I.J. Information Engineering and Electronic Business, 2017, 2, 36-44
Published Online March 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2017.02.05

Map-Reduce based Multiple Sub-Graph Enumeration Using Dominating-Set Graph Partition

Fathimabi shaik, RBV Subramanyam, DVLN Somayajulu
Department of Computer Science and Engineering, National Institute of Technology, Warangal, India.
Email: fathimanitw@gmail.com, rbvs66@gmail.com, soma@nitw.ac.in

Abstract—The purpose of this paper is to find all the instances of a given set of pattern graphs (sub-graphs) in a large data graph using a single round of Map-Reduce. For the simplest pattern graphs like a triangle and rectangle we propose the solution. This paper enumerates complex pattern graphs using the enumeration of simple pattern graphs. We proposed Dominating set based graph partition, it generates non-overlapped sub-graphs. Each sub-graph is processed by one machine in the cluster. We analyze both the communication cost and the total computational cost. Communication cost is reduced by using Map-Reduce based dominating set graph partition. At the same time Multiple pattern (sub-graphs) graphs can be enumerated with the same communication cost. The proposed method is not always superior to the conventional sub-graph enumeration, but in some cases involving large-scale data where this method wins, including (1) Adjacency list representation of the graph is the input (2) Number of partitions are decided based on the graph size. We experimentally show that our approach decreases significantly the computation cost, communication cost and scales the enumeration process with a large graph database.

Index Terms—Big Graph Processing, Map-Reduce, Graph Partition, Triangle Listing, Sub-Graph Enumeration.

I. Introduction

Graphs are used in many applications ranging from computer networks, social networks to Bioinformatics, Chem-informatics and others. These fields use graphs to describe their associated data, e.g., social networks consist of individuals and their relationships. In Bioinformatics, the protein structure is represented as a graph where amino acids represent as nodes and the interactions between them are represented as the edges. Finding repetitive and frequent sub-structures may give important insights on the data under consideration. These substructures may correspond to important functional fragments in proteins such as active sites, feature positions and junction sites.

Graph databases are categorized into two types: a graph transaction setting and a single graph setting. In the graph transaction setting a graph database consists of a large number of relatively smaller graphs, whereas in the single-graph setting the data of interest is a single graph. In this paper, we focus on the single graph which contains millions of vertices and edges.

Sub-graph enumeration plays an important role in graph query process. First, we have to enumerate simple pattern graphs. By using these simple pattern graphs enumeration information, we can enumerate complex pattern graphs. Now-a-days graph size is increased to millions of nodes and edges. To process this type of graphs centralized approaches are taking more time and requires more memory. Instead of using high-end systems, we can process using normal commodity systems and Hadoop MapReduce. Instead of scale up Hadoop uses a scale out approach.

Map-Reduce has been widely adopted by many industries and academicians. The features of Map-Reduce are scalability, simplicity, flexibility and fault tolerance. The data centric approach is adopted by the Map-Reduce with the idea of moving computation to data. It uses Hadoop Distributed File System(HDFS) particularly optimized to improve the IO performance while handling massive data. Moreover Map-Reduce framework hides the internal details. Therefore, it is an excellent platform for large graph processing. The paradigm is using two methods map and reduce. We can process large amount of data by using a number of machines in the cluster.

In this study multiple sub-graphs are enumerated simultaneously. Enumerating single sub-graph at a time is expensive process. So, instead of enumerating single sub-graph at a time, this paper enumerates multiple sub-graphs at the same time. All the resources are optimally utilized by enumerating multiple sub-graphs. With the same computation cost this study enumerates multiple sub-graphs.

In this paper, we focused on a solution to enumerate multiple sub-graphs in a big graph using Map-Reduce. To enumerate multiple sub-graphs from a big graph by using Map-Reduce there are different challenges, one main challenge is Graph partition. Our main objective is to design a graph partition. This paper generates non overlapped sub-graphs by using vertex based graph partitions. Here it uses Dominating set and extended graph. The advantages of non overlapping partitions are to reduce the communication cost during enumeration.

The other reason is to reduce the number of Map-Reduce rounds. This is called adaptive approach and wherever it is possible, we use centralized approach and the enumeration is done by using only one map-reduce round.

In this paper, the following are our contributions

1. Dominating set based graph partition using extended graph.
2. Non overlapped graph partitions to reduce the redundancy of data at the nodes in the cluster.
3. Enumeration of multiple sub-graphs using one Map-Reduce round.
4. We empirically demonstrate the performance of extended graph partition on synthetic as well as a real world large graph.

This paper is organized as follows: in the first section, we provide related work. We define the problem of enumerating multiple pattern graphs in a single large graph in section 3. We presented our proposed approach in section 4. We discussed the experimental study and obtained results in Section 5. In Section 6 we conclude the work.

II. Related Work

In this section, we present an overview of HDFS and Map-Reduce and the sub-graph enumeration algorithms. The algorithms are divided into two groups.

A. Hadoop Map-Reduce

Map-Reduce[25] is the most popular parallel computing paradigms for big data processing. Map-Reduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. It gained a lot of attention from both of industry and academia. Map-Reduce provides a distributed platform to process data intensive job with no hassle of managing the jobs across nodes. It adopts a data centric approach of distributed computing with the idea of moving computation to data. It uses a distributed file system and it hides the higher level details from the developer which allows them to concentrate more on the problem-specific computational logic. Fig. 1 shows the data flow of map and reduce functions.

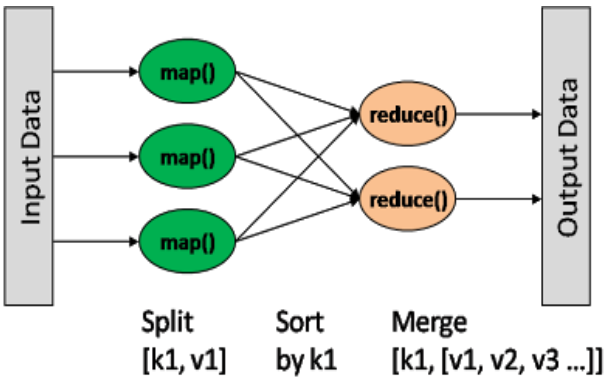

Fig.1. Example of Map-Reduce

B. Centralized Algorithms

In the 1990's there were good algorithms (1) To find the cycles of a given length and/or (2) Count the cycles of a given length[1]. The generalization of other sample graphs appears in [2]. These problems reduce to matrix multiplication. However, enumeration of all instances of a given sub-graph cannot be easily reduced. Probabilistic counting of triangles was discussed in [3]. More recently, there has been significant improvement in probabilistic counting of small sample graphs on large biological and social networks. The triangle listing in massive networks and dominating set based graph partition is proposed in [11]. All these algorithms are centralized algorithms that means they use only one machine to solve the problem.

C. Parallel Algorithms Using Map-Reduce

There are a number of algorithms designed using MapReduce. Enumeration of triangles using Map-Reduce has received attention recently. It was the thesis by Schank[4]. Suri and Vassilvitskii [5] which gave one and two rounds of Map-Reduce algorithms for finding triangles using overlapped sub-graphs. This overlapped graph partition increases the overhead communication cost.

In [6] Map-Reduce based triangle finding problem was experimentally studied and implementation of a randomized counting algorithm for triangles. In [7] sampling based technique for triangle enumeration using Map-Reduce was proposed. Enumeration of sub-graph instances using Map-Reduce was discussed in a paper [8] uses multiway join [9] to enumerate triangles, square etc. This multiway join partition the graph into overlapped partitions and it uses a single round of MapReduce. Because of overlapped partitions the communication cost is more. These graph partition algorithms generates a large number of intermediate data which causes network overloads and increases the processing time.

In [12] they proposed an efficient Map-Reduce algorithm for counting triangles in a very large graph, it classified the triangles into three types based on the availability of edges. It reduces the overlap of the partitions by using triangle types. [10] Scalable Sub-graph Enumeration in Map-Reduce, a left-deep-join framework that generalizes the edge-based joining to allow the right join argument to be a star (a tree of depth) rather than a single edge in each round. Joining a star is also inefficient. They propose the TwinTwigJoin, which uses a simple TwinTwig (an edge or two incident edges of a node) as the right join argument in each round. TwinTwigJoin, as a tradeoff between edge-based join and star-based join, has several advantages. TwinTwing Join is left-deep-join framework to join multiple edges in each round.

All these existing works are enumerating one sub-graph at a time, but in a distributed environment loading big graph and enumerating one sub-graph at a time is not at all optimum. Number of sub-graph queries are coming from different sources in the network. So we require efficient algorithm to process a number of pattern graphs at a time. We have to optimally utilize the resources by

enumerating multiple sub-graphs at a time. With the same computation and communication cost we can process number of pattern graphs.

In this paper, we designed a non overlapped graph partition, suitable for multiple sub-graph enumeration.

III. Problem Definition

Sub-graph enumeration, which aims to find all the sub-graphs of a large data graph that are isomorphic to the given set of pattern graphs, is a fundamental graph problem with a wide range of applications.

Sub-graph Enumeration: We model a data graph as an undirected and unlabeled graph G(V;E), where V=V(G) represents the set of nodes and E= E(G) represents the set of edges each of which connects two nodes in V(G). Let |V(G) |=N and |E(G)|=M, and assume M > N. We use $\{ u_1,u_2,.....,u_N\}$ to denote the set of nodes in G. For each $u_i \in V(G)$, we use $N(u_i)$ to denote the set of neighbor nodes of u_i, and we use $d(u_i)$ to denote the degree of u_i, i.e., $d(u_i) = |N(u_i)|$ and $d_{max}= \max_{ui \in V(G)} d(u_i)$. We define d=2M/N to be the average degree of the data graph.

The following are the definitions required for this paper:

Definition 1: (Sub-graph) A sub-graph s of G is a graph such that $V(s) \subseteq V(G)$, $E(s) \subseteq E(G)$.

Definition 2: (Graph Isomorphism) Given two graphs G_i and G_j are isomorphic, if and only if there exists a bijection between the vertex sets of G_i and G_j $f: V(G_i) \rightarrow V(G_j)$ such that any two vertices x and y of G_i are adjacent in G_i if and only if f(x) and f(y) are adjacent in G_j.

Definition 3: (Sub-graph Enumeration) Given a pattern graph P and a data graph G, sub-graph enumeration is to enumerate all sub-graphs s of G such that s is isomorphic to P.

Definition 4: (Dominating Set of a Graph) A dominating set of a graph G is a subset of vertices $D \subseteq VG$ such that every vertex in G is either in D or a neighbor of some vertex in D. Dominating Set(G).

Definition 5: (Extended Sub-graph) Let H = (VH , EH) be a sub-graph of G. An extended sub-graph of H in G, denoted by H+, is a extended sub-graph defined as H+ = (VH+,EH+), where $VH+ = VH \cup \{v : u \in VH, v \in VG\backslash VH, (u ,v)$ belongs to EG$\}$ and $EH+ = \{(u, v) : (u, v) \in EG$, u belongs to VH$\}$.

Definition 6 : (Extended edge) If we divide the Graph G into number of g1,g2,g3 sub-graphs, If the edge whose vertices belongs to two different sub-graphs, let E ={(u,v), u belongs to g1 and v belongs to g2 or g3 or any other sub-graph other than g1 } then that edge is called extended edge.

IV. Proposed Approach

In this section, we introduced a Dominating set based graph partition framework for sub-graph enumeration in Map-Reduce. For a given data graph G and a set of pattern graphs P, sub-graph enumeration is processed in a single Map-Reduce round. The entire work is divided into two phases : Preprocessing step and Enumeration of multiple sub-graphs using one Map-Reduce round.

A. Pre-Processing

In this preprocessing step we divided the vertices into groups/partitions. First, find Dominating set of vertices for the given large data graph, then partition the graph into non-overlapped sub-graphs by using Dominating set.

Finding Dominating Set

Finding dominating set is an NP-Complete problem. Using vertex and its degree, we start the dominating set algorithm with vertex which has the highest degree.

Find each vertex and its degree: Only one Map method is enough to get each vertex and its degree the algorithm is shown in Algorithm1. This algorithm generates each vertex and its degree in a file and store that file in distributed file system.

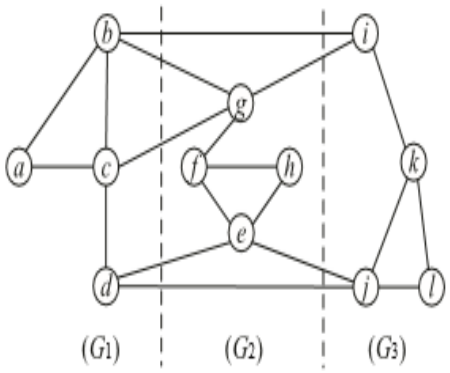

Fig.2. Example graph database

Algorithm 1: Find each Vertex's Degree

```
class Mapper {
map(offset int, Value :vertex and list of neighbors)
{
parse value;
key=vertex;
Adjacency list=concatenation of all values in value;
count=sum of a number of neighbors in the neighbors
        list;
value=adjacencylist+" "+count;
emit (key,value)
}
}
```

Sort: To sort the vertices based on degree, we used Terasort program available in Hadoop and sort the above file based on the count in descending order. So the highest degree vertex is first in the list. We find domination set of vertices by starting from that vertex.

According to a dominating set definition we get dominating vertices, the algorithm is shown in Algorithm 2. This algorithm is execute on a single machine.

Algorithm 2 : Algorithm to find Dominating set

```
u=highest degree vertex;
A : array of bits of size |V(G)| and initialize each bit
        with 0;
for each u in G, where A[u] = 0,
  do
      Add u to DS and mark u and all u's
            Neighbors as 1 in A;
return DS;
```

The number of partitions is decided based on input graph size. Use the DS (Dominating Set) to compute vertex partitions. Divide DS into p subsets according to min cut algorithm. Select the sub-graph that has at least (deg(v)/p) neighbors of v currently, and add v to that sub-graph. If there is more than one such sub-graph, add v to the sub-graph with the smallest size so far. The result of p subsets is loaded into a file "psubsets".

B. Map-Reduce based Graph Partition with Dominating Set

Workload distribution among mappers-reducers: In DS base approach, use the dominating vertices in each subset as seeds to grow each of the p subgraphs by attracting their neighbors. Again, we read G sequentially from disk. For each vertex v (together with adjG(v)) read, let degP(v) be the current total number of neighbors of v in all the subgraphs in the current P, which can be easily obtained by scanning adjG(v).

We choose the subgraph that has at least (degP(v)/p) neighbors of v currently, and add v to that subgraph. If there are more than one such subgraph, we add v to the subgraph with the smallest size so far. While in earlier algorithms this provision is not there. So definitely DS based approach is far better with respect to workload balancing among mappers and reducer.

Vertex subsets psubsets is used to compute graph partitions. Based on psubsets divide the graph into p partitions which are non overlapped sub-graphs. The overlapped edges are called as extended edges. These extended edges have to be placed in a file extended_file.

Fig. 1 shows the example graph and fig. 2 shows its extended edges and three partitions of the given graph.

One Map method is used to create Graph Partitions using Dominating Set and to create an extended edge file.

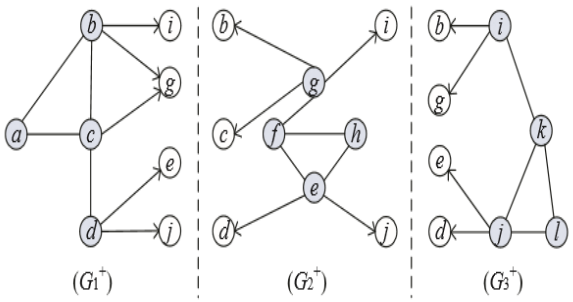

Fig.3. An example for Extended edges

Algorithm 3: Generation of Graph Partitions and extended edge file

```
Class Mapper {
setup()
{
load psubsets  file into distributed cache.
 }
method map( offset , value: vertex  v_neighbours) {
if (v ϵ p1 or p2 or p3 ..){
prepare key
for( each neighbor n in v_neighbours)
        if ( n is not in this group ){
emit ( v, n) to a file "extended_file"
}
else
emit(key, value) to  file "key_file"
}
}
```

The result of the mapper is graph partition files and one extended_file which contains extended edges.

The extended edges of the example graph given in Fig.1 are shown in Table 1.

The MultipleOutputs class is used to write output data to multiple outputs files. Here, we write partitions to different files provided by the user for example partition 1 is written in file_1 and the second partition is written in file_2. The file names are different and are used to identify the partition.

These extended edges are placed in a separate output file i.e Extended_File. Algorithm 3 shows the procedure to generate non-overlapped graph partitions and extended graph.

C. Multiple Sub-graph Enumeration using Map-Reduce based Graph Partitions using Dominating set

Using only one Map-Reduce round we can enumerate multiple sub-graphs. The input to the Map-Reduce round is the partitioned files and extended edge file is loaded into Distributed cache. Extended_File is available to all mappers. Each mapper gets one partition file. According to the number of partition files, we need the same number of mappers. Non overlapped sub-graphs are loaded into mapper, so the mapper work is to enumerate multiple sub-graphs. If any edge is not available it can check in the extended file. Using the edges in the extended file, we can enumerate sub-graphs and the result of mappers is going to the reducers.

Triangle Enumeration : Based on starting node, each triangle can be enumerated six times. To eliminate this extra computation we used node ordering.

Node ordering (<) assumes for an edge E(a,b) only if a < b then each triangle is discovered exactly once. To enumerate triangles we use this conjunctive query for three edges: E(A,B) & E(B,C) & E(A,C) and A < B < C.

Square Enumeration: The following three Conjunctive Queries are used to find each square exactly once:

E(A,B) & E(B,C) & E(C,D) & E(A,D) & A < B & B < C & C <D

E(A,B) & E(C,B) & E(C,D) & E(A,D) & A<C & C<B & B <D

E(A,B) & E(B,C) & D,C) & E(A,D) & A<B & B < D & D < C.

The algorithm to enumerate multiple sub-graphs is shown in Algorithm 4.

Algorithm 4 Enumeration of multiple sub-graphs

```
Mapper
setup()
{
load  Extended_File into distributed cache
}
map(offset, value:line)
{
read value and parse it
add vertex and edges to  local_sub_graph
}
cleanup()
{
mine triangle, square, star. using local_sub-graph
if (E(A,B) and E(B,C) and E(A,C) and A<B and B<C )
emit (A,B,C ) to triangle file
if( E(A,B) & E(B,C) & E(C,D) &E(A,D)&A<B&B <C & C <D)
emit (A,B,C,D ) to rectangle file
if ( E(A,B) & E(C,B) & E(C,D) & E(A,D) & A <C & C<B & B < D)
emit (A,B,C,D) to rectangle file
if( E(A,B) & E(B,C) & E(D,C) & E(A,D) & A <B & B<D & D < C)
emit(A,B,C,D) to rectangle_file
        for each vertex v
        {
          value =  add neighbours of v using space
          emit(v, value) to star_file
        }
}
}
```

In agorithm 4 mapper is generating multiple outputs by using MultipleOutputs class.

The MultipleOutputs class is used to write output data to multiple outputs files.specified by the user. If the job has no reducers, practitioners and combiners, each mapper outputs one output file.

At some point, we should run some post processing to collect the outputs into a single large file.

This proposed approach is efficient compared to TTP[12] and Multi-Way join[9]. Instead of enumerating one sub-graph at a time we enumerate Multiple sub graphs with same communication cost and computation cost.

For complex sub-graphs: We have to do pre-processing by dividing given complex sub-graphs into simple sub-graphs and we also have to get the instances of these simple graphs and then apply join technique. One more Map-Reduce Round is required to join the simple sub-graphs to get complex sub-graph enumeration.

We are reducing the search space by using these simple sub-graph enumeration to enumerate complex sub-graphs. This approach is reducing both computation cost and communication cost of complex sub-graphs enumeration.

V. Experiments

This section presents an experimental results that demonstrate the performance of our approach on real datasets. In the following experiments, we aim to determine the efficiency of dominating set based graph partition and enumeration of multiple sub-graphs using Map-Reduce. In particular, we compare the enumeration time and the number of bytes transferred. At first, it describes the used datasets and implementation details. Then, it presents a discussion of the obtained results.

Table 1 Extended edges

Edge	
b	I
b	G
d	E
d	J
g	I
e	J
c	G

A. Experimental setup

1. Datasets

The datasets used in our experimental study are described in Table 2 Real world graph datasets, which are taken from an online source that contains graphs extracted from the PubChem website. PubChem contains one million chemical structures. Each graph has 23.98 vertices, 25.76 edges, 3.5 distinct vertex labels, 2.0 distinct edge labels on average, and the total number of distinct vertex labels and distinct edge labels is 81 and 3, respectively. The size of PubChem dataset is 434 MB.

B. Implementation platform

We implemented this work in Java and using Hadoop (version 1.2.1) an open source version of Map-Reduce. The database files are stored in the Hadoop Distributed File System(HDFS) an open source implementation of GFS . All the experiments of our approach were carried out using a local cluster with 8 nodes. The real datasets are used to execute the experiments. The processing nodes used in our tests are equipped with a Hardware:1 + 8 Node Cluster.

Table 2 Real Data Sets

Dataset	Name	Number of vertices	Number of Edges
Live-journal	LJ	3,997,962	3,4681,189
Youtube	YT	3,223,589	12,223,774
As-skitter	AS	1,696,415	11,095,298
Enron	EN	36,692	183,831

Front-end: HP Proliant DL380P Gen8, 2 x Intel xeon CPU E5-2640 (2.5 GHz / 6-core/15MB/95w) processor, 64 GB RAM, 333 X 600GB HDD machine.

Data nodes : Run on 1 X Intel® xeon® E5-2640 (2.5 GHz / 6-core/15MB/95w) processor, 16GB RAM, 2 X 300GB HDD machines.

C. Experimental Results

1. Running time for Different Real Datasets

The running times of all algorithms were recorded in the minutes. This experiment is executed on 8 node cluster and uses the real datasets specified in table 2. We recorded execution time for each dataset separately. This experiment clearly shows that our approach is taking very less time compared to Suri paper[5] and TwinTwigJoin [10] algorithms. The running time is shown in table 3 for different real data sets specified in table 2. In suri[5] paper the overlapped edges are more so intermediate data is more. TwinTwig join uses more number of Map-Reduce rounds so it is taking more time compared to our proposed approach DS based graph partition.

Table 3 Execution Time

Dataset	Suri Paper	Twin-TwigJoin	DS-Based-GP
As-skitter	365	257	100
Livejournal	1657	1040	659
Youtube	1523	934	549
Enron	95	45	30

The running times are recorded for real datasets and is shown in fig 3 to fig 6.

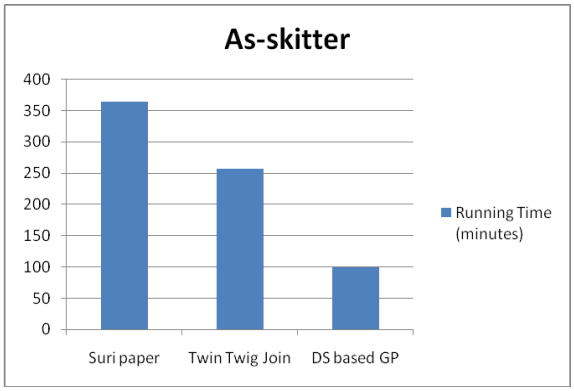

Fig.3. As-skitter Data Set Running Time

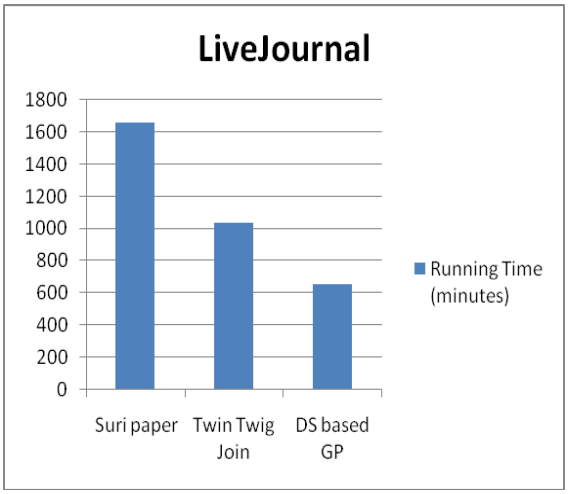

Fig.4. LiveJournal Dataset Running Time

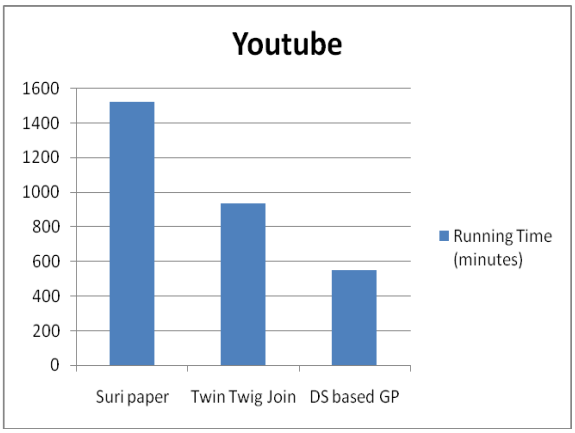

Fig.5. Youtube Dataset Running Time

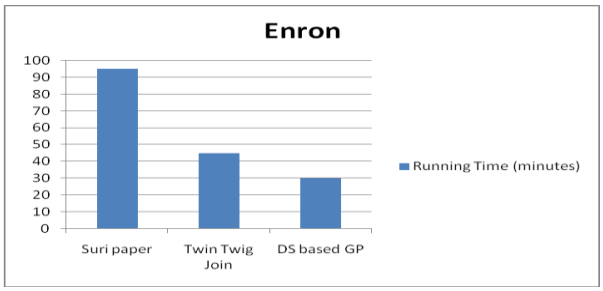

Fig.6. Enron Dataset Running Time

2. Communication cost for Real datasets

In this experiment, we recorded the number of bytes communicated over the network. Our DS-Based-GP approach is distributing less number of bytes because it partition the graph into extended sub-graphs not as overlapped sub-graphs. Table 4 shows the number of bytes transferred on the network for different algorithms. This proposed approach is dividing the graph into non-overlapped partitions so the number of bytes transferred on network is very less compared to other algorithms.

Table 4 Communication Cost

Dataset	Suri Paper	TwinTwigJoin	DS-Based-GP
As-skitter	111,267,456	101,345,789	50,234,123
Livejournal	398,123,345	250,125,345	120,345,212
YouTube	312,235,412	212,567,870	115,234,569
Enron	11,241,256	10,134,234	5,231,120

3. *Number of Nodes versus Running Time*

We conducted this experiment for different number of nodes in the cluster using dominating set based graph partition algorithm for LiveJournal dataset. As the number of nodes is increasing the time required to run the program was decreasing. If there are less number of nodes, then it requires more time. Fig. 7 shows the time (in minutes) versus number of nodes.

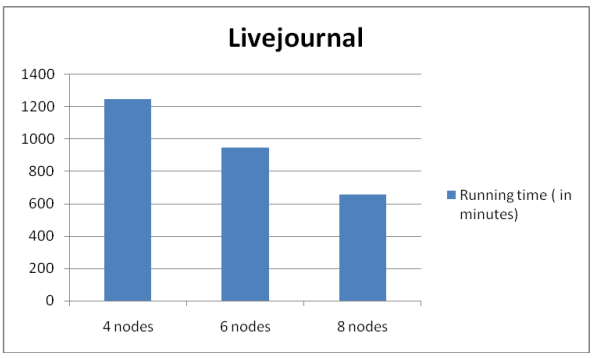

Fig.7. Number of Nodes versus Running Time

4. *Multiple sub-graph versus Running Time*

We conducted this experiment on 8 node cluster using Livejournal dataset. Here we enumerate triangles, then triangles and squares and then all three sub-graphs. We recorded the running time in minutes and is shown in Fig. 8. It shows that the time taking for triangle enumeration and triangle and square enumeration and all three enumerations are close to each other. If we execute separately it will take more time. This proposed approach enumerates multiple sub-graphs simultaneously with the same communication and computation cost.

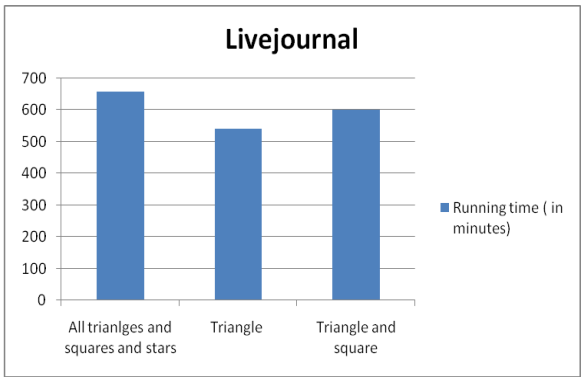

Fig.8. Multiple Sub-graph Enumeration vs Running Time

VI. Conclusions

This paper shows the dominating set based graph partition that is dividing graph into non-overlapped sub-graphs. This non-overlapped graph partition reduces the communication cost. In this paper we developed efficient algorithm to enumerate multiple sub-graphs simultaneously. This proposed approach enumerates multiple sub-graphs with same computational and communication cost. This Dominating Set Based Partition is efficient compared to earlier methods. We have some limitations. Dominating set based graph partitioning is effective when graphs exhibit high locality, i.e., vertices are naturally clustered according to the sequential order by which the graph is stored. For example in, a road network proximate vertices are assigned consecutive vertex IDs and they are stored sequentially in nearby positions in the adjacency-list graph representation. In social network graphs, local communities may also be stored together. So while you are going for adjacency-list graph re-presentation , DS based approach would be the best. When you want that load balancing should be the primary need for you, DS based approach would be the best choice. In future study we are going to develop graph mining algorithms over large graph using this dominating set based graph partition and spark which is an in memory cluster computing.

References

[1] Alon, Noga, Raphael Yuster, and Uri Zwick. "Finding and counting given length cycles." Algorithmica 17.3 (1997): 209-223. DOI: 10.1007/BF02523189

[2] Kowaluk, Mirosław, Andrzej Lingas, and Eva-Marta Lundell. "Counting and detecting small subgraphs via equations and matrix multiplication." Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms. SIAM, 2011.

[3] Tsourakakis, Charalampos E., et al. "Doulion: counting triangles in massive graphs with a coin." Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009,doi:10.1145/1557019.1557111

[4] Schank, Thomas. "Algorithmic aspects of triangle-based network analysis." Phd in computer science, University Karlsruhe 3 (2007).

[5] Suri, Siddharth, and Sergei Vassilvitskii. "Counting triangles and the curse of the last reducer." Proceedings of the 20th international conference on World wide web. ACM, 2011.DOI: 10.1145/1963405.1963491

[6] Kolda, Tamara G., et al. "Counting triangles in massive graphs with Map-Reduce." SIAM Journal on Scientific Computing 36.5 (2014): S48-S77. DOI: 10.1137/13090729X

[7] Pagh, Rasmus, and Charalampos E. Tsourakakis. "Colorful triangle counting and a Map-Reduce implementation." Information Processing Letters 112.7 (2012): 277-281.

[8] Afrati, Foto N., Dimitris Fotakis, and Jeffrey D. Ullman. "Enumerating subgraph instances using map-reduce." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013. DOI: 10.1109/ICDE.2013.6544814

[9] Afrati, Foto N., and Jeffrey D. Ullman. "Optimizing multiway joins in a map-reduce environment." IEEE

Transactions on Knowledge and Data Engineering 23.9 (2011): 1282-1298.DOI: 10.1145/1739041.1739056
[10] Lai, Longbin, et al. "Scalable subgraph enumeration in Map-Reduce." Proceedings of the VLDB Endowment 8.10 (2015): 974-985. DOI: 10.14778/2794367.2794368
[11] Chu, Shumo, and James Cheng. "Triangle listing in massive networks and its applications." Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011. DOI: 10.1145/2020408.2020513
[12] Park, Ha-Myung, and Chin-Wan Chung. "An efficient Map-Reduce algorithm for counting triangles in a very large graph." Proceedings of the 22nd ACM international conference on Information & Knowledge Management. ACM, 2013. DOI: 10.1145/2505515.2505563
[13] Hu, Xiaocheng, Yufei Tao, and Chin-Wan Chung. "Massive graph triangulation." Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM, 2013.DOI:10.1145/2463676.2463704
[14] Sun, Zhao, et al. "Efficient subgraph matching on billion node graphs." Proceedings of the VLDB Endowment 5.9 (2012): 788-799. DOI: 10.14778/2311906.2311907
[15] Viger, Fabien, and Matthieu Latapy. "Efficient and simple generation of random simple connected graphs with prescribed degree sequence." International Computing and Combinatorics Conference. Springer Berlin Heidelberg, 2005. DOI: 10.1007/11533719_45
[16] Zhang, Xiaofei, Lei Chen, and Min Wang. "Efficient multi-way theta-join processing using Map-Reduce." Proceedings of the VLDB Endowment 5.11 (2012): 1184-1195.DOI: 10.14778/2350229.2350238
[17] Zhao, Peixiang, and Jiawei Han. "On graph query optimization in large networks." Proceedings of the VLDB Endowment 3.1-2 (2010): 340-351. DOI: 10.14778/1920841.1920887
[18] Zhao, Zhao, et al. "Subgraph enumeration in large social contact networks using parallel color coding and streaming." 2010 39th International Conference on Parallel Processing. IEEE, 2010.DOI: 10.1109/ICPP.2010.67
[19] Wang, Chaokun, et al. "MapDupReducer: detecting near duplicates over massive datasets." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010.DOI: 10.1145/1807167.1807296
[20] Chiba, Norishige, and Takao Nishizeki. "Arboricity and subgraph listing algorithms." SIAM Journal on Computing 14.1 (1985): 210-223.
[21] Lin, Jimmy, and Chris Dyer. "Data-intensive text processing with Map-Reduce." Synthesis Lectures on Human Language Technologies3.1(2010):1-170
[22] Hadoop Distributed File System hdfs : http://hadoop.apache.org/hdfs
[23] Hadoop. http://hadoop.apache.org
[24] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.DOI: 10.1145/1165389.945450
[25] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI, pages 137–150, 2004. doi: 10.1016/j.procs.2015.07.392
[26] J. Cohen. Graph twiddling in a mapreduce world.Computing in Science and Engineering, 11(4):29–41, 2009.
[27] Vernica, Rares, Michael J. Carey, and Chen Li. "Efficient parallel set-similarity joins using MapReduce." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010, 10.1145/1807167.1807222
[28] Dilbag Singh,Jaswinder Singh,Amit Chhabra "Failures in Cloud Computing Data Centers in 3-tier Cloud Architecture", IJIEEB, vol.4, no.3, pp.1-8, 2012. DOI: 10.5815/ijieeb.2012.03.01
[29] Jitendra Singh,"Study of Response Time in Cloud Computing", IJIEEB, vol.6, no.5, pp.36-43, 2014. DOI: 10.5815/ijieeb.2014.05.06
[30] Karim Zarour, Nacereddine Zarour,"Data Center Strategy to Increase Medical Information Sharing in Hospital Information Systems", IJIEEB, vol.5, no.1, pp.33-39, 2013. DOI: 10.5815/ijieeb.2013.01.04

Author's profiles

Fathimabi shaik is pursuing PhD in the department of Computer Science and Engineering, National Institute of Technology, Warangal, India. She received her Master of Technology from University College of Engineering, Kakinada, JNTUK, Andhra Pradesh, India. Her areas of interest include Data Mining, Graph Data Mining, Big Data Analytics, Business Intelligence and Distributed Data Mining.

Dr. RBV Subramanyam is an Associate Professor in the department of Computer Science and Engineering, National Institute of Technology, Warangal. He received his Master of Technology and Doctor of Philosophy from Indian Institute of Technology, Kharagpur. His areas of Interest include data mining, Distributed Data Mining, Graph Databases, Fuzzy data mining, Big data analytics, pattern recognition, high performance computing, Soft Computing, Game Theory, Outlier Analysis.

Prof. DVLN Somayajulu received his PhD from Department of Computer Science and Engineering, Indian Institute of Technology, Delhi in 2002. Joined at NIT (REC) Warangal in September 1988 after completing Mtech from Indian Institute of Technology, Kharagpur in 1987. He is currently working as Professor in Department of Computer Science and Engineering, National Institute of Technology, Warangal. His areas of interest include Business Intelligence, Big Data Analytics, Database Security, Distributed Databases, Data Warehousing and Data Mining and Advanced Databases.

How to cite this paper: Fathimabi shaik, RBV Subramanyam, DVLN Somayajulu,"Map-Reduce based Multiple Sub-Graph Enumeration Using Dominating-Set Graph Partition", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.9, No.2, pp.36-44, 2017. DOI: 10.5815/ijieeb.2017.02.05

