the average running time of each step, thus the sum of the running time in this figure differs a little from the real running time in Figure 1(a).

Figure 1(b) shows the shuffled data size of CTTP with various total round numbers. As we mentioned in Section 4.1, there is a trade-off relationship between the total number of rounds and the shuffled data size per round. When $R = 1$, the shuffled data size is 985GB; it is almost 30 times larger than the input file size (32GB). If the shuffled data is generated enormously in a round, the system will fail due to the lack of space and heap memory. CTTP can easily avoid the system failure by increasing the number of rounds while it takes little more time.



(a) Running time (min) (b) Shuffled data size per round (GB)

Figure 3: Data scalability of three algorithms. (a) shows the running time of CTTP, TTP and GP, and (b) shows the shuffled data size per round on random subgraphs with k edges of YahooWeb graph, varying k from 0.2 to 6.4 billion. Only CTTP successfully processed all datasets while GP and TTP failed to process datasets containing more than 1.6 billion edges. As the graph size increases, the shuffled data size of TTP and GP rapidly increases while that of CTTP increases linearly with the number of edges. The enormous shuffled data make TTP and GP fail due to the lack of space.

5.3.2 Data Scalability

Data scalability describes how the running time of an algorithm increases as data grow, and how large data can be processed by the algorithm. We compare and analyze CTTP to recent MapReduce triangulation algorithms, TTP and GP, in terms of data scalability. We run the three algorithms on random subgraphs with k edges of YahooWeb graph where k is the number of randomly selected edges from the original graph. For k, $6.4\times10^9, 3.2\times10^9, 1.6\times10^9, 8.0\times10^8, 4.0\times10^8$, and 2.0×10^8 are used.

The experimental results are depicted in Figure 3. It shows the running time of three algorithms and the shuffled data size per round on random subgraphs of YahooWeb graph. Only CTTP successfully processes all the subgraphs while GP and TTP fail to process datasets containing more than 1.6 billion edges. Figure 3(b) shows the reason of the failure well. As the graph size increases, the shuffled data size of TTP and GP rapidly increases, and they cause 'out of space' error when $k > 1.6 \times 10^9$. On the contrary, CTTP generates small amount of intermediate data whose size increases linearly with the number of edges; thus, it is able to process much larger datasets compared to TTP and GP.

5.3.3 Machine Scalability

Machine scalability is the degree of performance improvement when the number of machines increases. In order to evaluate CTTP and competitors with regard to machine scalability, we run the algorithms on the Twitter datasets varying the number of mappers and reducers from 10 to 80. The experimental results are depicted in Figure 4. It shows the speedup factors of three algorithm on the Twitter dataset. The speedup factor is defined as t_{20}/t_r, where t_r is

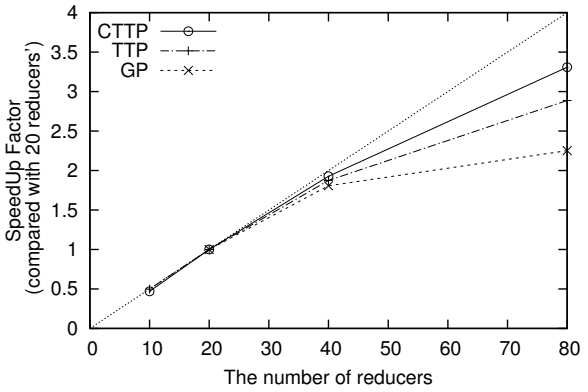
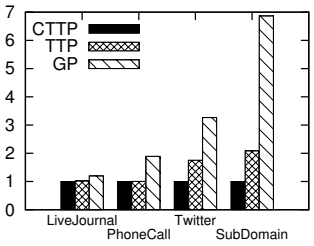


Figure 4: Machine scalability of three algorithms on the Twitter dataset. The dotted diagonal line is the optimum. CTTP is closest to the optimum among three algorithms. The speedup factor is defined as t_{20}/t_r, where t_r is the running time of an algorithm with r reducers.



(a) Running time ratio (/CTTP's)

Dataset	CTTP	TTP	GP
LiveJournal	1	1	1
PhoneCall	6	6	11
Twitter	92	162	302
SubDomain	218	455	1496
YahooWeb	1032	-	-
ClueWeb09	1328	-	-

(b) Running time (min)

Figure 5: The running time of all algorithms on all real-world datasets. (a) shows the relative running time to CTTP and (b) is the list of the running time. CTTP outperforms the other algorithms in all datasets and shows more than 2x faster performance in SubDomain dataset compared to TTP. YahooWeb and ClueWeb09 datasets are not presented in (a) because GP and TTP failed on the datasets.

the running time of an algorithm with r reducers. We set the reference point to the case of 20 reducers instead of 10 reducers since GP failed to run with 10 reducers because of the lack of space. The dotted diagonal line is the optimum and its slope is 1/20. The result shows that CTTP provides the best performance, closest to the optimum among three algorithms. Furthermore, the result implies that the performance gap will increase as the number of machines increases.

5.3.4 Performance on Real-world Datasets

Figure 5 presents the running time of all algorithms on all datasets. Figure 5(a) shows the relative running time to CTTP on four datasets where all algorithms run successfully, and Figure 5(b) shows the running time for every case. In CTTP, the number R of rounds is set to 1 on LiveJournal and PhoneCall datasets, and to ρ on the other datasets as we mentioned in Section 5.2. Only CTTP completed to enumerate all triangles in the YahooWeb and ClueWeb09 datasets; TTP and GP failed due to the lack of space. In this figure, we can see that CTTP outperforms the other algorithms on all datasets, and shows more than 2× faster performance with SubDomain dataset compared to TTP.

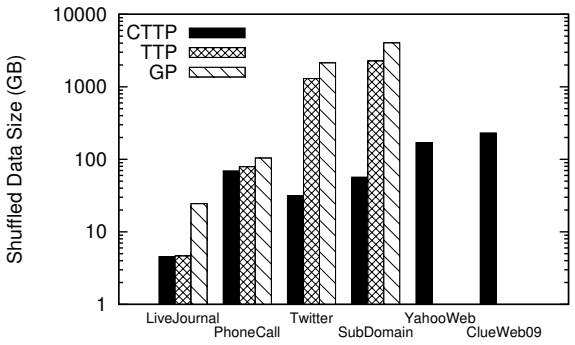


Figure 6: The shuffled data size of all algorithms on real-world datasets. In CTTP, the round number R is 1 on LiveJournal and PhoneCall datasets, and ρ on the other datasets as we mentioned in Section 5.2. CTTP generates the smallest amount of shuffled data per round. In contrast, TTP and GP generate a huge amount of shuffled data on Twitter dataset; and they failed to run on YahooWeb and ClueWeb09 datasets because of the enormous shuffled data.

Figure 6 shows the shuffled data size of all algorithms for real-world datasets. It indicates that TTP and GP generate a huge amount of shuffled data when the input graph is very large (e.g. Twitter and SubDomain graphs). The algorithms failed to run on YahooWeb and ClueWeb09 datasets because of the enormous intermediate data. The shuffled data sizes of CTTP on LiveJournal and PhoneCall dataset are similar with that of TTP because R is 1 on the datasets. On the other datasets, CTTP generates much smaller amount of the shuffled data compared to TTP and GP due to the effect of multiple round approach.

6. CONCLUSION

In this paper, we propose CTTP, a multi-round MapReduce randomized algorithm for triangle enumeration. CTTP requires $O\left(E^{3/2}/(M\sqrt{m})\right)$ rounds in the worst case, and uses M/E words per mapper, m words in expectation per reducer, and M words as total aggregate space. Moreover, the algorithm requires $O\left(E^{3/2}\right)$ total work in expectation, matching the best-known sequential algorithms. The claimed bounds are also shown to be optimal.

We further improve CTTP to get strong guarantees on the maximum load of each reducer with high probability. Specifically, we show that the aforementioned expected bounds on the space requirements and total work apply with probability at least $1-1/E^{\epsilon}$ when $m = \Omega\left(E^{3/4}\sqrt{\log E}\right)$. This result shows how to get rid with high probability of the "curse of the last reducer" [34], that is of an uneven distribution of the total work among the reducers.

Experiments show that CTTP outperforms the state of the art MapReduce algorithms on large real world graphs. Specifically, CTTP outperforms competitors by 2× on a phone call dataset with 30 million vertexes and 230 million edges; furthermore, CTTP enumerates 31 billion triangles of the ClueWeb09 graph with 4.8 billion vertexes and 7.9 billion edges in 23 hours, while competitors fail to run on the data.

Acknowledgment

This work was partly supported by the IT R&D program of MSIP/IITP. [10044970, Development of Core Technology for Human-like Self-taught Learning based on Symbolic Approach]. Silvestri is supported by the University of Padova under Project CPDA121378, and by MIUR of Italy under project AMANDA; this work was done while Silvestri was visiting the IT University of Copenhagen. Pagh is supported by the Danish National Research Foundation under the Sapere Aude program.

7. REFERENCES

[1] Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a Map-Reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, pages 16–24, 2008.

[3] Jonathan Berry, Luke Fostvedt, Daniel Nordman, Cynthia Phillips, C. Seshadhri, and Alyson Wilson. Why do simple algorithms for triangle enumeration work in the real world? In *ITCS*, 2014.

[4] Matteo Ceccarello and Francesco Silvestri. Experimental evaluation of multi-round matrix multiplication on mapreduce. Submitted (arXiv:1408.2858), 2014.

[5] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[6] Shumo Chu and James Cheng. Triangle listing in massive networks and its applications. In *KDD*, pages 672–680, 2011.

[7] Shumo Chu and James Cheng. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data*, 6(4):17:1–17:32, 2012.

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

[9] Robson Leonardo Ferreira Cordeiro, Caetano Traina Jr., Agma Juci Machado Traina, Julio López, U. Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *KDD*, pages 690–698, 2011.

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

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

[12] Facebook. Facebook newsroom: http://newsroom.fb.com/key-facts, 2014.

[13] Michael T. Goodrich and Pawel Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *ESA*, pages 664–676, 2011.

[14] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *ISAAC*, pages 374–383, 2011.

[15] Ronen Gradwohl and Amir Yehudayoff. t-wise independence with local dependencies. *Information Processing Letters*, 106(5):208 – 212, 2008.

[16] Hadoop information. http://hadoop.apache.org/.

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

[18] Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. arXiv:1305.3827, 2012.

[19] Svante Janson. Large deviations for sums of partly dependent random variables. *Random Structures & Algorithms*, 24(3):234–248, 2004.

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

[21] U Kang, B. Meeder, E. Papalexakis, and C. Faloutsos. Heigen: Spectral analysis for billion-scale graphs. *TKDE*, 26(2):350–362, February 2014.

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

[23] U Kang, Charalampos E. Tsourakakis, and Faloutsos Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *ICDM*, 2009.

[24] Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Math.*, 8(1-2):161–185, 2012.

[25] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600. ACM, 2010.

[26] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA*, pages 85–94, 2011.

[27] Robert Meusel, Oliver Lehmberg, Christian Bizer, and Sebastiano Vigna. Web data commons, http://webdatacommons.org/hyperlinkgraph/, 2014.

[28] Rasmus Pagh and Francesco Silvestri. The Input/Output complexity of triangle enumeration. In *PODS*, pages 224–233, 2014.

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

[30] Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. Space-round tradeoffs for MapReduce computations. In *ICS*, pages 235–244, 2012.

[31] Thomas Schank. Algorithmic aspects of triangle-based network analysis. *Phd Dissertation in computer science, University Karlsruhe*, 2007.

[32] Francesco Silvestri. Subgraph enumeration in massive graphs. Submitted (arXiv:1402.3444), 2014.

[33] SNAP. Stanford network analysis project, http://snap.stanford.edu/, 2012.

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

[35] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *SODA*, pages 615–624, 2004.

[36] Jia Wang and James Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

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

[38] ClueWeb09 Wiki. Clueweb12 web graph, http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php?page=web+graph, 2013.

[39] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.

[40] Yahoo! Webscope - yahoo! labs, http://webscope.sandbox.yahoo.com/, 2014.

[41] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. Uncovering social network sybils in the wild. In *Internet measurement conference*, pages 259–268. ACM, 2011.