Map-Reduce Local Community Mining

CMPUT 690 Project Report

Mohammad Motallebi, motalleb@ualberta.ca
Saeed Sarabchi, sarabchi@ualberta.ca

April 25, 2017

Abstract

In this work, we modified the L-Metric local community identification algorithm to parallelize it using the Map-Reduce framework. To do so, we modified the algorithm and introduced *Map* and *Reduce* tasks to be able to run it in parallel. We also ran different experiments on the proposed algorithm to show its applicability.

1 Introduction

Social Network Analysis (SNA), a subbranch of Data Mining, is a field where researchers work on a group of entities that are having kind of a relationship between each other that resembles a society. These entities usually are human beings. These entities and their corresponding relationships are usually modeled using a graph. A graph is defined as $G(V, E)$, where V is the set of vertices and E is the set of edges, each connecting two vertices to each other. Notice that these edges can be either weighted or unweighted, directed or undirected. Each vertex corresponds to a person and each edge corresponds to a relation those two people have with each other. As an example, a friendship between two persons can lead to an edge in a graph depicting friendships in a society. As another example, given a graph with vertices corresponding to customers of a telecommunication company, edges can show the minutes those two individuals have talked to each other during the last month.

Such graphs have been studied for many decades. Many scientists have been interested in such graphs since besides being a way to represent the relationship, graphs can reveal many hidden facts about the group of people in it, or the society. Among those important facts, is the communities that are involved within the society. A community in a graph, similar to a community in a real-life society, is defined to be a group of vertices where the strength of the relationship between them is more than other vertices in the graph [6].

A very preliminary but crucial step in community mining is identifying the communities in the graph. This task, which is called community identification, is the effort of identifying communities and labeling vertices based on the community they belong to. There have been many works related to identifying communities in a graph [9][10][4]; however, there is a big issue regarding these algorithms. There are many real-life graphs which are very big, as an example, there are over 1.86 billion active users as of April 2017[1] in Facebook[2]. All aforementioned algorithms need a huge amount of memory to be able to process such a gigantic graph, and this is problematic for them. To solve this problem, it has been about a decade since researchers started working on methods called local community identification where they try to solve this problem by proposing algorithms that just run on part of the graph. Nonetheless, these algorithms that are sequential, suffer from the long running-time problem.

Map-Reduce[5] is a programming framework for processing large amounts of data in a parallel paradigm and is highly scalable on large clusters of commodity machines. In this framework, the input data is chunked and replicated among all the cluster machines. From the big picture, Map-Reduce consists of two basic operations: Map, which passes over the input data and outputs key-value pairs, and Reduce, which aggregates the key-values with the same key and outputs the result. The idea behind this framework is that if we can define a program as a set of Maps and reduces, then we can gain high performance since the execution of these operations can be distributed over large clusters of machines.

In the next sections, first, some related work will be discussed in section 2. In section 3, the proposed method will be presented, then followed by some experiments and their corresponding results in section 4. Section 5 and 6 will end the paper by concluding and having some directives for future work.

2 Related Work

As discussed in the previous section, there have been some efforts by researchers to develop local community identification methods. These methods all intersect in their methodology. They all start with a root vertex and then try to find the underlying community for that vertex by iteratively adding another vertex to the community. The only difference, which creates different communities for different methods is the criterion they use to add the next vertex to the community. Clauset [3] developed a method called R measure as the criterion in selecting the next vertex to be added to the community. This measure is calculated in this way:

$$R = \frac{B_{in\_edge}}{B_{out\_edge} + B_{in\_edge}} \tag{1}$$

In this equation, B_{in_edge} corresponds to the number of edges between boundary vertices (i.e. vertices having edges to vertices outside the community) and vertices inside the community while B_{out_edge} is defined as the number of edges between boundary vertices and vertices outside the community.

[1]https://zephoria.com/top-15-valuable-facebook-statistics/
[2]https://www.facebook.com

Another measure that has been suggested by Luo et al [8] is M. The value M, with the current community being D, is equal to the ratio of the number of edges within the community over the number of edges between the community and outside world:

$$M = \frac{ind(D)}{outd(D)} \tag{2}$$

Chen et al [2] proposed another metric called L. In this metric, in contrast with R and M, the numbers are normalized. This metric is defined as follows:

$$L = \frac{L_{in}}{L_{out}} \tag{3}$$

where L_{in} and L_{out} are defined as follows:

$$L_{in} = \frac{\sum_{i \in D} IK_i}{|D|} \tag{4}$$

$$L_{out} = \frac{\sum_{j \in B} EK_j}{|B|} \tag{5}$$

which IK_i and EK_j correspond to the number of neighbors in the community and number of neighbors outside the community for vertices i and j, respectively.

2.1 Map-Reduced M Metric

Wang et al [11] proposed a method to map-reduce the M-metric algorithm introduced in [8]. In their work, they used Hadoop[1] to map-reduce the original algorithm. While their work is a good step forward and they showed promising results, as they mention on it themselves, M-metric algorithm has the drawback of neglecting outlier vertices. This shortcoming is covered in the L-metric algorithm [2].

It is also noteworthy that their work is based on Hadoop which is shown to be inefficient for iterative tasks in comparison with other Map-Reduce frameworks[7]. This is because of the fact that for each iteration, Hadoop scans the whole input file from the disk and after each *reduce* task, it writes the results to the disk. On the other hand, some other frameworks such as Spark make the portion of the data which is read-bound, available in the distributed memory of the cluster using a specific data type called *Resilient Distributed Dataset*[14], which leads to faster execution of the iterations. Hence, Spark is used in this work as the Map-reduce framework.

3 Methodology

The approach taken in this work is similar to that of [11] except it is modified to comply with differences L-metric algorithm has with M-metric.

3.1 Sequential L-metric

The sequential L-metric algorithm is shown in Algorithm 1. Original L-metric algorithm introduced in [2] computes L_{in} and L_{out} for each vertex as shown in the Equations 4 and 5, respectively. This algorithm, based on calculated values of L_{in} and L_{out}, divides border vertices to 4 separate categories:

1. $L'_{in} > L_{in}$ and $L'_{out} < L_{out}$
2. $L'_{in} < L_{in}$ and $L'_{out} < L_{out}$
3. $L'_{in} > L_{in}$ and $L'_{out} > L_{out}$
4. $L'_{in} < L_{in}$ and $L'_{out} > L_{out}$

where in each category,L_{in} and L_{out} correspond to current values of the community and L'_{in} and L'_{out} correspond to the values when the candid vertex is added to the community.

Vertices in the fourth category obviously have less L' value than current L value, and hence cannot be added to the community. The authors of the paper also mention vertices belonging to the second category as outliers and thus need not to be added to the community. They also evaluate vertices in the third case as the potential hub vertices that need to be added for now but checked later. This introduces the second stage of the algorithm that the L_{in} and L_{out} are re-evaluated for all vertices belonging to the community to make sure they are not a hub vertex.

Algorithm 1 Original L-metric Algorithm

```
1: Input: A social network $G$ and a start node $n_0$.
2: Output: A local community with its quality score $L$.
3: 1. Discovery phase:
4: Add $n_0$ to $D$ and $B$, and all $n_0$'s neighbors to $S$.
5: do
6:     for each $n_i \in S$ do
7:         compute $L'_i$
8:     end for
9:     Find $n_i$ with the maximum $L'_i$, breaking ties randomly
10:    Add $n_i$ to $D$ if it belongs to the first or third case
11:    Otherwise remove $n_i$ from $S$.
12:    Update $B$, $S$, $C$, and $L$.
13: while $L' > L$
14: 2. Examination phase:
15: for each $n_i \in D$ do
16:     Compute $L'_i$, keep $n_i$ only when it is the first case.
17: end for
18: 3. If $n_0 \in D$, return $D$, otherwise there is no local community for $n_0$.
```

3.2 Map-reduced L-metric

For implementing the parallel version of L-Metric algorithm, the basic ideas in [11] are used. The general pseudo code is shown in Algorithm 2 which can be implemented in a Map-Reduce framework such as Spark. In the parallel

Algorithm 2 Parallel version of L-metric Algorithm

```
Input: A social network $G$ and a start node $n_0$.
Output: A local community with its quality score $L$.

1. Add $n_0$ to $D$ and $B$
2. Compute $L_i\prime$ for each of the Shell nodes in parallel
3. Find $n_i$ with the maximum $L_i\prime$, breaking ties for the lower nodes
4. Add $n_i$ to $D$ if it belongs to the first or third case
5. If $n_i$ is added, Update $B$, $C$, and $L$ and Goto 2
6. Retrieve the graph for $C$.
7. Do the Examination Phase sequentially
```

version, whenever there is a need for the graph to be scanned, it is in parallel except the examination phsae which is executed sequentially due to the fact that usually the graph associated with a community is much smaller than the whole graph. Hence the overhead for running this phase in parallel is higher than executing it sequentially. As it is shown in the algorithm, the steps used for the parallel version are nearly the same as the sequential one, except the fact that in finding the maximum L, breaking the ties is not random, in order to make the algorithm deterministic in each step. Computing the L-Measure is the most time consuming step of the alorithm which can be performed in parallel. In order to compute L, the following expression can be used[2]:

$$L'_i = \frac{\frac{Ind+2*Ind_i}{|D|+1}}{\frac{Outd-Ind_i+Outd_i}{|B'|}} \tag{6}$$

where Ind and Outd are the number of inward edges and outward edges in the commumty before merging with node i. So the inward and outward degrees of the communities should be calculated before the discovery phase. In the first iteration, since the community is composed of only one node, there would be no inward edge, but outward edges should be calculated initially for each node. The Map-Reduce version of computing the initial outDegrees for the starting nodes are shown in Algorithms 3 and 4. In algorithm 3, the map phase for this task is illustrated. It simply emits value '1' for each node of an edge and in the reduce phase, it summarizes each of the '1's to compute the outDegree of that node.

For computing the L-measure in parallel, first each node that contributes to the inDegree of its community (Shell nodes) should be mapped with flag:1, while those nodes that contribute to the outDegree of the community are mapped with flag:-1. This mapping phase is shown in Algorithm 5.

Algorithm 3 Initial Count Out-Degree Mapper

```
Input: List of Edges (i,j)
Output: List of (Node,1)

for each edge(i,j) do
    Write(i,1)
    Write(j,1)
end for
```

Algorithm 4 Initial Count Out-Degree Reducer

```
Input: (Node,List of ones)
Output: (Node,OutDegree)

Set OutDegree = 0
for each (Node, one) do
    OutDegree = OutDegree+1
end for
Write (NodeID,OutDegree)
```

Algorithm 5 Compute L Mapper

```
Input: Edges (i,j), Community D
Output: (Key, Value:Flag)

for each Edge(i,j) do
    if $i \in$D and $j \notin$D then
        Write (j, i:1)
    end if
    if $i \notin$D and $j \in$D then
        Write (i, j:1)
    end if
    if $i \notin$D and $j \notin$D then
        Write (i, j:-1)
        Write (j, i:-1)
    end if
end for
```

In the reduce phase shown in Algorithm 6, the L-measure for each of the shell nodes are calculated. The inDegree gain for each shell node is calcualted by accumulating the positive flags. similarly, the outDegree gain for each shell node can be determined by summarizing the negative flags.

Algorithm 6 Compute L Reducer

```
Input: (NodeId, List of (nodeID, Flag) ),
    Community D, D.Indegree, D.outDegree and D.Borders
Output: (NodeID, L)
Set localIndegree=0 , localOutDegree=0
for each (nodeID,Flag) do
    if $Flag$ =1 then
        localIndegree = localInDegree + 1
    else
        localOutDegree = localOutDegree + 1
    end if
end for
if $localIndegree$ >0 then
    $L'_{in} = \frac{D.InDegree+2*localIndegree}{|D|}$
    $L'_{ex} = \frac{D.OutDegree-localInDegree+localOutDegree}{|NewBorder|}$
    $L' = \frac{L'_{in}}{L'_{ex}}$
    Write (NodeID, L')
end if
```

4 Results

4.1 Sequential Implementation

In the first step, since the code for original L-metric was not available, the authors implemented the algorithm themselves. To verify the correctness of the code, the Zachary's Karate club dataset [13] was given to it to find communities of some of its vertices. At the same time, the output from running executable file of the algorithm implemented as part of the Meerkat project [3] was used to compare them.

In the aforementioned dataset, there are 34 vertices making up 2 communities. In order to compare the results of two implementations, the same approach taken in original L-metric algorithm paper [2] is used. By giving a node as the start point to the algorithm, it finds the corresponding community. Then, precision and recall are calculated using the ground truth of the graph. The precision is based on the number of vertices in the guessed community that belong to the same community as the starting vertex in the two communities in the ground truth. Recall, on the other hand, is calculated based on the number of selected

[3]https://www.amii.ca/meerkat/

Implementation	Precision	Recall	F1-score
Meerkat	96.87	31.61	47.66
Authors	100	38.97	56.08

Table 1: Precision, Recall, and F1-score comparison for Original L-metric Implementation and The Authors' Implementation on Zachary's karate Club dataset

Datasets	Nodes	Edges	Communities
Karate Club	34	78	5
Facebook egonets1 (Synthetic)	50	88	6
Facebook egonets2 (Synthetic)	100	277	15

Table 2: Datasets used for experimentation on accuracy

vertices which also belong to the original community. Finally, the F1-score is calculated. The results are shown in Table 4.1 As shown in the table, the results are relatively similar and this proves the correctness of the implementation. Also, it is noteworthy that since L-metric algorithm chooses some vertices randomly in some steps of the algorithm, this difference in the results is unavoidable.

4.2 Parallel Implementation

The parallel version of L-Metric algorithm was implemeted using spark framework version 2.0.2 for hadoop 2.7. In total, 5 instances of m1.small size of Cybera-Rac cluster were used: 1 master node and 4 worker-nodes. The operating system for each of the instances was Ubuntu 16.04 and the configuration for each of the instances is as follows:CPU(s): 2 ,CPU MHz: 2294.686, Memory: 2G,HDD: 20G.

The first experimentation for parallel implementation was to make sure that if the performance for spark is better than hadoop in community mining. In order to test this assumption, a spark version of 3MA algorithm[12] was implelemted and tested againt its original Hadoop implemetnation on test graph consisting of 8 nodes and 11 edges. The result in Figure 1 shows that the performance of spark is much higher than hadoop (In the case of this test, nearly 3 times).

In order to measure the accuracy of the parallel version, the algorithm was executed on 3 different datasets and for each dataset, the precision, recall and F-measure was calculated (Figure 2), assuming the sequential algorithm as ground truth. On each dataset, the parallel algortihm received F-measure of one.

The next set of experiments are on the performance of parallel L-Metric. In figure 3, the execution times for parallel algorithm on 4 worker nodes is shown. The reason that the sequential algorithm is so much faster than the map-reduced version is that the overhead for mapping and reducing task was approximately 3 to 4 second each, and for each node, there should be a map-reduce phase.

Next experimentation was to make sure if the execution time for parallel

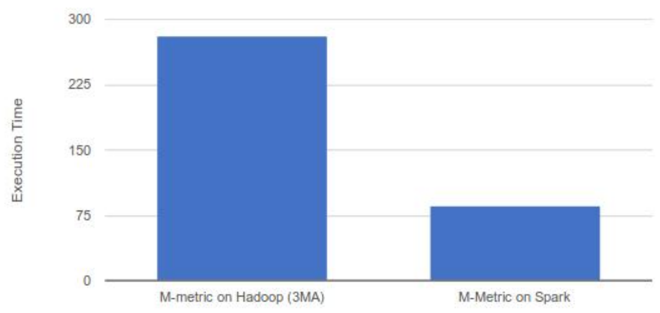

Figure 1: Performance Comparison Between M-Metric using Hadoop and Spark

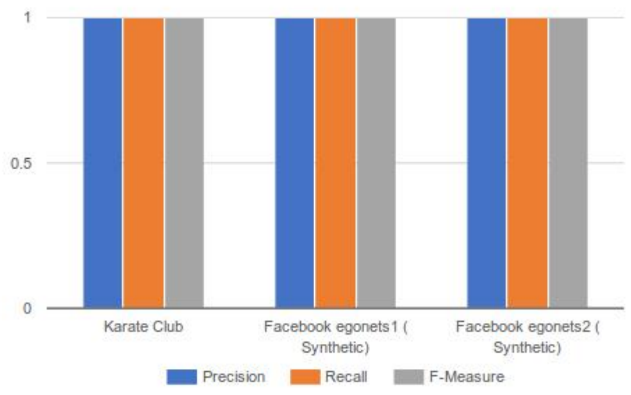

Figure 2: Accuracy of MapReduced L-metric

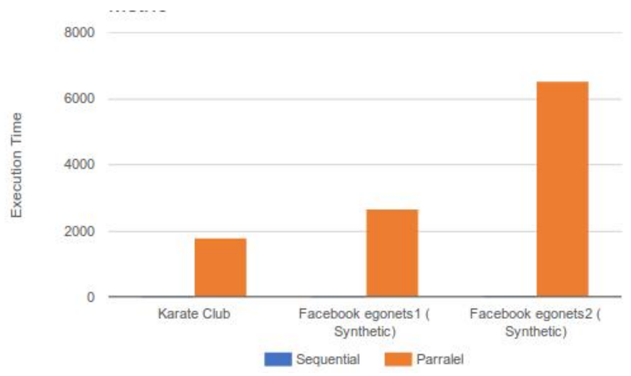

Figure 3: Execution Time for Sequential and MapReduced L-Metric

algorithm will improve if more nodes are added to the spark cluster. This experimentation was done using a large datasets from Orkut social netowrk, of size 1 Gigabyte. Since the number of nodes adn edges are very large in this dataset and the execution time would be too lengthy, the algorithm was changed to find exit when the first community was found. The results are shown in Figure 5 and 4. The experiments were done three-fold averaging on the execution times. In Figure 4 it is shown that the parallel algorithm gets a linear speedup when

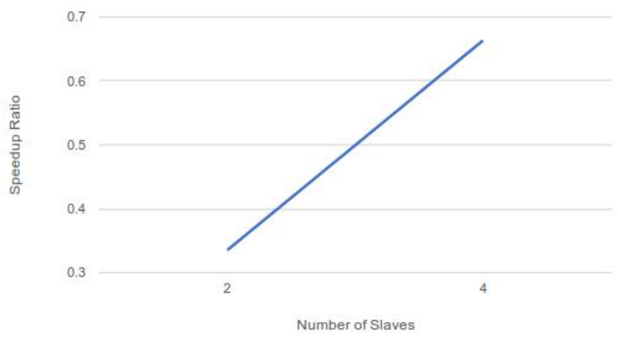

Figure 4: Speed-up of MapReduced L-Metric over Sequential L-Metric

2 nodes are increased to 4 nodes.

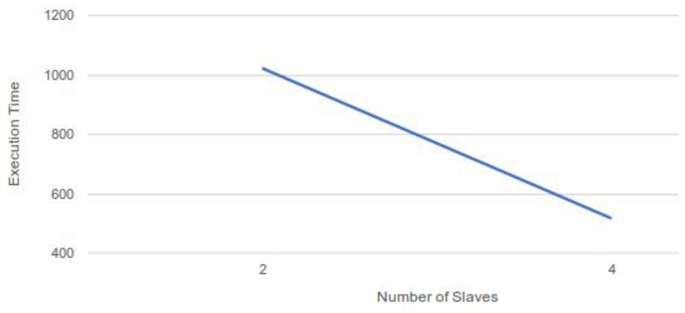

Figure 5: Improvement of MapReduced L-Metric Execution Time by Adding More Slaves on Orkut Dataset(1 Gigabyte)

5 Conclusion

In this work, we implemented a parallel version of L-Metric algorithm for community mining on Spark framework which is a more stable measure in comparison with other measures such as M and R. The experimentaions showed that it can beat Hadoop in terms of execution time, its accuracy is stable and it can get a logical scaleup.

6 Future Work

As a probable further step to continue the work, larger datasets can be used to evaluate the efficiency of the method. Also, more machines can be employed. Taking these steps can eventually result in a publication.

7 Acknowledgement

The authors would like to thank Abhi Panwar for his collaboration in providing authors with the original L-metric algorithm executable file from Meerkat project.

References

[1] Hadoop. `http://hadoop.apache.org/`.

[2] J. Chen, O. R. Zaiane, and R. Goebel. Detecting communities in large networks by iterative local expansion. In *2009 International Conference on Computational Aspects of Social Networks*, pages 105–112, June 2009.

[3] Aaron Clauset. Finding local community structure in networks. *Physical review E*, 72(2):026132, 2005.

[4] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] Steve Gregory. An algorithm to find overlapping community structure in networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 91–102. Springer, 2007.

[7] Haejoon Lee, Minseo Kang, Sun-Bum Youn, Jae-Gil Lee, and YongChul Kwon. An experimental comparison of iterative mapreduce frameworks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2089–2094, New York, NY, USA, 2016. ACM.

[8] Feng Luo, James Z Wang, and Eric Promislow. Exploring local community structures in large networks. *Web Intelligence and Agent Systems: An International Journal*, 6(4):387–400, 2008.

[9] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726. ACM, 2007.

[10] Nathaniel Tashima, Cathleen Crain, Kevin O'Reilly, and Claire Sterk Elifson. The community identification (cid) process: a discovery model. *Qualitative Health Research*, 6(1):23–48, 1996.

[11] Ren Wang, Andong Wang, Talat Iqbal Syed, and Osmar R Zaiane. Scalable local community detection with mapreduce for large networks. *International Journal of Data Mining and Knowledge Management Process*, 7(2):35–49, 2017.

[12] Hongjun Yin, Jing Li, and Yue Niu. Detecting local communities within a large scale social network using mapreduce. *Int. J. Intell. Inf. Technol.*, 10(1):57–76, January 2014.

[13] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.