Mansoura Journal of Computers and Information Scinces

Enhanced Algorithms for Counting Rectangles in Large Bipartite Graphs using MapReduce

Ahmed T. Sharafeldeen
Faculty of computers and information systems, C.S dep.
Mansoura University, Egypt
ahmed.taher@mans.edu.eg

Mohammed F. Alrahmawy
Faculty of computers and information systems, C.S dep.
Mansoura University, Egypt
mrahmawy@mans.edu.eg

Samir Elmougy
Faculty of computers and information systems, C.S dep.
Mansoura University, Egypt
mougy@mans.edu.eg

ABSTRACT
Rectangles for bipartite graphs are like triangles for unipartite graphs as both represent the smallest cycles in such graphs. *Rectangle Counting* is considered an important task in many bipartite network analysis metrics and is considered the core of computing such metrics, especially in cluster coefficient, bitruss, etc. However, there are few efficient algorithms to deal with this problem, especially in a large bipartite graph. In this work, we use MapReduce to enhance an algorithm to count rectangles in a large bipartite graph. The results show that our proposed MapReduce-based algorithm gives a better execution time than the existing algorithms, especially when it is applied in very large bipartite graphs.

Keywords
Rectangle Counting; Bipartite Graph; MapReduce; Large-Scale Graph Analysis.

1. INTRODUCTION
In the last few years, the amount of bipartite graphs has increased noticeably. A bipartite graph is two disjoint sets of nodes where the nodes in the first set are connected only to the nodes in the second set. Because of the huge amount of computations required to process such graphs, sequential algorithms can't be used to deal with such graphs due to the memory and CPU restrictions. Hence, developing parallel algorithms can be used to overcome this problem. Therefore, there is a need for new or enhanced algorithms based on parallel execution models such as MapReduce [1]. A bipartite network analysis uses rectangles counting as the core of computing such metrics; for example, computing cluster coefficient, bitruss, etc.

In this paper, we first present an enhanced version of the sequential algorithm presented in [2] to count the number of rectangles in bipartite graphs. Then, we use a parallelized version of this enhanced algorithm to develop a novel MapReduce-based algorithm in the case of large bipartite graphs. This proposed MapReduce-based algorithm divides a bipartite graph into sub-graphs following by counting rectangles in each sub-graph, where the rectangles in the sub-graphs are categorized into two different categories.

We evaluate our proposed MapReduce-based algorithm on a local machine then on a cluster of seventeen machines using several datasets of different sizes.

Section 2 discusses the related work. Section 3 explains how MapReduce model works. Section 4 defines the problem of rectangle counting. Section 5 presents our proposed algorithms. Section 6 shows and discusses experimental results. Finally, Section 7 discusses the conclusions of the paper and future work.

2. Related Work
The problem of counting the number of rectangles contained in bipartite graphs is somehow similar to the famous problem of counting triangles in unipartite graphs, as both problems aim to get the smallest cycles in such graphs. However, there are a relatively few research works for counting rectangles in a bipartite graph. Wang et al. [2] developed four algorithms in which two of them are sequential algorithms (In-Memory Rectangle Counting (IM-Rect) and I/O-Efficient Rectangle Counting (I/O-Rect)) and the others are parallel based algorithms (MapReduce Rectangle Counting (MR-Rect) and Partition-based parallel algorithm (PAR-Rect) based on Messaging Passing Interface (MPI)). Their experimental results showed that PAR-Rect is most efficient among the other three algorithms to count rectangles in a huge amount of bipartite graph. Some other works searches for rectangles in large graphs to solve bitruss decomposition problem, e.g. Peel algorithm proposed by Zou [3] in which it is similar to truss decomposition problem, with the difference that bitruss is based on rectangle while truss is based on triangle. This means that k-bitruss is a sub-graph of a bipartite graph in which each edge in the sub-graph contains k rectangles.

3. MapReduce Model
MapReduce is considered one of the most popular parallel and distributed models in the recent years [1]. Hadoop is an open source framework, and it is the most common framework for implementing MapReduce model [4]. MapReduce consists of three steps: map, shuffle and reduce. Map step is scripted by the programmer, and it is responsible for reading the input file, where each map reads only one line from the file to process the read data. This read data is divided into small chunks and these chunks are sent to the next step. The input and the output of the map step is represented as $\langle key; value \rangle$. The shuffle step aggregates the output chunks of the map step into groups, where each group contains the values that have the same key, then the results of aggregation step are sorted

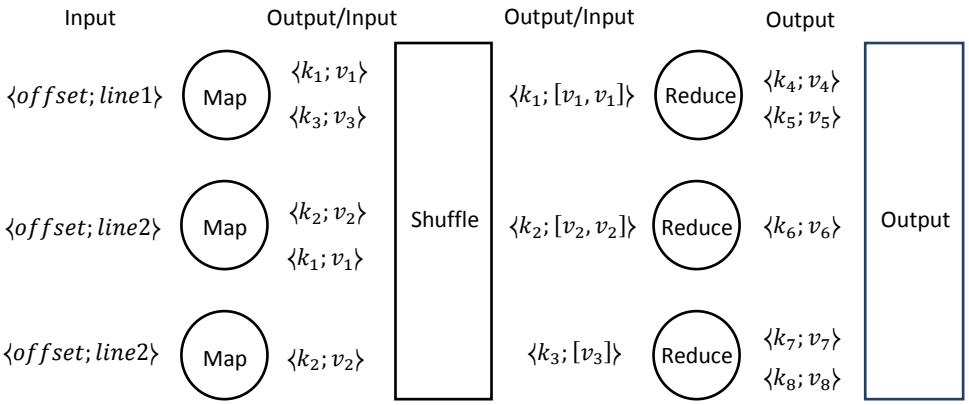

Fig 1: MapReduce Example

by their keys and the results are sent to reduce step. The shuffle step is usually done automatically by the framework implementing the MapReduce model. In the reduce step, each reduce instance takes one group of the shuffle as input and apply the required processing on it, and finally the results are saved. Figure 1 shows how MapReduce model works.

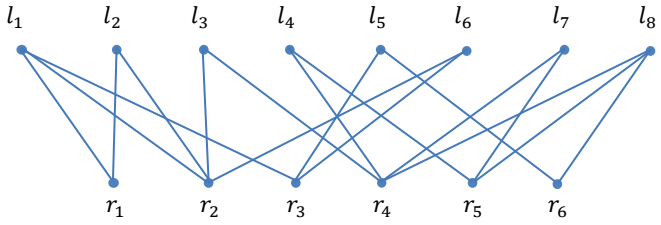

Fig 2: Graph input

4. Problem Definition

Table 1 shows the main terms used in this paper. Suppose that $G(L, R, E)$ is an undirected bipartite graph where L is a set of its left vertices, R is a set of its right vertices (i.e. $V = L \cup R$), E is a set of edges between the left and the right vertices, $n = |L|$, and $m = |E|$. Also, let $\tau(l) = \{r: (l, r) \in E\}$ is a set of the neighbors of a vertex $l \in L$, and let $\mathrm{d}(l) = |\tau(l)|$ is the degree of vertex l.

A rectangle can be defined as the smallest cycle exists in bipartite graph, denoted by $\square(l, r, l', r')$ such that $l, l' \in L$ and $r, r' \in R$, and it contains the following edges: (l, r), (l', r), (l, r'), and (l', r').. An example of bipartite graph is shown in Figure 2. In this example $L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\}$, $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$, and $E = \{(l_1, r_1), (l_1, r_2), (l_1, r_3), (l_2, r_1), (l_2, r_2), (l_3, r_2), (l_3, r_4), (l_4, r_4), (l_4, r_5), (l_5, r_3), (l_5, r_6), (l_6, r_2), (l_6, r_3), (l_7, r_4), (l_7, r_5), (l_8, r_5), (l_8, r_6)\}$. The number of the rectangles founded in G is five: $\square(l_1, r_1, l_2, r_2)$, $\square(l_1, r_3, l_6, r_2)$,$(l_4, r_4, l_7, r_5)$,$\square(l_4, r_4, l_8, r_5)$, and $\square(l_7, r_5, l_8, r_4)$ We focus in this paper on the problem of identifying and counting rectangles in a bipartite graph.

5. The Proposed Algorithms

We proposed two algorithms to count the number of rectangles in a bipartite graph. The first one is a modified version of a sequential algorithm called In-Memory algorithm, denoted by IM-Rect [2]. The second algorithm is based on a MapReduce model which counts the number of rectangles in a large bipartite graph. The two algorithms are presented in Section 5.1 and Section 5.2, respectively.

5.1 The First Algorithm

In IM-Rect [2], each rectangle has two left vertices in L, so it counts the same rectangle twice from those two left vertices of the rectangle. For example, when applying IM-Rect on the graph in Figure 2, the two rectangles $\square(l_1, r_1, l_2, r_2)$ and $\square(l_2, r_1, l_1, r_2)$ are identified by IM-Rect, although they refer to the same rectangle. To overcome this problem, we present In-Memory++ algorithm (IM++RECT) which is an enhanced version of the IM-Rect in [2] and its pseudocode is in listing *Algorithm 1*.

Table 1. Notations used in this paper

Notation	Description
$G(L, R, E)$	Undirected bipartite graph
L	A set of left vertices in bipartite graph
R	A set of Right vertices in bipartite graph
E	A set of edges between left and right vertices in bipartite graph
n	Number of left vertices
m	Number of edges
(l, r)	Is an edge, where $l \in L$ and $r \in R$
$\tau(l)$	Set of neighbors of a vertex l
$\tau(l, l')$	$\tau(l) \cap \tau(l')$
$\mathrm{d}(l)$	Number of neighbors of a vertex l i.e. $d(l) = \|\tau(l)\|$
$\mathrm{d}(l, l')$	$\|\tau(l, l')\|$
$2path(l)$	Set of left vertices l' away two edges from l
ρ	Number of partitions
$P(l)$	Partition number of left vertex l
$\square(l, r, l', r')$	Rectangle; $(l, r), (r, l'), (l', r'), (r', l) \in E$
$G_i(L_i, R_i, E_i)$	1-partition bipartite sub-graph with L_i left vertices, R_i right vertices and E_i edges
$G_{ij}(L_{ij}, R_{ij}, E_{ij})$	2-partition bipartite sub-graph with $L_{ij} = L_i \cup L_j$, $R_{ij} = R_i \cup R_j$ and $E_{ij} = E_i \cup E_j$, where $j > i$
$\langle key; value \rangle$	Is a set of key and value

The proposed algorithm assumes that each vertex in the graph has a unique id based on its category (left vertices or right vertices). Let $\succ$ be a total order of vertices, i.e. $u \succ v$ means that u's id is greater than v's id. IM++RECT avoids IM-Rect problem by using the total ordering to count each rectangle only once. The algorithm uses two sets $2path(l)$ and $\tau(l, l')$, where $2path(l)$ is the set of the left vertices l', where l' is two edges away from vertex l and $\tau(l, l')$ is the set of all common neighbors for both vertex l and l'. For example, in Figure 2, $2path(l_1) = \{l_2, l_3, l_5, l_6\}$ and $\tau(l_1, l_2) = \tau(l_1) \cap \tau(l_2) = \{r_1, r_2\}$.

IM++RECT consists of two parts. The first part finds both $2path(l)$ and $\tau(l, l')$, for each $l \in L$ and $l' \in L - \{l\}$ as shown in Lines 5-8. Then, in the second part, the number of rectangles is counted by iterating on $2path(l)$ as shown in Lines 9-13.

Algorithm 1: IM++RECT algorithm

```
1   R ← 0
2   for l ∈ L do
3       2path(l) ← ∅;
4       τ(l, l') ← ∅ , l' ∈ L − {l};
5       for r ∈ τ(l) do
6           for l' ∈ τ(r) − {l} and l' ≻ l do
7               τ(l, l') ← τ(l, l') ∪ {r};
8               2path(l) ← 2path(l) ∪ {l'};
9       for l' ∈ 2path(l) do
10          for s ∈ [1, d(l, l')] do
11              for t ∈ [s + 1, d(l, l')] do
12                  R ← R + 1
13                  Print (l, τ(l, l')[s], l', τ(l, l')[t]);
```

5.1.1 *Analysis*

Lemma 1. IM++RECT algorithm counts rectangles in a bipartite graph correctly and only once.

Proof. For each left vertex $l \in L$, IM++RECT computes $2path(l)$ only when $l' \succ l$, for each $l' \in L$. Thus, it is guarantee that each rectangle in bipartite graph is seen only once.

Lemma 2. IM++RECT algorithm takes $O\left(d^2(r)d^2(l, l')\right)$

Proof. Line 1 takes $O(1)$, Lines 2-4 take $O(n)$ (Line 4 is defined as list of keyValue pairs), Line 7-8 takes $O\left(d^2(r)\right)$ (i.e. in total, number of all possible two paths which equals to $\binom{d(r)}{2} = O\left(d^2(r)\right)$), Line 9 takes $O\left(d^2(r)\right)$ (i.e. in total, the number of all possible two paths), Lines 12-14 takes $O\left(d^2(r)d^2(l, l')\right)$. Therefore, IM++RECT takes:

$$1 + n + d^2(r) + d^2(r) + d^2(r)d(l, l')$$
$$= 1 + n + 2d^2(r) + d^2(r)d^2(l, l')$$
$$\leq n + n + 2d^2(r) + d^2(r)d^2(l, l'), \quad \text{for } n \geq 1$$

Equation (1)

Since $d(r) \geq n$, then using Equation (1),

$$1 + n + d^2(r) + d^2(r) + d^2(r)d^2(l, l')$$
$$= O\left(d^2(r)d^2(l, l')\right)$$

5.2 The Second Algorithm

Our second proposed algorithm is called **R**ectangle counting in **B**ipartite graphs using **P**artitioning (RBP) algorithm based on MapReduce model to count rectangles in a large bipartite graph. The main idea of RBP is to divide a large bipartite graph into ρ sub-graphs with equal number of left vertices. The algorithm is shown in listing *Algorithm 2*. Before we present the algorithm, we define some terms required to understand the algorithm. In our work, any rectangle can be categorized either as *Type-1* or *Type-2*, where:

Type-1: two left vertices l, l' of the rectangle exist in the same partition, e.g. $\square(l_1, r_1, l_2, r_2)$ shown in Figure 3.

Type-2: two left vertices l, l' of the rectangle exist in different partitions, e.g. $\square(l_4, r_4, l_7, r_5)$ shown in Figure 3.

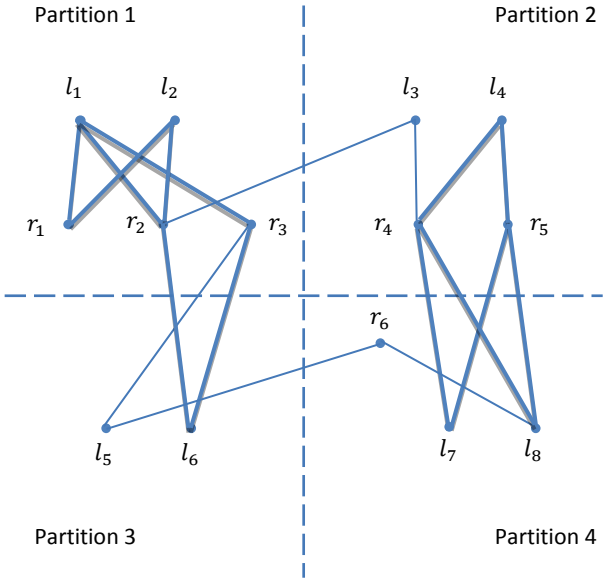

Fig 3: Partitioning of the graph presented in Figure 2 for the RBP algorithm, $\rho = 4$

Also, the partitioning of a bipartite graph can be one of two types: *1-partition* or *2-partition*. These two types are defined as follow:

1-partition: 1-partition graph is a sub-graph of bipartite graph denoted by $G_i = (L_i, R_i, E_i)$ for $i \in [1, \rho]$; where the partition number of left vertex, $P(l)$, of each edge in such graph equals to i (i.e. $P(l) = i$). For example, for $\rho = 4$, 1-partition sub-graphs of the bipartite graph shown in Figure 3 are (G_1, G_2, G_3, G_4) as shown in Figure 4.

2-partition: 2-partition graph is denoted by $G_{ij} = \left(L_{ij}, R_{ij}, E_{ij}\right)$ for $1 \leq i < j \leq \rho$, which is a sub-graph of a bipartite graph, with the partition number of left vertex, $P(l)$, of each edge in such graph equals to i or j (i.e. $P(l) \in \{i, j\}$). For example, for $\rho = 4$, 2-parition sub-graphs of bipartite graph shown in Figure 3 are $(G_{12}, G_{13}, G_{14}, G_{23}, G_{24}, G_{34})$ which are shown in Figure 5. In general, it is easily proofed that for any bipartite graph divided into ρ sub-graphs, there are $\binom{\rho}{2}$ 2-partition sub-graphs.

When a bipartite graph is divided, Type-1 rectangles can be deduced from both 1-partition and 2-partition graphs, while type-2 rectangles can be deduced only from 2-partition graphs. Therefore, RBP divides a bipartite graph only in 2-partition graphs as shown in the Map function in Lines 1-4. After finishing the Map step, MapReduce model combines each group of outputs of the map step that have the same key

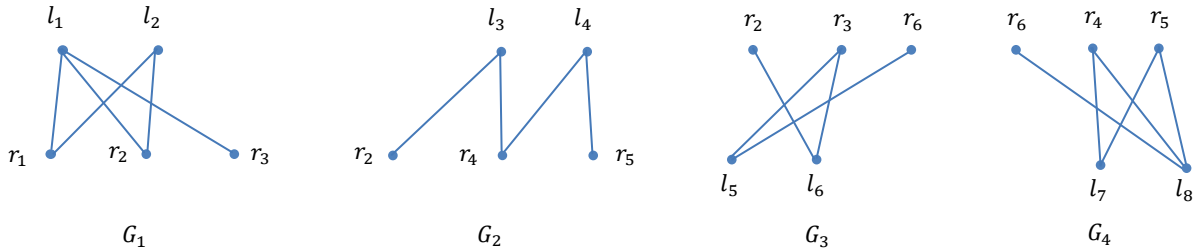

Fig 4: 1-Partition sub-graphs of the graph presented in Figure 3

(i.e. combines edges in the same sub-graph) as mention before in section 3, then the reduce step counts the rectangles in the combined edges (i.e. 2-partition sub- graphs) as shown in Lines 5-24.

Algorithm 2: The proposed RBP algorithm

```
    Map : input ⟨ ∅ ; (l, r) ⟩
1     for i ∈ [1, ρ − 1] do
2       for j ∈ [i + 1, ρ] do
3         if P(l) ∈ {i, j} then
4           emit ⟨ (i, j) ; (l, r) ⟩;
    Reduce : input ⟨ (i, j) ; E_ij ⟩
5     Parallel for l ∈ L_ij do
6       2path(l) ← ∅;
7       τ(l, l′) ← ∅ , l′ ∈ L_ij − {l};
8       for r ∈ τ(l) do
9         for l′ ∈ τ(r) − {l} and l′ ≻ l do
10          τ(l, l′) ← τ(l, l′) ∪ {r};
11          2path(l) ← 2path(l) ∪ {l′};
12      Parallel for l′ ∈ 2path(l) do
13        for s ∈ [1, d(l, l′)] do
14          for t ∈ [s + 1, d(l, l′)] do
15            r = τ(l, l′)[s], r′ = τ(l, l′)[t];
16            if P(l) = P(l′) then
17              // Type-1
18              if (P(l) = i or P(l) = ρ)
                  and (j = i + 1) then
19                lock
20                  emit ⟨ (l, r, l′, r′) ; 1 ⟩;
21            else
22              // Type-2
23              lock
24                emit ⟨ (l, r, l′, r′) ; 1 ⟩;
```

The function in the Reduce step parallelizes the IM++RECT algorithm to increase the performance of the algorithm. We notice that Type-1 rectangles are duplicated in all 2-partition graphs, due to partition number of two left vertices (i.e. $P(l)$) belong to one site of 2-partition graph. For example, $\square(l_1, r_1, l_2, r_2)$ in Figure 3 is a Type-1 rectangle, $P(l_1) = P(l_2) = 1$, which appears in G_{12}, G_{13}, G_{14} shown in Figure 5. So, to overcome the duplication problem, Lines 16-20 deal with this problem by only seeing Type-1 rectangle once in G_{ij}, where i equals to the partition number of left vertex of the rectangles (i.e. $P(l) = i$) and j is assigned to $i + 1$ (first condition) if a rectangle doesn't exist in the last partition. For example, $\square(l_1, r_1, l_2, r_2)$ is identified only in G_{34} show in Figure 5. If a rectangle exists in the last partition (i.e. $P(l) = \rho$), then the rectangle is identified only in the last sub-graph (i.e. $j = i + 1$). For example, $\square(l_7, r_4, l_8, r_5)$ is identified only in the last 2-partition graph G_{34}. Lock mechanism is used in Line 19 and Line 23 to deal with the critical section and to overcome race condition that occurs when multiple iterations write their results to the same file at the same time.

5.2.1 *Analysis*

Lemma 3. Type-1 and Type-2 rectangles are counted only once in a bipartite graph by RBP.

Proof. RBP sees Type-1 rectangles only in the first 2-partition graph G_{ij} ($i = P(l)$ and $j = i + 1$), or in the last 2-partition graph; if Type-1 rectangle is existed in the last partition ($P(l) = \rho$ and $j = i + 1$). So, Type-1 rectangles are counted only once by RBP.

On the other hand, Type-2 rectangles appear only one time in 2-partition graphs. Therefore, RBP counts the rectangles correctly and only once in a bipartite graph.

Lemma 4. Output of all map instances is $(\rho - 1)m = O(m\rho)$.

Proof. Each map instance takes an edge (l, r) as input and classifies it to 2-partition graph G_{ij}, when the partition number of a left vertex l belongs to any of two site of sub-graph (i.e. $P(l) \in \{i, j\}$). Hence, any edge in a bipartite graph appears in $\rho - 1$ sub-graphs. For example, an edge (l_1, r_1) in Figure 3 is shown in G_{12}, G_{13}, and G_{14} as shown in Figure 5. Therefore, for m edges, they appear $(\rho - 1)m$. So, the output of all map instances is:

$$(\rho - 1)m = O(m\rho)$$

Lemma 5. Input of each reduce instance is $O\left(\frac{m}{\rho}\right)$.

Proof. The probability that each edge belongs to a specific partition is $\frac{1}{\rho}$. Therefore, probability of the existence of an edge in 2-partition is $\frac{1}{\rho} + \frac{1}{\rho} = \frac{2}{\rho}$. The probability of m edges in 2-partition is $\frac{2m}{\rho}$; so the input of each reduce instance is $\frac{2m}{\rho} = O\left(\frac{m}{\rho}\right)$.

Lemma 6. Reduce instance takes $O\left(k^2 + d^2(l, l')\right)$, where k is the maximum degree.

Proof. Line 5 takes $O(\lg n +$ running times of Lines 6-24), Lines 6-7 take $O(1)$, Lines 10-11 take $\tau(l)\tau(r) = k^2$ (Assume k is the maximum degree, and in the worst case $k \geq \tau(l)$ and $k \geq \tau(r)$), Line 12 takes $O(\lg k^2 +$ running time of Lines 13-24), Lines 15-24 take $O\left(d^2(l, l')\right)$. Therefore, reduce instances takes:

$$\lg n + 1 + n + k^2 + \lg k^2 + d(l, l')$$
$$\leq \lg n + n + n + k^2 + \lg k^2 + d^2(l, l'), \text{ for } n \geq 1$$

i.e. $\lg n + 1 + n + k^2 + \lg k^2 + d^2(l, l')$
$$\leq \lg n + 2n + k^2 + \lg k^2 + d^2(l, l')$$
$$= O\left(k^2 + d^2(l, l')\right)$$

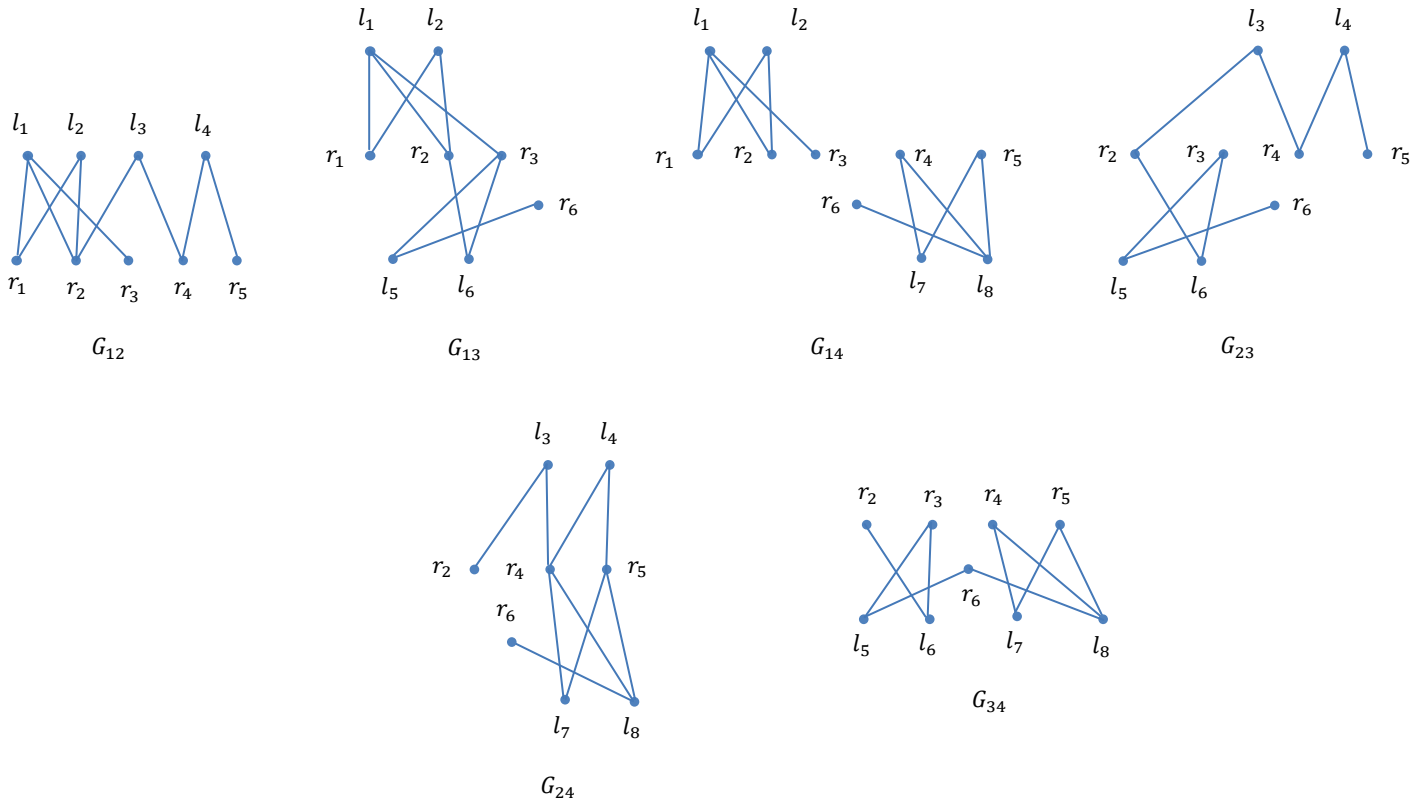

Fig 5: 2-Partition sub-graphs of the graph presented in Figure 3

6. Experimental Results

In this section, we describe the set of experimental results conducted to evaluate our proposed algorithm with comparing it with PAR-Rect algorithm [2]. The experiment is divided into two sets. The first set is used to run the two algorithms on one machine, while the second set is used to run the two algorithms on multi machines. The characteristics of the datasets used in the experiments are shown in Table 2 [5].

6.1 Evaluation on a Single Machine

In the first set of experiments, our proposed algorithm and the PAR-Rect were run and tested on a single machine with Intel Core i5 2.67GHz, 5.8 GB RAM and running Ubuntu. Apache Hadoop and MPJ Express (i.e. Messaging Passing Interface library for java) framework are running on this machine. The two algorithms were run on this machine with $\rho = 20$. Table 3 shows the results, where it shows that the performance of our proposed algorithm beats PAR-Rect algorithm. PAR-Rect can't output the result of Youtube dataset, and gives an error (i.e. insufficient memory), when running in MPJ Express cluster configuration with one machine. On the other hand, RBP gives result for this dataset (12.35 minutes) as shown in this table. Therefore, it is clear that our proposed MapReduce-based algorithm has a better performance than PAR-Rect and it can handle large graphs with a good performance even in small clusters.

We also evaluated the effect of number of partitions on the running times of RBP and PAR-Rect algorithms on a single machine from 5 to 60 partitions using Writer dataset, as shown in Figure 8 in which it shows that RBP is more efficient than PAR-Rect.

Table 2. Characteristics of the datasets used in the experiments

Dataset	# Left Nodes	# Right Nodes	# Edges	# Rectangles
Collaboration	16,726	22,015	58,595	70,549
Writer	89,355	46,213	144,340	126,753
Producer	48,833	138,839	207,268	266,983
Starring	76,098	810,85	281,396	221,363
YouTube	94,238	30,087	293,360	12,540,261

Table 3: Running times of all algorithms on a single machine (min)

Dataset	PAR-Rect	RBP
Collaboration	0.76	0.82
Writer	11.02	3.37
Producer	7.95	7.33
Starring	42.85	9.32
YouTube	*out-mem*	12.35

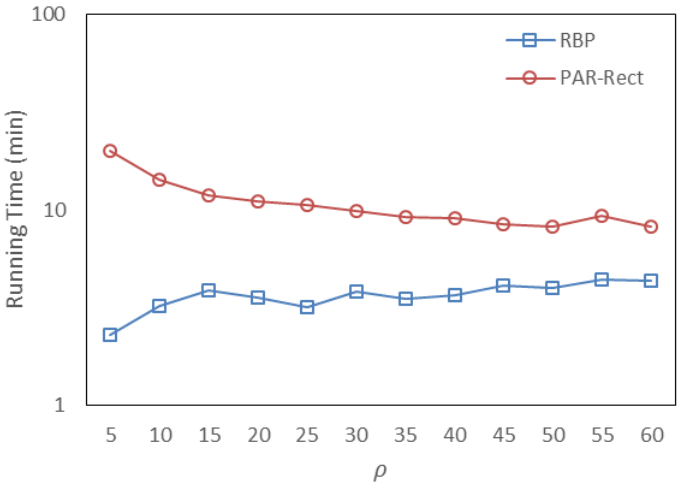

Fig 8: Running times of PAR-Rect and RBP on a single machine using Writer dataset with different ρ sizes

6.2 Evaluation on Multiple Machines

In the second set of experiments, the proposed algorithm and PAR-Rect were run and tested on a cluster of 17 machines (one master, sixteen slaves); each one of these machines has Intel core 2 Quad 2.83GHz, 3.7 RAM, and running Ubuntu. Also, each machine runs Apache Hadoop and MPJ Express. We tested the two algorithms on those machines with the mentioned datasets. The results of experiments are show in Table 4. We applied PAR-Rect only on four datasets because it takes too long time to partition and distribute large graph to slave's machines. As shown in Table 4, RBP beats PAR-Rect in counting the number of rectangles in large bipartite graphs.

We also evaluated the effect of number of partitions on the running times of RBP and PAR-Rect on multi machines using Writer dataset with several values of $\rho = 5$ up to $\rho = 60$, as shown in Figure 9. The figure shows clearly that RBP beats PAR-Rect in the performance. Also, the running time of RBP is almost constant when using different values of ρ.

Table 4: Running times of all algorithms on a multi node (min)

Dataset	PAR-Rect	RBP	RB2PL
Collaboration	0.32	0.35	0.47
Writer	2.83	0.63	0.6
Producer	5.89	1.73	0.48
Starring	14.17	2.33	0.55

7. CONCLUSIONS and Future Works

Since, rectangle counting is a fundamental problem in analyzing large bipartite graphs, we enhanced an existing sequential algorithm and proposed a MapReduce-based algorithm. Our experimental results showed that the proposed MapReduce-based algorithm gave better execution time in most cases than the existing algorithms especially for very large graphs. In the future work, we plan to implement a MapReduce model that uses MPI and develop more enhanced and efficient algorithms for extracting and counting shapes from very large scale bipartite and general graphs.

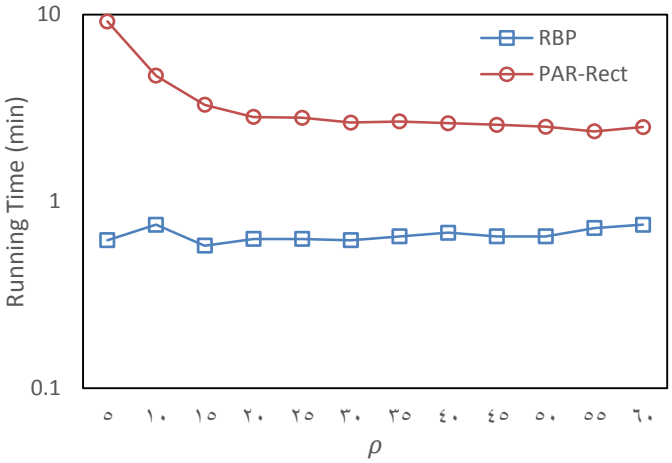

Fig 9: Running times of PAR-Rect and RBP on multi machines using Writer dataset with different ρ sizes

8. REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation.

[2] Jia Wang, Ada Wai-Chee Fu, James Cheng. 2014. Rectangle counting in large bipartite graphs. In Proceedings of the 2014 IEEE International Congress on Big Data, 17–24.

[3] Zhaonian Zou. 2016. Bitruss Decomposition of Bipartite Graphs. In Proceedings, Part II, of the 21st International Conference on Database Systems for Advanced Applications, 218-233.

[4] Apache Hadoop. http://hadoop.apache.org/.

[5] Graph datasets. http://konect.uni-koblenz.de/ (September 19, 2017).