DATA SCIENCE JOURNAL

Al Aghbari, Z, et al. 2019. GeoSimMR: A MapReduce Algorithm for Detecting Communities based on Distance and Interest in Social Networks. *Data Science Journal*, 18: 13, pp. 1–20. DOI: https://doi.org/10.5334/dsj-2019-013

RESEARCH PAPER

GeoSimMR: A MapReduce Algorithm for Detecting Communities based on Distance and Interest in Social Networks

Zaher Al Aghbari[1], Mohammed Bahutair[2] and Ibrahim Kamel[2]

[1] Department of Computer Science, University of Sharjah, AE

[2] Department of Electrical and Computer Engineering, University of Sharjah, AE

Corresponding author: Zaher Al Aghbari (zaher@sharjah.ac.ae)

Analyzing social networks has received a lot of reviews in the recent literature. Many papers have been proposed to provide new techniques for mining social networks to help further study this huge amount of data. However, to the best of our knowledge, none of them considered the *semantic meaning* of the nodes interests while clustering the network. In this work, we propose a new algorithm, namely GeoSim, for clustering users in any social network site into communities based on the *semantic meaning* of the nodes interests as well as their relationships with each other. Moreover, this paper proposes a parallel version of the GeoSim algorithm that utilizes the MapReduce model to run on multiple machines simultaneously and get faster results. The two versions of the algorithm (centralized and parallel) are examined thoroughly to test their performance. The experiments show that both versions of the GeoSim algorithm achieve high community detection accuracy and scale linearly with the size of the cluster.

Keywords: Social Networks; Communities; Geodesic Location; Interest Similarity; MapReduce

I. Introduction

Social network sites, such as Facebook, have played a great role in our daily lives due to its ability to link users regardless of their geographical location (Tang et al., 2014). With this huge popularity of social network sites, a great need for techniques to process this tremendous amount of data has increased. Many research works have proposed techniques and algorithms to extract information from the data held in the social network sites.

A social network can be represented by a graph, in which users are the nodes in the graph and the relations (or friendship) between users are the edges (Kheirkhahzadeh et al., 2016). A node in a social network could also have a list of attributes that defines the interests of that particular user. As a result of this increasing importance of social networks, several studies have proposed different techniques for mining social networks.

A crucial aspect of such networks is the community detection problem (Sun et al., 2012 and Tobias et al., 2014). Community detection aims to divide the social network into groups. These groups consist of nodes that are highly related to each other. Community detection has a lot of applications such as recommendations, influence analysis and customer segmentation (Qi et al., 2012). Most of the proposed community detection algorithms cluster the network based solely on the linkage behavior between the nodes. However, in the real world, the *relatedness* of users does not depend only on the relation between the users. Another key factor to obtain a more accurate measure of how two nodes are related is by comparing the interests of both nodes. On the other hand, if only the interests of the nodes are considered to measure closeness, the fact that two nodes that are close to each other does not necessarily mean they are related.

One way to get the relatedness of two nodes is by calculating the geodesic distance between the nodes. The geodesic distance between any two given nodes is obtained by the number of hops between them. So, if the two nodes, *A* and *B*, are connected directly to each other, then each one of them is a hop away from the other. This is the closest distance any two nodes can get from each other. On the other hand, if there is a node in the middle, i.e. node *A* connected to a random node and the random node is connected to node *B*, then nodes *A* and *B* are two hops away from each other. When the distance between the two nodes increases, i.e. more hops in the middle, the probability of them knowing each other decreases and therefore their relatedness decreases as well.

While geodesic distance provides a relatively good measure of the relatedness, it is not adequate to fully get a sense of how much a node knows the other node. To further enhance this measure, the interests of each node should also be taken into consideration to give a more accurate estimate of the relatedness between the nodes. For instance, node *A* is connected directly to nodes *B* and *C*. Node *A* has very similar interests to that of *B* but only little to that of *C*. Taking the geodesic distance only to measure the relatedness between nodes *A* and *B* and *A* and *C* will not give a clear result since we have direct links in both cases. If the interests are taken into account, we can then calculate the *interests similarity* between each pair of nodes to get a more refined estimate for the relatedness.

The contribution of this paper is to devise a new algorithm to detect communities in a given social network based on the **geo**desic locations of the nodes and the **sim**ilarities between their interests (GeoSim Algorithm). The nodes in the resulting communities are **strongly connected** and their interests are very **similar in terms of their semantic meaning**. Furthermore, based on the GeoSim algorithm, we propose a parallel version of the algorithm that utilizes Hadoop, a MapReduce framework, since social networks are typically very large and a single-machine-processing is simply insufficient.

The rest of the paper is organized as follows. Section II gives some of the related work on community detection. A brief description about the MapReduce model is given in Section III. In Section IV the GeoSim algorithm is introduced. Section V presents the parallel version of the algorithm, called GeoSimMR. The experiment results are discussed in Section VI. Finally, we conclude the paper in Section VII.

II. Literature Review

Many approaches have been proposed that divide a social network into communities. Rosvall and Bergstrom (Rosvall and Bergstrom 2008) propose an information-theoretic algorithm for detecting communities. Their approach depends on a random walker that traverses the network and records the nodes it visits. Afterward, each visited node is given a Huffman code based on how frequently the walker visited that node. The walk is then described by concatenating the codewords of the visited nodes. Since the walker would probably stay a longer time in a single cluster, one can describe the walk using a fewer number of bits by creating a two-level coding. This is carried out by Huffman coding clusters and the nodes in each cluster separately. Doing this means that codewords can be reused by different nodes in different clusters which will result in fewer bits to describe the random walk.

Blondel et al. (2008) present another technique that creates a hierarchy of communities. Their algorithm comprises of two phases. In the first phase, the algorithm treats each node as a community. Then, given a node and its neighbors, it calculates the *power* of its neighbors to include that node into their respective community. In the second phase, a new graph is formed by treating each community from the first phase as a single node. Edges between communities are transformed into a single edge and its weight is equal to the summation of all of its forming edges. The two phases are again repeated using the newly generated graph from the previous step.

Newman and Girvan (2004) propose an algorithm to find communities in social networks by looking for the natural divisions in a network. They do this by iteratively removing edges from the network to split it into communities. Their later work in Newman (2006) and Clauset et al. (2004) optimizes on modularity, which is a graph measure that is basically the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random. The works in Gregory (2007, 2008) propose an algorithm for detecting overlapping communities by expanding the community detecting algorithm discussed in Newman and Girvan (2004), which was based on hierarchical clustering.

Tantipathananandh et al. (2007) propose algorithms for identifying communities in social networks that change over time. Their algorithms are based on dynamic programming, exhaustive search, maximum matching, and greedy measures.

Ghosh and Lerman (2010) propose an alternative definition, which states that a community is composed of individuals who have more influence on others within the community than on those outside of it. They propose an algorithm based on an influence-based modularity metric and show how to use it to partition the network into communities. A somewhat similar approach was adopted by (Khorasgani et al., 2013), where they regard a community as a set of followers congregating around a potential leader. Their algorithm starts by identifying promising leaders in a given network then iteratively assembles followers to their closest leaders to form communities, and subsequently, finds new leaders until convergence.

Zhang and Yu (2015) introduce a closeness measure called intimacy. They propose a technique, namely Cold stArt community Detector (CAD), which calculate the intimacy matrix among users across aligned attribute augmented heterogeneous networks with information propagation model.

Altunbey and Alatas (2015) present an algorithm for detecting overlapping communities in social networks. The algorithm tries to optimize network modularity using parliamentary optimization algorithm with fitness function and has shown promising results. The limitation of the technique is that only modularity measure was used as the fitness function to find the overlapping community of a network.

Qi et al. (2014) introduce an algorithm that constructs a weighted graph from a dendrogram. Then, the max-flow and min-cut theory are applied on the new obtained weighted graph. It is worth mentioning that detection of communities on social networks can benefit many applications such as event detection (Shi et al. 2017, 2018) and recommendation systems (Capuruco and Capretz 2010, 2012).

III. MapReduce

MapReduce (Dean and Ghemawat 2008) is a programming model developed by Google in 2004. The goal of this model is to provide an implementation for algorithms to be suitable for running in distributed systems. A MapReduce-based algorithm consists of two main steps, namely **map** and **reduce**. In the map step, data are filtered and sorted whereas aggregate operations such as finding minimum value or counting are done in the reduce step. A MapReduce system, such as Hadoop, is a group of computers connected in a master-slave setup. Master machines manage the distribution of data across slaves, resource allocation for processes and the parallel execution of the application. Slave machines hold data, typically in Terabytes, and execute the actual application on the data it has. MapReduce system also provides data redundancy and fault tolerance. It is worth noting that MapReduce system is only beneficial when dealing with huge volumes of data. So, if an application with relatively small data size were to run on a MapReduce system, it would take more time to finish than running it on a typical computer. This is due to the fact that MapReduce systems have communication overhead as well as disk read and write overhead. In other words, MapReduce systems are very suitable for situations where the processing time is much larger than the communication and disk read/write time.

IV. GeoSim Algorithm

In this paper, we propose a new algorithm, namely GeoSim, for clustering a social network into communities. At the beginning, the core algorithm is discussed and then we show our implementation of the algorithm on MapReduce framework (GeoSimMR).

A. The Basic Idea

Social networks can be represented as a graph *G* with a set of nodes *N* and a set of edges *E*. The nodes represent the users in a real-world social network whereas edges can be considered as the friendship relations between the users. GeoSim aims to cluster the given graph into communities based on the relatedness of the nodes. The relatedness between two nodes can be thought of as how similar the two nodes are in terms of their connections and their interests. A group of nodes that are very related to each other can be considered as a single community.

The issue here is how to define a metric that measures the relatedness between two nodes in order to cluster the network. Two users that are connected to each other should not necessarily be in the same community. For instance, assume users *A* and *B* are friends in some social networks. User *A* is interested in sports like hiking and skiing whereas user *B* is interested in medicine. Although users *A* and *B* are friends, they have completely different interests from each other. Similarly, users that have exactly the same interest should not necessarily be grouped into the same communities. This can happen when the users share the same interests but are far apart from each other.

Based on the above, the distance between the nodes and the similarity of their interests both have a great effect on how to cluster the social networks. The key idea behind the GeoSim algorithm is that it incorporates both the nodes interests as well as their distances from each other. In other words, two nodes that are close to each other in terms of distance and have very different interests have a very low probability of being in the same community. On the other hand, if two nodes have very similar interests but are far away from each other would still have a very low chance to be grouped in the same community. As a result, two nodes will be in the same community if and only if they are close to each other as well as have similar interests. GeoSim algorithm achieves this by using a weighted function that gives a score of how close two nodes are to each other.

B. Detailed Description

As mentioned earlier, the GeoSim uses both distance and interest to determine the clusters in a social network graph. To measure the closeness of any two nodes, we called it the GeoSim score, equation (1) is used.

$$GSS(n_1,n_2)=\alpha\frac{Geo(n_1,n_2)}{LSP(G)}+(1-\alpha)(1-Sim(n_1,n_2)) \tag{1}$$

$GSS(n_1, n_2)$ is the the GeoSim score between the two nodes n_1 and n_2 where $n_1, n_2 \in N$. A weighting variable α is between zero and one. $Geo(n_1, n_2)$ is the geodesic distance between the two nodes, $LSP(G)$ is the longest shortest distance in the graph and $Sim(n_1, n_2)$ is the similarity between the two nodes in terms of their interests. The GeoSim score is a number bound between zero and one, the lower the score means the two nodes are very close to each other.

The geodesic distance can be calculated using any shortest path algorithm. In this work, Dijkstra algorithm is used. As shown in equation (1), the distance between two nodes is divided by the longest shortest path in the graph. This division is a normalization factor to ensure that the distance remains between zero and one.

As for the similarity between interests, the WordNet ontology (Miller 1995) is used to obtain the ancestors of a certain word as well as the information content of that word. To measure the similarity between two interests, we use Lin's similarity (Lin 1998) shown in equation (2).

$$LinSim(i_1,i_2)=\frac{2\max_{\gamma\in\Gamma(i_1,i_2)}-\log\left[\widetilde{p_\gamma}\right]}{\left|\log\widetilde{p_{i_1}}+\log\widetilde{p_{i_2}}\right|} \tag{2}$$

Where $LinSim(i_1, i_2)$ is the similarity between interest 1 and interest 2 bound between zero and one, $\Gamma(i_1, i_2)$ is a set of all common ancestors between the two interests and $\log[\widetilde{p_\gamma}]$ is the information content in the word. Equation (2) gives a score that indicates whether the two interests are close or not. When the score is one, the two interests are very close to each one.

In a social network, a single node can have a set of interests, I, I: $i_1, i_2, ..., i_n$. Equation (2) gives a score between two individual interests. To obtain the score between two sets of interests we use equation (3).

$$Sim(I_1,I_2)=\sum_{i_1\in I_1}\sum_{i_2\in I_2}LinSim(i_1,i_2) \tag{3}$$

Where $Sim(I_1, I_2)$ is the similarity between the two sets of interest I_1 and I_2.

The GeoSim algorithm (see **Algorithm 1**) takes as an input all the nodes in the social network graph along with their friends and interests. GeoSim also takes as an input the number of communities, which is the number of clusters the graph should be divided into.

The algorithm can be divided into three stages: Centroids Selection, Distances and Similarities Calculation and Clustering. GeoSim is an iterative algorithm, which means the three stages are executed repeatedly until the membership of communities become stable. We define the *community_variation* as the percentage difference between the membership of communities in two consecutive iterations. As shown in line 4 of the GeoSim algorithm, if the *community_variation* is less than a user give *threshold*, the GeoSim algorithm stops and returns the set of detected communities, *C*.

In the Centroids Selection stage (lines 5–12), a mean node is selected to represent each community. In the first iteration of the algorithm, since no communities are detected yet, *K* number of nodes are selected randomly as mean nodes, where *K* is the number of communities. The selection criteria used in this stage is based on the degree of the node. The node degree is the number of connections a node has. In other words, the number of friends is the degree of the node. From each community, the mean node is the one with the highest degree. The rational for selecting a centroid of a cluster is that a centroid shall be the most central node to all members of the cluster (most accessible to all members of the cluster). Therefore, the node with the most connections to the other members of the cluster is the most central.

The second stage, namely the Distances and Similarities Calculation stage (lines 17–18), is the heart of the algorithm. In this stage, all the required distances and similarities are calculated. The distance and the similarity are calculated between the node and all the available mean nodes. The distance is calculated using Dijkstra's shortest path algorithm. Although in our implementation we used Dijkstra algorithm due to its simplicity, other more efficient algorithms such as A* algorithm and Floyd-Warshall algorithm can be employed in the second stage. The similarities between interests are measured using equation (3).

Algorithm 1: The GeoSim Algorithm.

Input: *V*: set of all nodes in the social network, *E*: set of all edges in the network, *I*: set of all interests for all nodes and *K*: number of communities.

Output: *C*: set of detected communities.

```
1:  procedure GeoSim($V, E, I, K$)
2:      $C = createRandomClusters(V, K)$;
3:      $distances = apsp(V, E)$ // all-pairs shortest paths
4:      while $community\_variation > threshold$ do
5:          for $c \in C$ do
6:              $max\_degree = 0$
7:              for $node \in c$ do
8:                  if $node.degree > max\_degree$ then
9:                      $mean\_nodes[c] = node$
10:                     $max\_degree = node.degree$
11:                 end if
12:             end for
13:         end for
14:         for $v \in V$ do
15:             $min\_gss = \infty$
16:             for $mean\_node \in mean\_nodes$ do
17:                 $distances = distances(mean\_node, v)$
18:                 calculate $gss$ using equation 1
19:                 if $gss < min\_gss$ then
20:                     $c\_id = mean\_node.community$
21:                     $min\_gss = gss$
22:                 end if
23:             end for
24:             $C[c\_id].addNode(v)$
25:         end for
26:     end while
27:     return $C$;
28: end procedure
```

The last stage, the Clustering stage (lines 19–22), uses the results found by the previous stage to perform the actual clustering. The GeoSim scores between every node and all communities (mean nodes) are calculated using equation (1). Each node would have *k* scores, where each score corresponds to one community. To determine a community for that node, the minimum score is found and the node is added to the community of the minimum score. **Algorithm 1** summarizes the steps for the GeoSim algorithm.

C. Time Complexity

The algorithm consists of a main loop that will iterate until a threshold is matched or the number of maximum iterations *P* is reached, whichever happens first. Inside the main loop, there are two other loops. The first loop finds the next set of mean nodes while the other one calculates the *GSS* scores between the mean nodes and every other node. The first loop iterates *N* times where *N* is the number of nodes. The second loop has a total of *Q* (number of mean nodes) times *N* iterations. As a result, the main loop has a time complexity of $O(2NPQ)$. Note that the algorithm needs the distances between the mean nodes and the rest of the nodes, so worst case scenario is that all nodes become mean nodes at one time, though that is very unlikely to happen. For this situation, the all-pairs shortest paths (APSP) should be calculated. Using Dijkstra, the time complexity for finding APSP is $O(N^3)$ The total time complexity for the GeoSim algorithm is $O(2NPQ + N^3)$.

V. GeoSim MapReduce Version (GeoSimMR)

The GeoSimMR algorithm is a parallel version of the original GeoSim that is modified to run on the MapReduce framework. Recall that GeoSim algorithm consists of three stages as shown in **Figure 1**.

In the GeoSimMR, each stage is implemented separately as each stage depends on the result of the previous one and cannot run simultaneously. In other words, each stage has its own implementation of the map and the reduce methods. From **Figure 1**, it is clear that the algorithm is iterative. Furthermore, stage two of the GeoSimMR is iterative as well. This is due to distances calculation using Dijkstra since it requires several passes on the graph to get the shortest paths.

A. Input Format

Since each mapper in the framework processes files in a one-line-basis, a certain format should be accommodated to be used across all input files. In our implementation each line in the input files has the following format:

$$n\_id < tab > f\_ids,\ i\_ids,\ c\_id,\ d,\ s,\ v$$

The *n_id* is the id of the node which is typically a unique number assigned to each node. The *f_ids* is a list of all the friends ids of the node separated by a pipe character '|'. The *i_ids* holds the set of interests of the node separated by a pipe character. The *c_id* indicates to which community the node is associated. The *d* and s store the distance to the mean node of the community and the similarity between the node and the mean node, respectively. The *v* is a flag used for the distance calculation and it can take one of three states: W indicates that the node has not been visited yet, G means that the node is visited and currently calculating its distance and B indicates that the node has been visited and its distance has been calculated.

B. Mean Nodes Selection

In this stage, a mean node from each community is selected. In the map phase, each mapper receives a line that is formatted as discussed earlier. Based on the list of friends from the input line, the mapper counts the number of friends. The mapper then emits the community id of the node as the key and the node itself as the value.

The combiner receives the results from the mapper stage and combines the values of each pair of the same key into a list. The partitioner takes these lists and sends them to the reducers.

In the reduce stage, each reducer receives a community along with a list of nodes in that community. Using the number of friends calculated in the map phase, the reducer selects the mean node for that community. Recall that the mean node for a community is the node with the highest number of friends. Once the reducer finds the mean node, it emits the community id as the key and the node id with its interests as a value.

Figure 2 shows an example of the Mean Nodes Selection stage with five nodes and three communities. In this example, three mappers and three reducers have been used. The five nodes are split across the three mappers. For instance, mapper 1 takes the nodes 1 and 2. Since both nodes are initially at community 1, both are emitted with key equals to 1. At the reducers side, each reducer handles the nodes in each community. For example, reducer 1 receives the nodes of community 1, namely nodes 1 and 2. Since the number of friends of node 1 is larger than node 2, the reducer selects node 1 as the mean of community 1. **Algorithm 2** summarizes the map and reduce phases of the first stage.

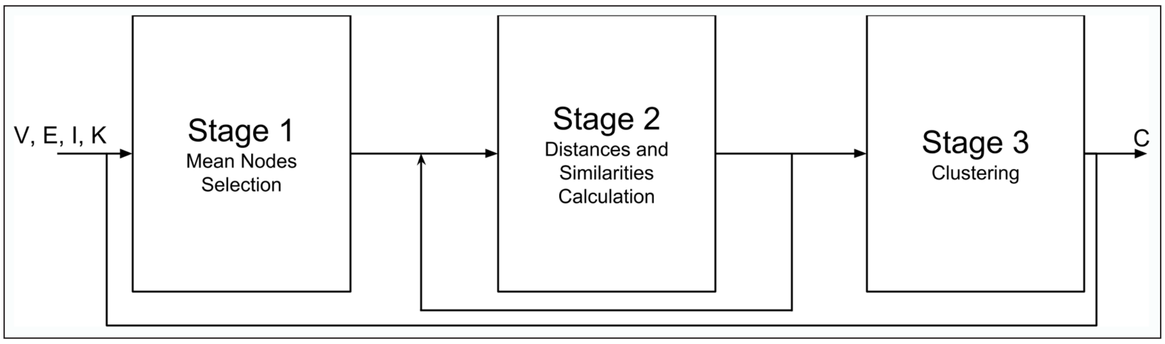

Figure 1: The three stages of the proposed GeoSim algorithm, where *V* is the set of all nodes in the social network, *E* is the set of all edges in the network, *I* is the set of all interests for all nodes and *K* is the number of communities.

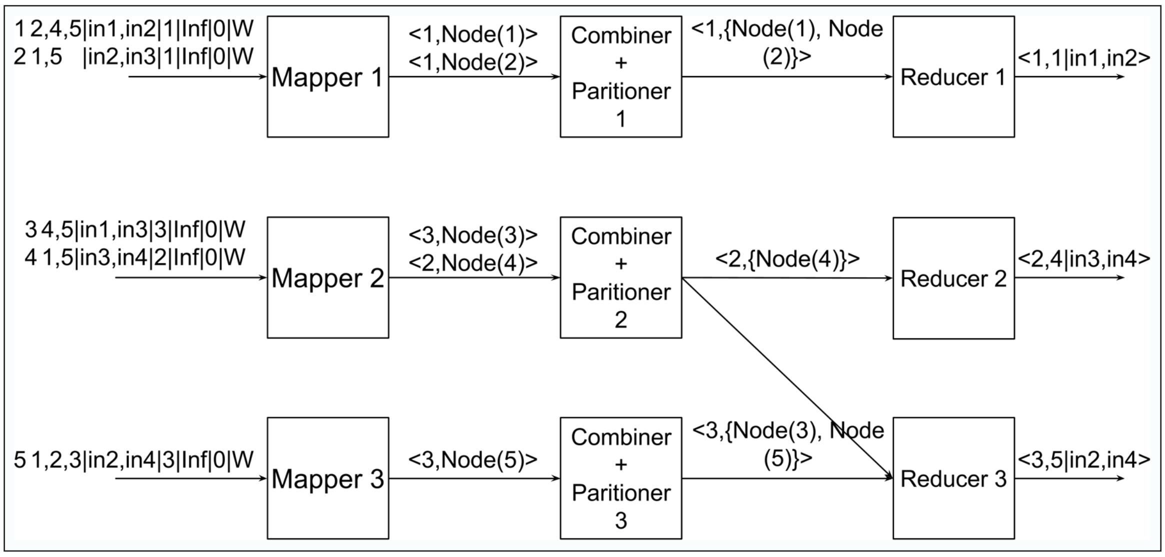

Figure 2: An example of the map and reduce phases of the Mean Nodes Selection stage of the GeoSimMR algorithm.

Algorithm 2: Mean Nodes Selection Stage.

Input: *input_file:* The input file, a file containing the nodes information.

Output: *output_file:* A file containing a set key-value pairs, the key is the community id and the value is the node id and its interests.

```
1:  procedure MAP(line)
2:      node = parseNode(line)
3:      node.calculateFriendsNumber()
4:      emit(node.communityId, node)
5:  end procedure
6:
7:  procedure REDUCE(communityId, nodesList)
8:      maxFriends = 0
9:      for node ∈ nodesList do
10:         if node.friendsNumber > maxFriends then
11:             meanNode = node
12:             maxFriends = node.friendsNumber
13:         end if
14:     end for
15:     emit(communityId, node.printIdAndInterests())
16: end procedure
```

C. Distance and Similarity Calculation

This stage is the core of the GeoSimMR since all the necessary calculations are done here. The input to this stage is the same as the previous one in addition to the mean nodes obtained in the Mean Node Selection stage. The input file is split into lines and fed to the mappers. In the map phase, each mapper receives a group of lines representing the nodes as well as all the mean nodes and their interests. For each selected node, the mapper performs two operations. The first operation is the calculation of the similarities, using equation (3), between the current node and all the mean nodes. Since the output of the previous stage includes the interests of each mean node, finding the similarities here is not an issue.

The second operation for the mapper involves the distance calculation. Recall that Dijkstra's algorithm, for a given source node, finds the shortest paths from that source node to all nodes in the graph. The source nodes in the GeoSimMR are the mean nodes. The mapper knows whether an input node is a source node or not by checking its ID against the IDs in the mean nodes list. If a certain node is found to be a mean node, the mapper starts the shortest path calculation between this node and all nodes in the graph.

Note that this process happens for each mean node in the graph. So for a certain node *n*, it can be the source node in community *p* but not in community *q*. That is, each node gets processed *K* times, where *K* is the number of communities.

It is worth mentioning that when a node is emitted in the map phase, the key value has two parts. The first part of the key is the community id and the second part is the node id. The value of the pair is the node itself.

In the reduce phase, the reducer receives a list of distances for each node. These distances represent the cost of the path so far from this node to the mean node. The role of the reducer is to select the smallest distance and assign it to the node. Then, the reducer emits the node with its information along with the updated distance.

As stated before, this stage is an iterative process. This is because all nodes must be visited in a sequential manner in order to find the shortest paths from all mean nodes to all nodes in the graph. So, the stage keeps on traversing and updating the nodes distances until all of them have been visited.

One thing to note here is that operations in the first iteration and the rest differ. In the first iteration, both the similarity and the distance are calculated. Since the mean nodes remain the same, one mean nodes selection at each main iteration; for the rest of the iterations and therefore their interests, there is no need to calculate the similarities again.

Figure 3 shows an example of the second stage, which consists of the map and reduce phases of the same data set shown in the example of **Figure 2**. Each mapper receives a group of nodes to process them as well as the mean nodes found in the first stage. The first line received by the first mapper holds the information about node 1. The mapper processes this node across all communities (1, 2 and 3). Recall from **Figure 2** that node 1 is the mean node for community 1. This means that node 1 is the source node for community 1 and the mapper will loop through all of its friends. As shown in **Figure 3**, the value of the distance field of node 1 changes from inf to 0. The similarity is one since we are comparing this node to the mean which is itself. The *v* value is changed to B and the node is emitted. As for the friends nodes, their distance value changes to the distance value of node 1 plus one. Since node 1 has a distance value of zero, all of its friends will have a distance of one. The value of *v* is changed to G so it will be processed in the next iteration. Note that the values for fields interests, similarity and friends are marked NA. This is because this information is not available to the mapper and will get filled at the reducer in the pick and merge step discussed earlier.

Once looping through the friends is done, the same node, node 1, is processed again against community 2 and 3, as shown in the figure. Node 1 is not the mean node for community 2. In this case, only the similarity between node 1 and the mean node for this community is calculated and then the node is emitted.

When node 1 is emitted for community 1, the key consists of the community id and the node id. Similarly, when node 1 is emitted for community 2 the key is {2, 1}.

At the reduce stage, reducer 1 receives several lists, one of them is a list of node 4 for community 1. In **Figure 3**, there are two records for the key {1, 4}, of which the first record has updated values for the distance

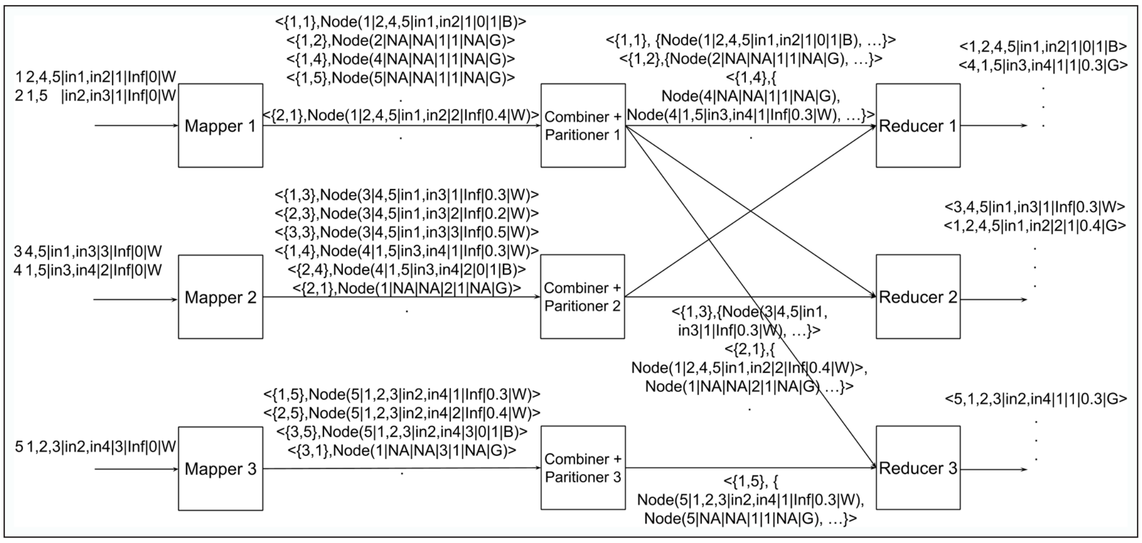
1 2,4,5|in1,in2|1|Inf|0|W
2 1,5 |in2,in3|1|Inf|0|W
Mapper 1
<{1,1},Node(1|2,4,5|in1,in2|1|0|1|B)>
<{1,2},Node(2|NA|NA|1|1|NA|G)>
<{1,4},Node(4|NA|NA|1|1|NA|G)>
<{1,5},Node(5|NA|NA|1|1|NA|G)>
<{2,1},Node(1|2,4,5|in1,in2|2|Inf|0.4|W)>
3 4,5|in1,in3|3|Inf|0|W
4 1,5|in3,in4|2|Inf|0|W
Mapper 2
<{1,3},Node(3|4,5|in1,in3|1|Inf|0.3|W)>
<{2,3},Node(3|4,5|in1,in3|2|Inf|0.2|W)>
<{3,3},Node(3|4,5|in1,in3|3|Inf|0.5|W)>
<{1,4},Node(4|1,5|in3,in4|1|Inf|0.3|W)>
<{2,4},Node(4|1,5|in3,in4|2|0|1|B)>
<{2,1},Node(1|NA|NA|2|1|NA|G)>
5 1,2,3|in2,in4|3|Inf|0|W
Mapper 3
<{1,5},Node(5|1,2,3|in2,in4|1|Inf|0.3|W)>
<{2,5},Node(5|1,2,3|in2,in4|2|Inf|0.4|W)>
<{3,5},Node(5|1,2,3|in2,in4|3|0|1|B)>
<{3,1},Node(1|NA|NA|3|1|NA|G)>
Combiner + Paritioner 1
Combiner + Paritioner 2
Combiner + Paritioner 3
<{1,1}, {Node(1|2,4,5|in1,in2|1|0|1|B), …}>
<{1,2},{Node(2|NA|NA|1|1|NA|G), …}>
<{1,4},{
Node(4|NA|NA|1|1|NA|G),
Node(4|1,5|in3,in4|1|Inf|0.3|W), …}>
<{1,3},{Node(3|4,5|in1,
in3|1|Inf|0.3|W), …}>
<{2,1},{
Node(1|2,4,5|in1,in2|2|Inf|0.4|W)>,
Node(1|NA|NA|2|1|NA|G) …}>
<{1,5}, {
Node(5|1,2,3|in2,in4|1|Inf|0.3|W),
Node(5|NA|NA|1|1|NA|G), …}>
Reducer 1
Reducer 2
Reducer 3
<1,2,4,5|in1,in2|1|0|1|B>
<4,1,5|in3,in4|1|1|0.3|G>
<3,4,5|in1,in3|1|Inf|0.3|W>
<1,2,4,5|in1,in2|2|1|0.4|G>
<5,1,2,3|in2,in4|1|1|0.3|G>

Figure 3: An example of the map and the reduce phases of the Distances and Similarities Calculation stage of the GeoSimMR algorithm.

and the *v* fields. The reducer merges these two records into one node by taking the most recent values from each one. The reducer then emits the node in the same format as the original input. **Algorithm 3** summarizes the map and reduce phases of the Distances and Similarities Calculation stage.

Algorithm 3: Distances and Similarities Calculation Stage.

Input: *input_file*: The input file which contains the nodes information., *meanNodes*: A list of mean nodes obtained from the Mean Nodes Selection stage.

Output: *output_file*: A file containing the distances and the similarities from each node to all mean nodes.

```
1:  procedure MAP(line)
2:      node = parseNode(line)
3:      for meanNode ∈ meanNodes do
4:          if isFirstIteration then
5:              node.calculateSimilarity(meanNode)
6:          end if
7:          if node == meanNode || node.v == G then
8:              if node == meanNode then
9:                  node.distance = 0
10:             end if
11:             for friendinnode.friends do
12:                 friend.v = G
13:                 friend.distance = node.distance + 1
14:                 emit({meanNode.communityId,
15:                 friend.id}, friend)
16:             end for
17:         end if
18:         emit({meanNode.communityId, node.id}, node)
19:     end for
20: end procedure
21:
22: procedure REDUCE(communityId, nodeId, nodesList)
23:     outNode.communityId = nodeId
24:     outNode.id = communityId
25:     outNote.friends = getFriends(nodesList)
26:     outNote.interests = getInterests(nodesList)
27:     for node ∈ nodesList do
28:         if outNode.distance > node.distance then
29:             outNode.distance = node.distance
30:         end if
31:         if outNode.similarity < node.similarity then
32:             outNode.similarity = node.similarity
33:         end if
34:         if outNode.v < node.v then
35:             outNode.v = node.v
36:         end if
37:     end for
38:     emit (outNode.id, outNode.printInfo())
39: end procedure
```

D. Clustering Stage

As discussed earlier, in this stage, the calculated distances and similarities are used to perform the actual clustering using equation (1). The input to this stage is the output of the Distance and Similarity Calculation stage. In the map stage, each mapper receives a group of nodes with the distances and similarity already calculated. The mapper uses equation (1) to calculate the *GSS* for the node to the assigned community. The node is then emitted to the reducer. The key-value pair consists of the node itself as the key and the *GSS* and the community as the value.

In the reduce phase, each reducer receives the node with a list of *GSS*s, each corresponds to a certain community. The job of the reducer is to associate the node with the community that resulted in the lowest *GSS*. The reducer emits the final node in the same format as the original input.

Figure 4 shows an example for the Clustering stage. The shown input to mapper 1 are the nodes 1 and 4. Mapper 1 calculates the *GSS* of both nodes and emits them to the reducer. Reducer 1 receives the *GSS*s of node 1. Node 1 in this example has *GSS* equals to 0 for community 1 and 0.45 for community 2. The reducer assigns node 1 to community 1 since it has the lowest *GSS*. Note that the reducer is also resetting all fields to their initial values (except for the community id) to prepare it for the next iteration. **Algorithm 4** summarizes the Clustering stage.

VI. Experiments

In this section, we show the results obtained using the GeoSim and GeoSimMR algorithms. GeoSim is experimented to see how well it detects the communities in a given social network. The GeoSimMR is tested to see the scalability performance when executed on big networks on top of MapReduce framework. The experiments are conducted using a three-node cluster. Each workstation has 22GB of system memory. Three of these machines are powered by an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz whereas the last workstation is powered by an Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60GHz. In total, we have 42 mappers and 21 reducers. Apache Hadoop version 2.7 is used for testing the MapReduce version of GeoSim algorithm.

A. GeoSim Results

The GeoSim is tested to show the accuracy of the algorithm in detecting communities. For this test, we used a relatively small network to facilitate the analysis. The network consists of 50 nodes with 74 different interests distributed across all nodes with four to five interests per node. The 74 interests can be grouped into five main categories: Languages, Medicine, Computer, Professions, Sports. The distribution of interests is not random, each ten-node group is assigned with interests under the same main category. For instance, the nodes from 0 to 9 have interests from the Languages group and 10 to 19 from Medicine and so on. We setup the input network this way to get an indication of how accurate the output of the algorithm is. The results of running GeoSim over the network is shown in **Figure 5**.

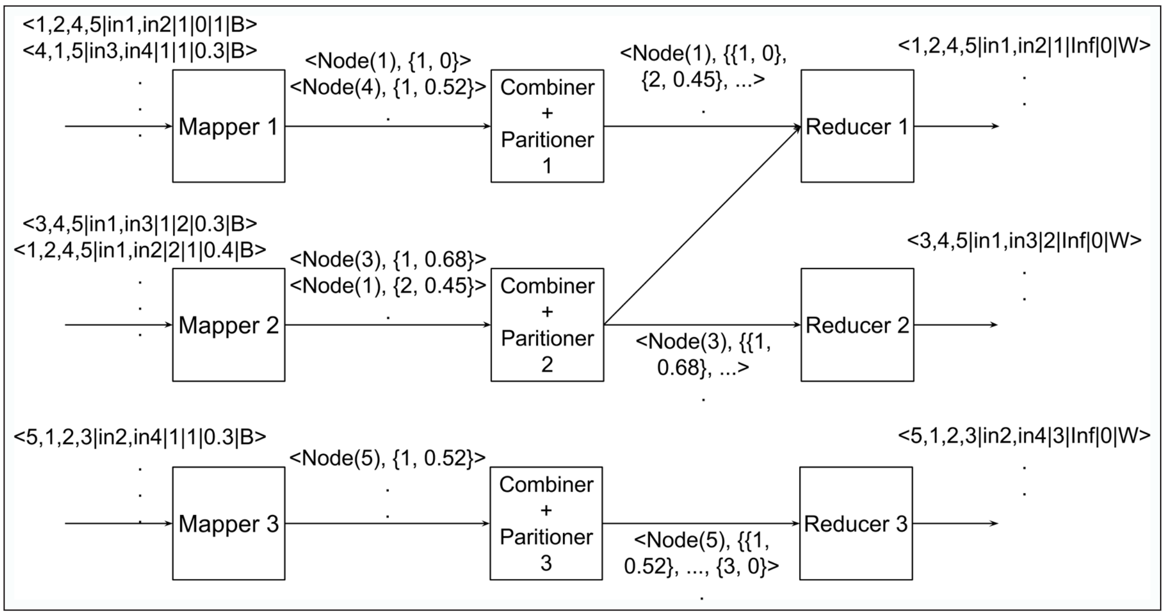
<1,2,4,5|in1,in2|1|0|1|B>
<4,1,5|in3,in4|1|1|0.3|B>
Mapper 1
<Node(1), {1, 0}>
<Node(4), {1, 0.52}>
Combiner + Paritioner 1
<Node(1), {{1, 0}, {2, 0.45}, ...>
Reducer 1
<1,2,4,5|in1,in2|1|Inf|0|W>
<3,4,5|in1,in3|1|2|0.3|B>
<1,2,4,5|in1,in2|2|1|0.4|B>
Mapper 2
<Node(3), {1, 0.68}>
<Node(1), {2, 0.45}>
Combiner + Paritioner 2
<Node(3), {{1, 0.68}, ...>
Reducer 2
<3,4,5|in1,in3|2|Inf|0|W>
<5,1,2,3|in2,in4|1|1|0.3|B>
Mapper 3
<Node(5), {1, 0.52}>
Combiner + Paritioner 3
<Node(5), {{1, 0.52}, ..., {3, 0}>
Reducer 3
<5,1,2,3|in2,in4|3|Inf|0|W>

Figure 4: An example of the map and reduce phases of the Clustering stage of the GeoSimMR algorithm.

Algorithm 4: Clustering Stage.

Input: *input_file*: The output file of the Distances and Similarities Calculation.
Output: *output_file*: List of all nodes and their detected communities.

```
1:  procedure MAP(line)
2:      node = parseNode(line)
3:      Calculate GSS using equation 1
4:      emit(node, GSS)
5:  end procedure
6:
7:  procedure REDUCE(communityId, communityGSSPairs)
8:      minScore = inf
9:      for pair ∈ communityGSSPairs do
10:         if pair.GSS < minScore then
11:             minScore = pair.GSS
12:             communityId = pair.communityId
13:         end if
14:     end for
15:     node.communityId = communityId
16:     emit(node.id, node.printInfo())
17: end procedure
```

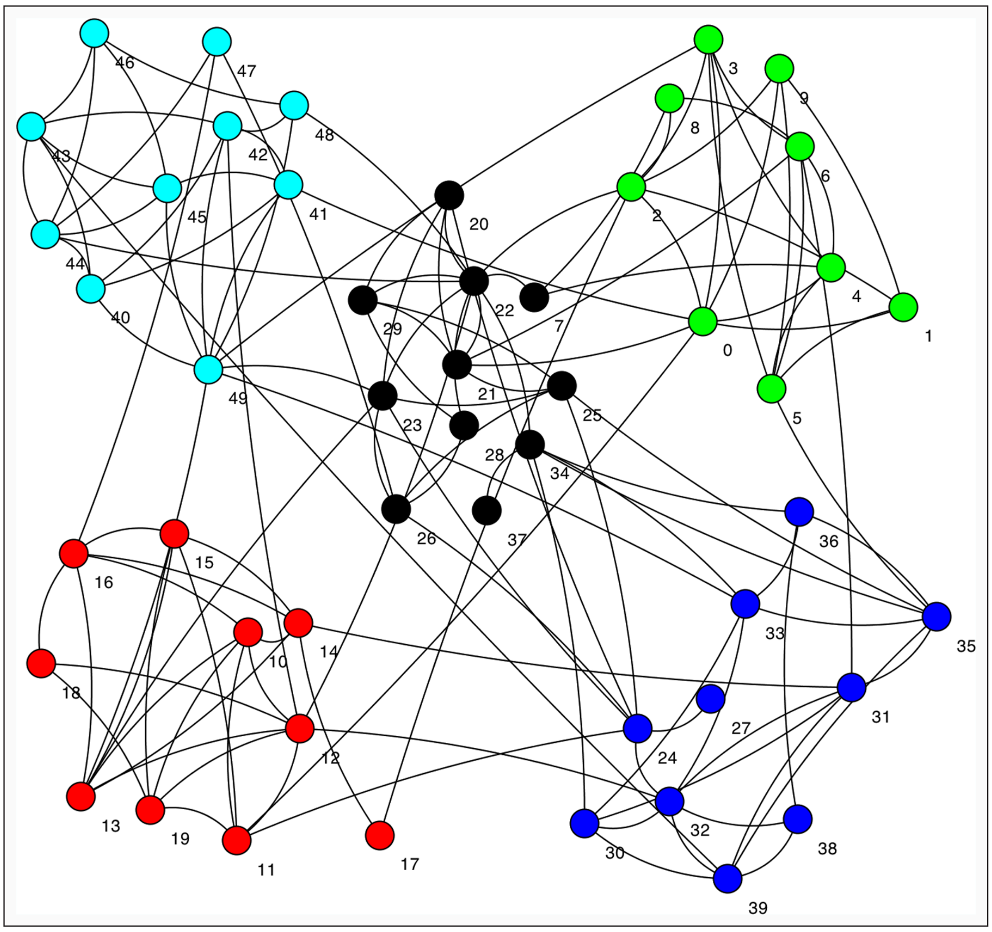

Figure 5: The detected communities in the network by the GeoSim algorithm ($\alpha = 0.7$).

Figure 5 shows the detected communities where each color represents a single community. In this test the α variable (from equation (1)) is set to be 0.7. It is clear from **Figure 5** that the community detection accuracy of GeoSim is high as similar and close nodes are grouped with each other. For example, nodes from 0 to 9 (except node 7) are grouped together as one community. This is because they have similar interests and they are very close to each other. As for node number 7, it is grouped with the black community instead of the green one despite having interests from the same category as the nodes from 0 to 9 (the green community). The reason behind this is that node 7 is far from the mean node of the green community (node 0) and it is only one hop away from the mean node of the black community (node 22), so GeoSim groups it with the black community.

Another interesting case is with node number 42 in the cyan community. This node is connected to two mean nodes: node 49 (for the cyan community) and node 12 (for the red community). So even though node 42 is directly connected to two mean nodes, it was grouped with 49 rather than 12. The reason behind this is that the mean node 49 has more interests in common with node 42 than the mean node 12.

Another experiment was carried out on a real graph from the DBLP database. A total of 30 nodes were selected: nodes from 0 to 9 published papers related to database, nodes with ids from 10 to 19 published papers related to data mining and nodes from 29 to 30 published papers related to artificial intelligence. **Table 1** shows the authors and their respective fields. A link between two nodes means that these nodes coauthored a paper together. In this test, α is set to 0.6. Results for this test are shown in **Figure 6**. From the results, the GeoSim algorithm made three communities: the green community consists of authors who have interests related to database, the red community has authors that are interested in data mining, and the black community comprises of researchers who are into artificial intelligence. According to the proposed GeoSim algorithm, in the initial iteration, the nodes with the highest degree are selected as the mean nodes. Therefore, the mean node for the database community is Timos K. Sellis since he has the highest number of coauthors in the dataset. For the data mining, Christos Faloutsos is the mean node and Wei Liu is the mean node for the artificial intelligence community. It is clear from the results that GeoSim achieved accurate results in determining the community for each node. For example, authors V. S. Subrahmanian, Vassilis J. Tsotras and Raymond T. Ng have been clustered into the same community, since they are interested in database-related topics, shown in **Table 1**. They are also close to each other in terms of connection and their distance to the mean node.

B. GeoSimMR results

We have conducted several experiments to test the scalability of the GeoSimMR algorithm. In these experiments, the input data consists of 100,000 nodes, where each node has four to five skills.

The first experiment examines the different execution times of the GeoSimMR when varying the number of mappers and reducers. **Figure 7** shows the results of this experiment.

Figure 7 shows the execution times for the GeoSimMR using different number of mappers and reducers. The x-axis represents the number of mappers and the y-axis indicates the time of execution in seconds. As shown in **Figure 7**, the execution time of the GeoSimMR decreases drastically as the number of mappers

Table 1: DBLP authors used as a ground truth in the community detection experiment.

Database	Data Mining	Artificial Intelligence
V. S. Subrahmanian	Alex Beutel	Donald Perlis
David j. DeWitt	Rakesh Agrawal	Mary Anne Williams
Michael Stonebraker	Ramakrishnan Srikant	Wei Liu
Hans Peter Kriegel	Christos Faloutsos	Keith Johnson
Timos K. Sellis	Flavio Figueiredo	Sanjay Chawla
Laura M. Haas	Bruno Ribeiro	James Bailey
H. V. Jagadish	Bussara M. Almeida	Christopher Leckie
Vassilis J. Tsotras	Yasuko Matsubara	Kotagiri Ramamohanarao
Raymond T. Ng	lei li	Daren Ler
Christian S. Jensen	Evangelos E. Papalexakis	Irena Koprinska

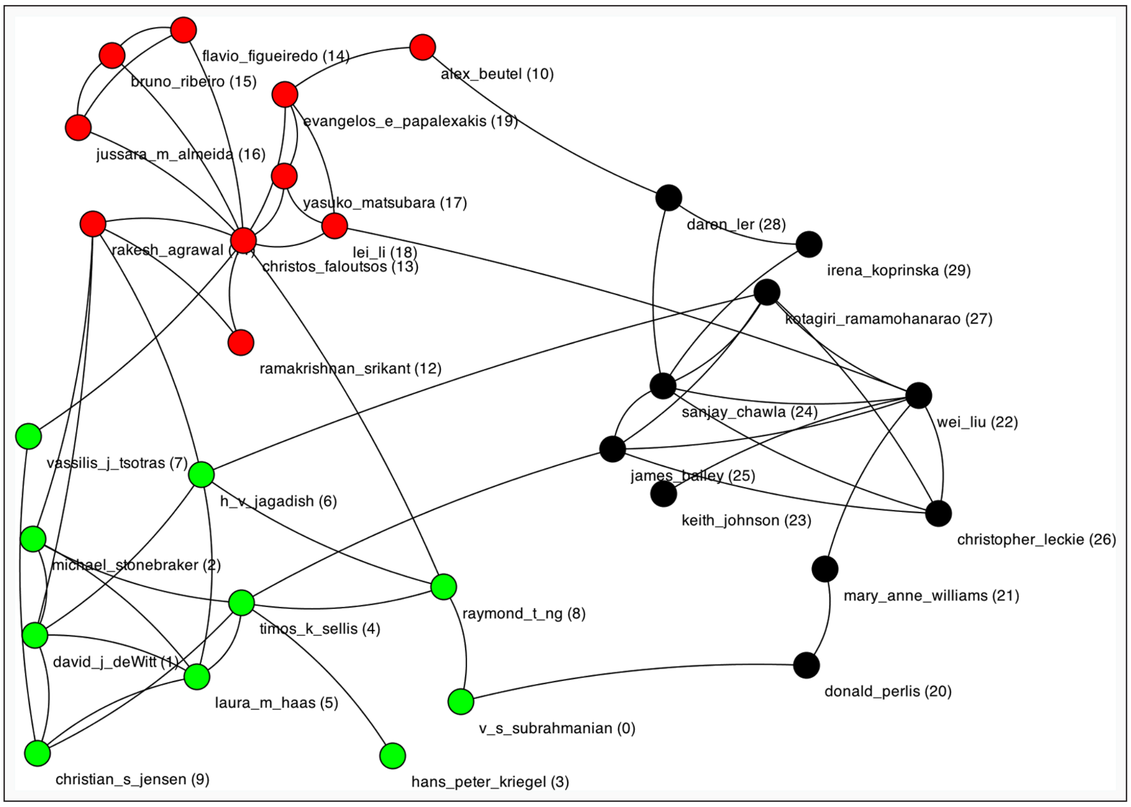

Figure 6: Detected communities in a DBLP graph.

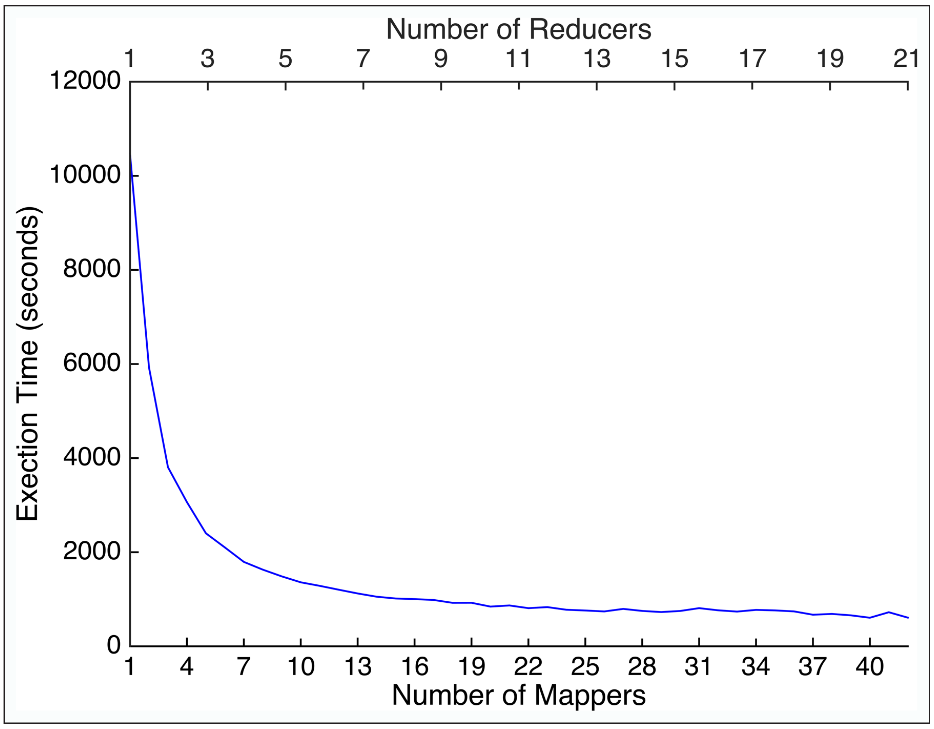

Figure 7: Execution time for the GeoSimMR using different number of mappers and reducers.

increases. When only one mapper is used, GeoSimMR took almost 10,500 seconds to finish executing, whereas adding another mapper to the setup cuts the execution time by almost half. As the number of mappers increases, the execution time continues to decrease to be 609 seconds when 42 mappers and 21 reducers are used, which resulted in 94% improvement.

As shown in **Figure 7**, adding more mappers to the setup clearly decreases the execution time. This keeps on decreasing until adding more mappers and reducers; e.g. around 37 mappers in **Figure 7**, makes a very subtle or no change at all in the execution time. The reason behind this is that, for the dataset of 100,000 nodes, the number of mappers and reducers needed to execute the algorithm efficiently has been reached and adding more will cause no improvement. That is, the improvement in execution time is canceled out by the communication overhead of adding new mappers and reducers.

Recall from a previous section, the GeoSimMR algorithm consists of three stages: Mean Nodes Selection, Distance and Similarity Calculation and Clustering stages. To further analyze the GeoSimMR algorithm, the execution time of each stage is presented.

Figure 8 shows the execution time for the Mean Nodes Selection stage using a different number of mappers and reducers. The x and y-axes represent the number of mappers and the execution time in seconds, respectively. As shown in **Figure 8**, the first stage takes between 10 to 18 seconds to finish executing.

Figure 9 shows the execution time for the Distance and Similarity Calculation stage using various number of mappers and reducers. The number of mappers is represented by the x-axis and the execution time in seconds is represented by the y-axis. When using only one mapper, the second stage takes about 10,000 seconds to finish executing. Adding one more mapper decreased the execution time to 5,700 seconds, which accounts for 43% performance increase. When 42 mappers were used, the time of execution was 570 seconds, which is about 94% better performance than using only one mapper.

In contrast to the execution time of the first stage, which showed minor variation when changing the number of mappers, the second stage clearly benefited a lot from the parallelism. In fact, the execution time of the second stage is very close to the execution time of all stages combined. That is 96% of the whole time was spent on executing the second stage. This is due to the fact that all the necessary calculations, distances and similarities are performed in this stage.

Note that the distance and similarity calculation stage is an iterative phase. To analyze this stage even further, the execution time for each iteration is examined. **Figure 10** shows the execution time for each iteration of the second stage. The x-axis represents the iteration number whereas the y-axis represents the execution time. For 100,000 nodes, the second stage takes 13 iterations to finish calculating the distances between the mean nodes and each node in the graph. The execution time shown in **Figure 10** is the average of execution times using 1 mapper/1 reducer to 42 mappers/21 reducers. The first iteration takes about

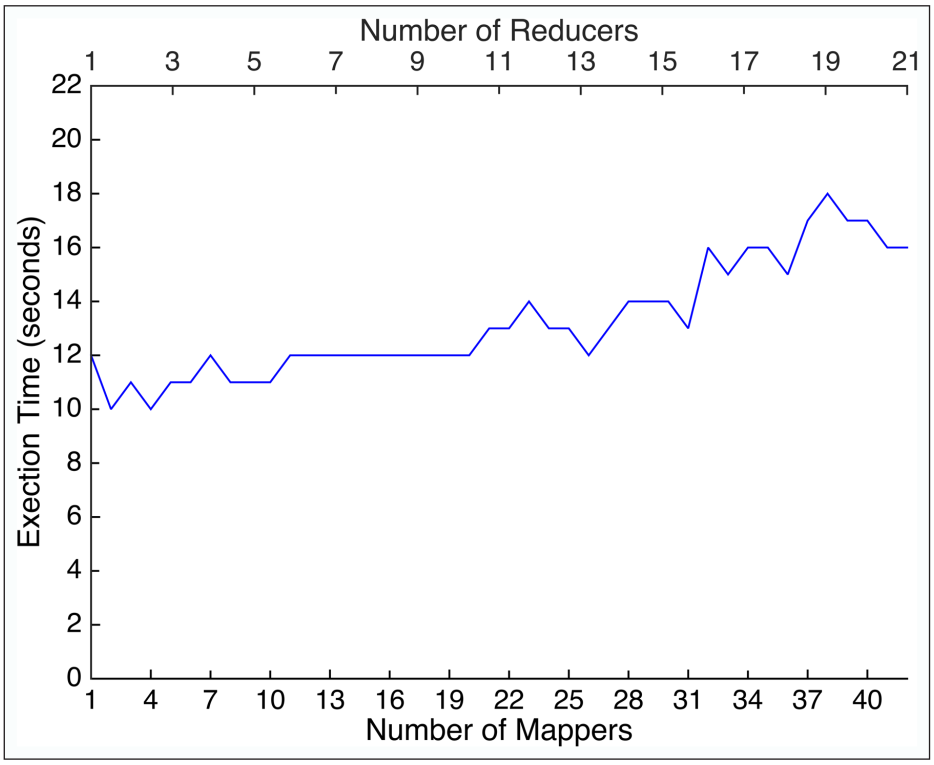

Figure 8: The execution time of the Mean Nodes Selection stage using different number of mappers and reducers.

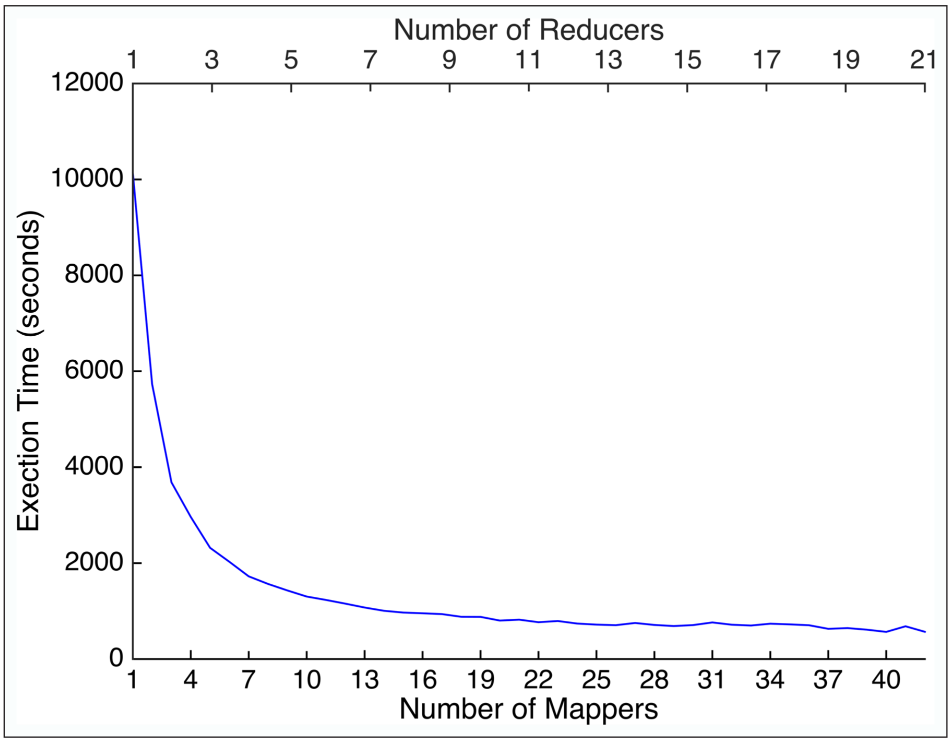

Figure 9: The execution time of the Distances and Similarities Calculation stage using different number of mappers and reducers.

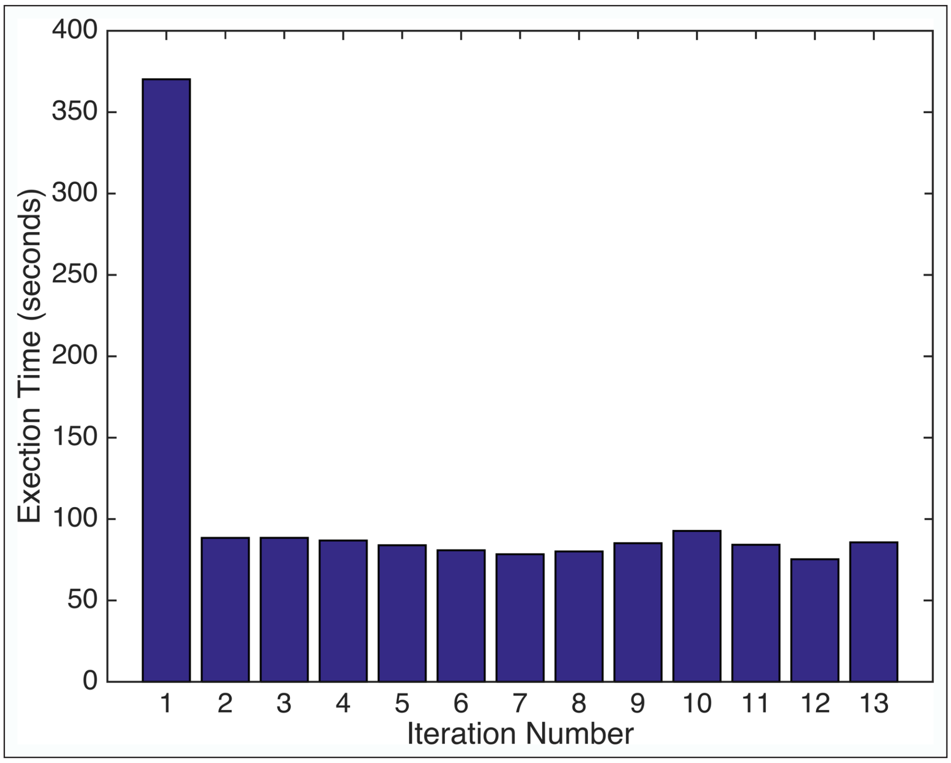

Figure 10: The execution time for every iteration of the second stage of the GeoSimMR.

370 seconds, whereas the rest of the iterations take almost the same time, which is about 80 seconds. The first iteration always requires larger execution time because the distances, as well as the similarities, are calculated in this iteration. By the end of the first iteration, all of the requireed similarities are obtained; thus, the other iterations will continue calculating the distances only.

Figure 11 Shows the execution time of the clustering stage using different number of mappers and reducers. The x-axis indicates the number of mappers whereas the y-axis represents the execution time in seconds. Using only one mapper and one reducer, the third stage takes approximately 300 to finish executing. In case of two mappers and one reducer, the clustering stage takes 185 seconds, which is an improvement of 38% as compared to using one mapper. When 42 mappers and 21 reducers were used, the execution time of the third stage dropped to 25 seconds giving 92% performance increase in execution time.

It is clear that the execution time for the clustering stage does not take as much as the second stage. The reason behind this is that the third stage does not involve any complex calculation. Recall from the previous section, the third stage uses the distances and similarities obtained from the second stage to group the nodes into communities.

Two more experiments were conducted to analyze the effect of changing the number of the mappers and reducers. **Figure 12** shows the execution time of the GeoSimMR using 18 reducers and changing the mappers from 1 to 42. The x-axis represents the number of mappers and the y-axis indicates the execution time in seconds. The GeoSimMR takes 11,000 seconds when only one mapper is used and 5,700 seconds when two mappers are used. The execution time drops to 1062 seconds by using 42 mappers.

It is clear that when the number of mappers is small, adding another mapper drastically improves the execution time. For instance, in case of one-mapper, the algorithm takes about 11,000 seconds to finish. Adding another mapper to the first one reduces the execution time to 5600 seconds (about 50% improvement). As the number of mappers increases, the effect of the added mappers starts to fade away. The reason behind this behavior is that adding new mappers while the number of reducers is fixed will cause more results to come out of the map stage. This leads to a point where the current number of reducers cannot handle that amount of data from the map stage. In other words, the reducers becomes the bottleneck of the whole job.

Figure 13 also shows the results of executing the GeoSimMR algorithm using 42 mappers and varying the number of reducers from 1 to 21. The x-axis represents the number of reducers while the y-axis represents the execution time in seconds. When only one reducer was used, the execution time is around 1900 seconds. Adding one more reducer improves the execution time by 26%. When all the 21 reducers are used the execution time is 942 seconds, achieving 50% better performance.

Another experiment was conducted to show the gain obtained from using MapReduce. For the sake of comparison, we created centralized version where all stages are re-coded and optimized to run on a single

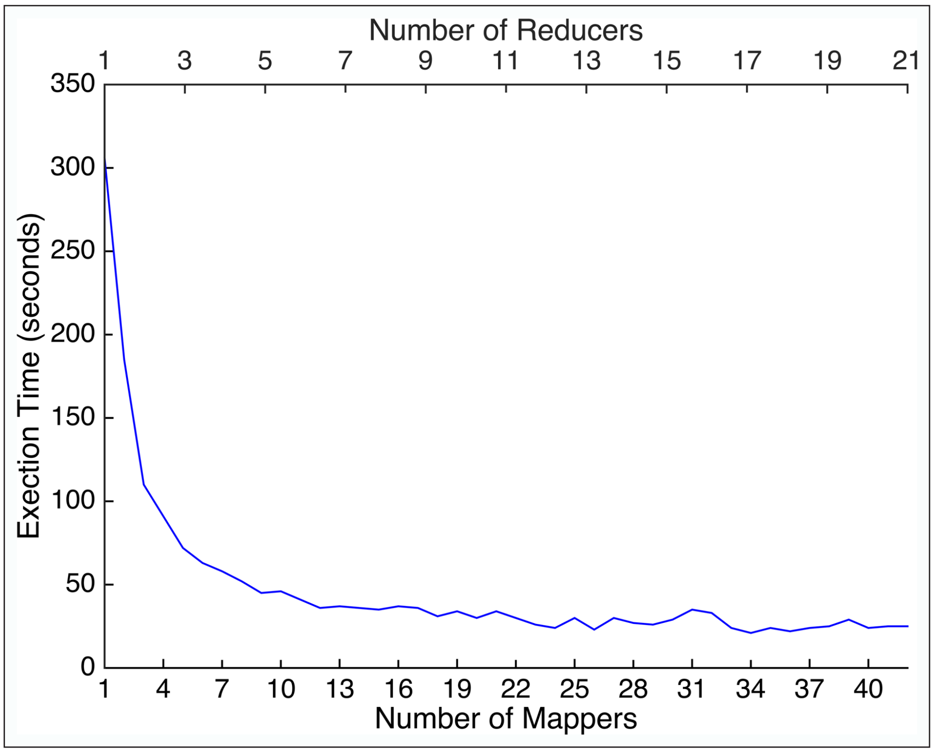

Figure 11: The execution time for the Clustering stage using different number of mappers and reducers.

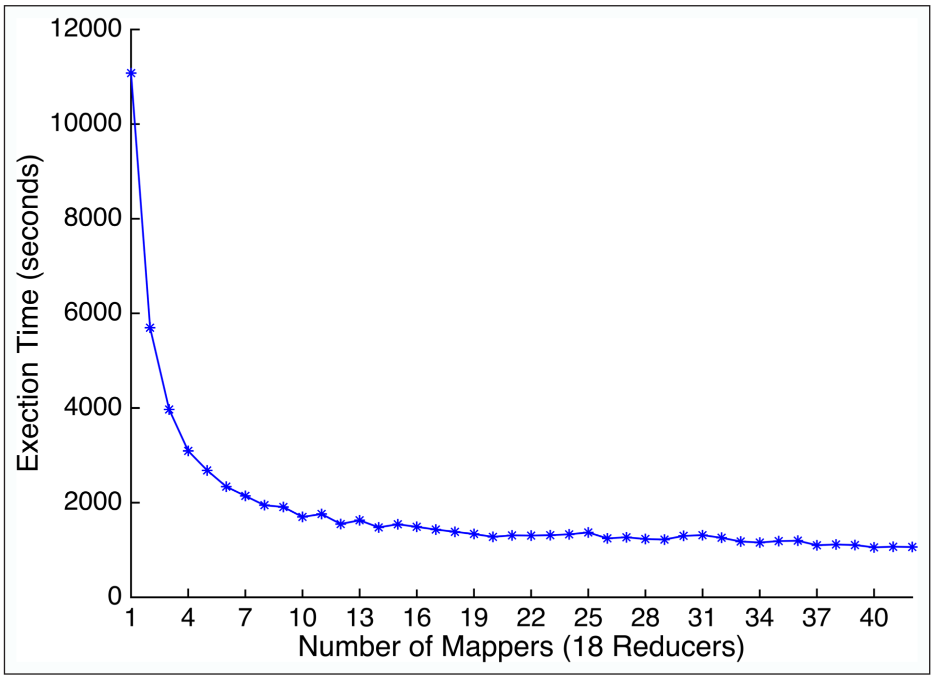

Figure 12: The overall execution time for different mappers while fixing the number of reducers.

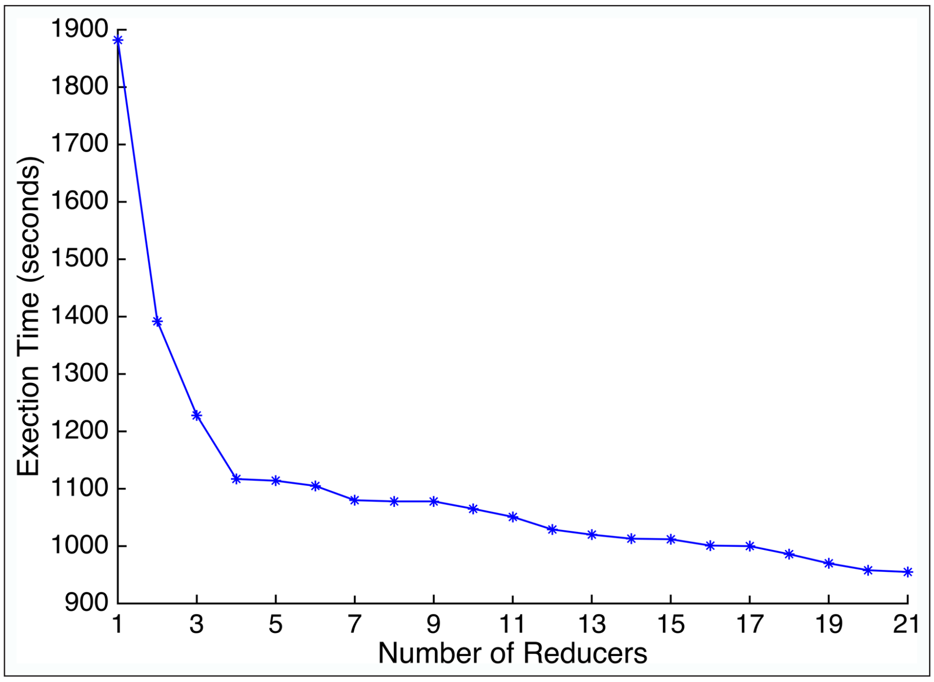

Figure 13: The overall execution time for different reducers while fixing the number of mappers to 42.

machine without the use of MapReduce. We ran the MapReduce version (with 42 mappers and 21 reducers) and the centralized version to show the time difference between the two versions. Five datasets are used in this experiment. The datasets have the following number of nodes: 100 k, 250 k, 1 m, 5 m and 10 m each clustered into 100 communities. The results of this test are shown in **Figure 14**.

In **Figure 14**, the x-axis indicates the number of nodes and the y-axis represents the execution time in seconds (logarithmic scale). For 100,000 nodes, the centralized version of the algorithm takes 246 seconds to finish whereas the MapReduce version finished after 1056 seconds. For higher the number of nodes, the centralized version starts to take a very long time, so an estimation of the execution time is proposed. Since GeoSimMR algorithm processes each node individually, we measure the execution time for processing

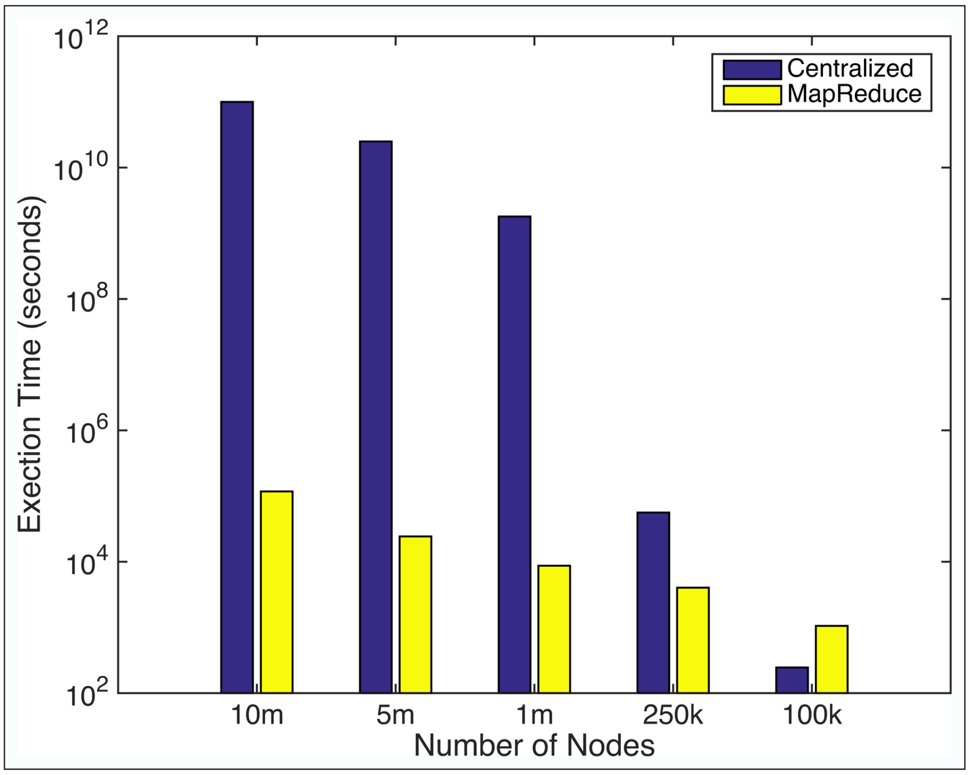

Figure 14: The overall execution time for the GeoSim and GeoSimMR using different number of nodes (Time axis is semi-log).

few nodes and then calculated the average time for processing one node. This time is then multiplied by the number of the nodes in the graph to get the estimated execution time. When the number of nodes is 10 million, the MapReduce version takes 32 hours to finish, whereas the centralized version needs approximately 3170 years to complete execution.

For a smaller number of nodes, for example, 100,000 nodes, the MapReduce version of the algorithm takes a longer time (around 73% more time) than its centralized version. This is because the overhead of the communication between the mappers and reducers along with the cost of read/write operations are much higher than the computations for this number of nodes. When the number of nodes increases to 1,000,000 nodes, the overhead time becomes very small and negligible compared to the time required for the computation.

VII. Conclusion

In this paper, we presented the GeoSim algorithm that clusters any given social network into communities. Unlike previous community detection techniques, where only the network structure is taken into consideration, GeoSim involves the interests of each node into the clustering process. This paper also introduced a parallel version of GeoSim, namely GeoSimMR, that runs on top of a MapReduce framework. Having a parallel community detection algorithm is very crucial since social networks are typically very huge and a single machine is inadequate to process them. Both GeoSim and GeoSimMR have been examined and the results were presented in the paper. GeoSim achieved high accuracy in terms of detecting communities by grouping close nodes that have similar interests into a single community. In addition to the high accuracy of community detection, the GeoSimMR algorithm takes advantage of available mappers and reducers in order to reduce the execution time drastically.

In the future, we plan to implement our algorithms using other Big Data platforms such as Spark and conduct an analysis and comparison with the MapReduce platform. Additionally, it would be interesting to use these Big Data implementations to find the influential nodes in the detected communities.

Competing Interests

The authors have no competing interests to declare.

References

Altunbey, F and **Alatas, B.** 2015. Overlapping community detection in social networks using parliamentary optimization algorithm. *International Journal of Computer Networks and Applications*, 2(1): 12–19.

Blondel, VD, Guillaume, J-L, Lambiotte, R and **Lefebvre, E.** 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: Theory and experiment*, 2008(10). DOI: https://doi.org/10.1088/1742-5468/2008/10/P10008

Capuruco, RAC and **Capretz, LF.** 2010. Integrating Recommender Information in Social Ecosystems Decisions. *4th European Conference on Software Architecture*, 143–150. DOI: https://doi.org/10.1145/1842752.1842783

Capuruco, RAC and **Capretz, LF.** 2012. A Fuzzy-Based Inference Mechanism of Trust for Improved Social Recommenders. *3rd International Workshop on Social Recommender Systems.*

Clauset, A, Newman, MEJ and **Moore, C.** Dec 2004. Finding community structure in very large networks, *Phys. Rev. E*, 70. DOI: https://doi.org/10.1103/PhysRevE.70.066111

Dean, J and **Ghemawat, S.** January 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun., ACM*, 51(1): 107–113. DOI: https://doi.org/10.1145/1327452.1327492

Ghosh, R and **Lerman, K.** 2010. Community Detection Using a Measure of Global Influence. *Proceedings of the Second International Conference on Advances in Social Network Mining and Analysis, ser.SNAKDD'08*, 20–35. Berlin, Heidelberg: Springer-Verlag. DOI: https://doi.org/10.1007/978-3-642-14929-0_2

Gregory, S. 2007. An Algorithm to Find Overlapping Community Structure in Networks. *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, ser. PKDD 2007*, 91–102. Berlin, Heidelberg: Springer-Verlag. DOI: https://doi.org/10.1007/978-3-540-74976-9_12

Gregory, S. 2008. A Fast Algorithm to Find Overlapping Communities in Networks. *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases – Part I, ser. ECML PKDD '08*, 408–423. Berlin, Heidelberg: Springer-Verlag. DOI: https://doi.org/10.1007/978-3-540-87479-9_45

Kheirkhahzadeh, M, Lancichinetti, A and **Rosvall, M.** 2016. Efficient community detection of network flows for varying Markov times and bipartite networks. *Physical Review E*, 93(3). DOI: https://doi.org/10.1103/PhysRevE.93.032309

Khorasgani, RR, Jiyang, C and **Osmar, RZ.** 2013. Top leaders community detection approach in information networks. In: *Proceedings of the 4th Workshop on Social Network Mining and Analysis, 2010*, 228. ISSN: 2319-7323.

Lin, D. 1998. An Information-Theoretic Definition of Similarity. *Proceedings of the Fifteenth International Conference on Machine Learning, ser ICML '98*, 296–304. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Miller, GA. 1995. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11): 39–41. DOI: https://doi.org/10.1145/219717.219748

Newman, MEJ. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23): 8577–8582. DOI: https://doi.org/10.1073/pnas.0601602103

Newman, MEJ and **Girvan, M.** 2004. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69. DOI: https://doi.org/10.1103/PhysRevE.69.026113

Qi, GJ, Aggarwal, CC and **Huang, T.** Apr 2012. Community Detection with Edge Content in Social Media Networks. In: *Data Engineering (ICDE), 2012 IEEE 28th International Conference*, 534–545.

Qi, X, Tang, W, Wu, Y, Guo, G, Fuller, E and **Zhang, CQ.** 2014. Optimal local community detection in social networks based on density drop of subgraphs. *Pattern Recognition Letters*, 36: 46–53. DOI: https://doi.org/10.1016/j.patrec.2013.09.008

Rosvall, M and **Bergstrom, CT.** 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4): 1118–1123. DOI: https://doi.org/10.1073/pnas.0706851105

Shi, LL, Liu, L, Wu, Y and **Hardy, J.** 2017. Event Detection and User Interest Discovering in Social Media Data Streams. *IEEE Access*, 5: 20953–20964. DOI: https://doi.org/10.1109/ACCESS.2017.2675839

Shi, LL, Wu, Y, Liu, L, Sun, X and **Juang L.** 2018. Event detection and identification of influential spreaders in social media data streams. *Big Data Mining and Analytics*, 1: 34–46. DOI: https://doi.org/10.26599/BDMA.2018.9020004

Sun, Y, Aggarwal, CC and **Han, J.** Jan 2012. Relation Strength-aware Clustering of Heterogeneous Information Networks with Incomplete Attributes. *Proc. VLDB Endow*, 5(5): 394–405. DOI: https://doi.org/10.14778/2140436.2140437

Tang, J, Chang, Y and **Liu, H.** 2014. Mining social media with social theories: a survey. *ACM SIGKDD Explorations Newsletter*, 15(2): 20–29. DOI: https://doi.org/10.1145/2641190.2641195

Tantipathananandh, C, Berger-Wolf, T and **Kempe, D.** 2007. A Framework for Community Identification in Dynamic Social Networks. In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser KDD '07*, 717–726. DOI: https://doi.org/10.1145/1281192.1281269

Tobias, JA, Planqué, R, Cram, DL and **Seddon, N.** 2014. Species interactions and the structure of complex communication networks. *Proceedings of the National Academy of Sciences*, 111(3): 1020–1025. DOI: https://doi.org/10.1073/pnas.1314337111

Zhang, J and **Yu, PS.** 2015. Community detection for emerging networks. *SIAM*. DOI: https://doi.org/10.1137/1.9781611974010.15

How to cite this article: Al Aghbari, Z, Bahutair, M and Kamel, I. 2019. GeoSimMR: A MapReduce Algorithm for Detecting Communities based on Distance and Interest in Social Networks. *Data Science Journal,* 18: 13, pp. 1–20. DOI: https://doi.org/10.5334/dsj-2019-013

Submitted: 03 March 2018 **Accepted:** 02 March 2019 **Published:** 11 April 2019