A Modularity Maximization Algorithm for Community Detection in Social Networks with Low Time Complexity

Mohsen Arab
Department of Computer Science
IASBS
Zanjan, Iran
Email: m_arab@iasbs.ac.ir

Mohsen Afsharchi
Department of Computer Engineering
University of Zanjan
Zanjan, Iran
Email: afsharchim@znu.ac.ir

***Abstract*—Graph vertices are often divided into groups or communities with dense connections within communities and sparse connections between communities. Community detection has recently attracted considerable attention in the field of data mining and social network analysis. Existing community detection methods require too much space and are very time consuming for moderate-to-large networks, whereas large-scale networks have become ubiquitous in real world. We proposed a method that can find communities of a graph with good time and space complexity and good accuracy as well.**

I. Introduction

In recent years, community detection has been in the center of attention due to its wide use in data mining, information retrieval and social network analysis. Most of the complex networks usually have modular or community structure and appear as a combination of groups that are fairly independent of each other.Vertices of the same community usually share some common behaviors. For instance people of the same community usually have a set of common properties such as having similar hobbies, working on a research with the same topic and so on. Thus, finding communities enables us not only to extract useful information of complex networks but also to understand how different groups or communities in a network evolve.

The issue of community detection closely corresponds to the idea of graph partitioning in computer science and graph theory, and hierarchical clustering in sociology. Recently, the computer revolution has provided scholars with a huge amount of data and computational resources to process and analyze these data. The size of real networks one can potentially handle has also grown considerably, reaching millions or even billions of vertices. The need to deal with such a large number of units has produced a deep change in the way that graphs are approached [8]

Since moderate-to-large networks are becoming ubiquitous in our real world, current methods are not satisfactory from the time complexity point of view. In this paper, we present an effective algorithm for finding communities of the graph with a good time and space complexity and also with an acceptable quality of output which is comparable with the existing outputs of recent community detection algorithms. We follow a bottom up approach in which we start community detection by considering every vertex or two vertices as preliminary communities. Then based on a well known criterion which is called ”modularity”, we merge these preliminary communities. Merging is stopped when the maximum modularity achieved.

The structure of the paper is as follows: In the next Section we present a review of the literature. In Section III we provide a detail discussion of our work which is followed by complexity analysis of our work. Finally in Section V we present the result of our experiments. In this paper, We denote the number of the vertices of a graph as n and the number of the edges of the graph as m .

II. Related Works

The most well-known algorithm for community detection was proposed by Girvan and Newman[1]. This method is historically important due to the opening a new era in the field of community detection. This method uses a new similarity measure called *edge betweenness*. Edge betweenness is referred to the number of shortest paths between all vertex pairs that run along that edge. The algorithm has a complexity $O(n^3)$ on a sparse graph. In the following we will refer to it as GN. In another work [10] Newman reformulated modularity in terms of eigenvectors of a new characteristic matrix for the network and called it modularity matrix. He obtained a time complexity $O(n^2 logn)$ for sparse graphs(denoted as N_{eig}).

Clauset et.al. [2] have proposed a fast greedy modularity optimization method. Starting from a set of isolated nodes, the links of the original graph are iteratively added such to produce the maximum possible increase in the modularity of [12] at each step. The algorithm has a complexity of $O(nlog^2 n)$ on sparse graphs.In the following we will refer to it as CNM.

A novel divisive algorithm for modularity maximization is presented by Duch et.al [11]. The total cost of their algorithm is $O(n^2 log^2 n)$. In the following we will refer to it as EO.

Another modularity optimization has been presented by Blondel et.al. [3]. This is a multi-step technique based on the local optimization of Newman-Girvan modularity in the neighborhood of each node. The computational complexity is essentially linear in the number of links of the graph.

With the spirit of Girvan and Newman, Radicchi et. al. have presented another algorithm [14]. In fact, it is a divisive hierarchical method where links are iteratively removed based on the value of their edge clustering coefficient. The algorithm is $O(n^2)$ on a sparse graph.

Cfinder is a local algorithm proposed by Palla et. al. [7] that looks for communities that may overlap. The complexity of this procedure can be high as the computational time needed to find all k-cliques of a graph is an exponentially growing function of the graph size.

Markov Cluster Algorithm is an algorithm developed by S.Van Dongen [4], which simulates a peculiar diffusion process on the graph. The algorithm is $O(nk^2)$ where $k < n$. The structural algorithm is presented by Rosvall and Bergstrom [16]. Here the problem of finding the best cluster structure of a graph is turned into the problem of optimally compressing the information on the structure of the graph, so that one can recover as closely as possible the original structure when the compressed information is decoded.

Donetti and Munoz presented spectral algorithm[5]. The idea is that eigenvector components corresponding to nodes in the same community should have similar values, if communities are well identified. The algorithm is $O(n^3)$.

Expectation-maximization is another algorithm by Newman and Leicht [6]. Here Bayesian inference is used to deduce the best fit of a given model to the data represented by the actual graph structure. The complexity is parameter dependent.

III. Our Work

Our idea for community detection is generally based on finding small communities (i.e. sub-communities) and then merging them in order to obtain real communities of a graph. Like communities, subcommunities are vertices with dense relationship in which most or all of their neighbors are in common.

In this approach, for each subcommunity c_i we try to find a neighbor subcommunity c_j so that merging them will result in increasing the modularity value. If there exist several such neighbor subcommunities, we prefer to choose the one with maximum increase in modularity. As we will discuss later, modularity is one globally accepted criterion for measuring the quality of dividing a graph into communities.

Merging subcommunities must be repeated several times. Although merging all pairs of neighbor communities with highest increase in modularity (i.e. pairwise merging) is a good idea but it is too slow. Merging multiple communities together is more quick but it is less accurate. Therefore, we use both of them and call it "Hybrid" merging. We also use a vertex similarity measure to find small communities which we denote them as *preliminary communities* and then apply the modularity maximization strategy on these preliminary communities that will result in community detection with higher modularity value.

In this paper, *subcommunity* means small *community* and these two terms may be used alternatively. Also if it is not specially mentioned, m is the number of the edges of the graph and n is the number of the vertices

A. Modularity Optimization

In this section, we are going to go more depth in the concept of modularity and rewrite it with more details. This enables us to know that merging which two subcommunities will result in increasing the value of modularity.

Basically we need a function to evaluate the goodness of partitioning of a graph into clusters. A well accepted criterion is *modularity* which has the unique privilege of being at the same time a global criterion to define a community, a quality function and the key ingredient of the most popular method of graph clustering. This criterion which is introduced by Newman and Girvan [12] formally defined as follows:

$$Q = \sum_i e_{ii} - a_i^2 \tag{1}$$

where e_{ii} is the fraction of edges that connects two nodes inside the community i and a_i represent the fraction of edges that connect two vertices in community i(i.e. having one or both vertices inside the community i). The sum extends to all communities i in a given network. The larger the Q is, the corresponding partition would be more accurate.

In the other words, e_{ii} is the real fraction of edges within a community i. With disregarding the underlying structure, the expected value of the fraction of links within a community can be estimated. a_i^2 is simply the probability that an edge begins at a vertex in community i, multiplied by the fraction of edges that end at a vertex in community i. So, the expected number of intra-community edges is just $a_i a_i$. We can compute these two values directly and sum over all the communities in the graph [15].

We define E_i as the number of the internal edges of the community i and also e as the number of the edges of the graph, so we have:

$$e_{ii} = \frac{E_i}{e} \tag{2}$$

The term a_i can be thought as the sum of degrees of vertices belonging to the community c_i divided by the sum of the degrees of all vertices of the graph. So considering d_v as the number of the neighbors of vertex v (or degree of vertex v) in graph G, we could write:

$$a_i = \frac{\sum_{v \in c_i} d_v}{\sum_{v \in G} d_v} \tag{3}$$

Obviously in this formula and for undirected graphs, internal edges are counted twice, and if we denote the number of all external edges connecting to community c_i as Ext_i then a_i can be written as follows:

$$a_i = \frac{2E_i + Ext_i}{2e} \tag{4}$$

Now let's consider a graph with a set of some subcommunities including two subcommunities c_i and c_j and the same graph with the same set of subcommunities except with subcommunity c_m which is formed by merging c_i and c_j. Also assume that Q_1 is the modularity value for the former case and Q_2 is the modularity value for the latter one. We are going to compute the difference between these two modularity values (ΔQ) in order to define which pair of communities, if we merge, would result in increasing the modularity value. Suppose that for community c_r we have: $Q_{c_r} = e_{rr} - a_r^2$. Hence, $Q = \sum_r Q_{c_r}$. Due to the fact that the number of internal edges and also external edges for the rest subcommunities is unchanged, we could write:

$$\Delta Q = Q_{c_m} - (Q_{c_i} + Q_{c_j}) \tag{5}$$

and in more details:

$$\Delta Q = e_{mm} - a_m^2 - (e_{ii} - a_i^2 + e_{jj} - a_j^2) \tag{6}$$

It is obvious that the number of internal edges of the resulted community c_m from merging two communities c_i and c_j is:

$$E_m = E_i + E_j + E_{ij} \tag{7}$$

where E_{ij} is the number of the edges between two communities c_i and c_j. So, using equation 2 we have:

$$e_{mm} = e_{ii} + e_{jj} + \frac{E_{ij}}{e} \tag{8}$$

If we rewrite Ext_i and Ext_j according to E_{ij}, we have:

$$Ext_i = E_{ij} + \alpha, Ext_j = E_{ij} + \beta \tag{9}$$

where α and β are the number of external edges of communities c_i and c_j respectively which are the external edges of c_m as well. Thus

$$Ext_m = \alpha + \beta \tag{10}$$

and using equations 4, 7 and 10 and 9 , we could derive:

$$a_m = \frac{2E_i + 2E_j + Ext_i + Ext_j}{2e} \tag{11}$$

or

$$a_m = a_i + a_j. \tag{12}$$

Finally, using equations 6, 8 and 12 we have:

$$\Delta Q = e_{ii} + e_{jj} + \frac{E_{ij}}{e} - (a_i + a_j)^2 - (e_{ii} - a_i^2 + e_{jj} - a_j^2) \tag{13}$$

which could be summarized to:

$$\Delta Q = \frac{E_{ij}}{e} - 2a_i a_j \tag{14}$$

and using equation 4,

$$\Delta Q = \frac{E_{ij}}{e} - 2\frac{2E_i + Ext_i}{2e}\frac{2E_j + Ext_j}{2e}. \tag{15}$$

Thus ΔQ has been rewritten according to the terms of two communities c_i and c_j, i.e E_i, E_j, E_{ij}, Ext_i and Ext_j.

Equation 14 is a very interesting result. It means that after merging two communities c_i and c_i, ΔQ will be the real fraction of the edges connecting two communities minus the expected fraction of edges between these two communities.

Back to the modularity maximizing algorithm for finding communities and using equation 15, we want to know which two communities must be merged in order to have a maximum increase in modularity value. Suppose that we start our algorithm with considering all single vertices as a community. As we know, a community c_i with only one vertex, have not any internal edge which means E_i =0. Besides, for a simple graph two vertex can be connected with only one edge for two communities c_i and c_j which means E_{ij} =1. Moreover, for a single-member community c_i having only one vertex i, the number of external edges for this community(i.e. Ext_i) equals with the degree of the vertex i. Using d_i to denote the degree of the vertex i, ΔQ in equation 15 for this situation can be written as

$$\Delta Q = \frac{1}{e} - 2\frac{d_i}{2e}\frac{d_j}{2e} \tag{16}$$

Therefore, starting with single vertices as subcommunities, the lower the degree of two neighbor vertices, the higher their corresponding ΔQ after merging. Thus this approach will cluster neighbor vertices with low degree into the same community, while they may not exist strong relationship between them. In Section V we will show that this will result in poor community detection. It is a good idea to cluster neighbor vertices with strong relationship into the small communities (i.e. preliminary communities) at first. Then these preliminary communities will be used as starting points for the modularity maximization approach. In the following section, we describe a similarity measure between two vertices which is used to find the strength of relationship between two vertices. The vertices with higher degree of similarity make the preliminary communities.

B. Similarity Measure

It is a very normal assumption that communities are groups of vertices similar to each other. One of the most common vertex similarity measures is called structural similarity in which, vertex similarity is measured only based on the structure of a network. Two vertices are structurally similar if they share some common neighbors, even if they are not adjacent themselves. Vertices with large degrees and different neighbors are considered very far from each other.

In a very naive way the number of the common friends of vertices i and j could be considered as the measure of structural similarity

$$S_{naive}(i,j) = |N_i \cap N_j| \tag{17}$$

where N_i and N_j are the sets of the neighbors of vertices i and j respectively. Alternatively, one sophisticated and normalized vertex similarity which is called *cosine* similarity is defined as the following:

$$S_{cosine}(i,j) = \frac{|N_i \cap N_j|}{\sqrt{|N_i||N_j|}} \tag{18}$$

C. Weighting

We are going to find cosine weight for all edges as the similarity measure between its vertices. After that, we cluster each vertex v and one of its neighbor vertices having maximum cosine weight into the same community. As a result, we will have some subcommunities with clustered vertices based on their similarity values to be used later in the our modularity maximization algorithm.

```
1  foreach vertex $v$ do
2      foreach neighbor $u$ of $v$ do
3          Cfriends=0;
4          foreach neighbor $z$ of $u$ do
5              if $z$ and $v$ are neighbor then
6                  ++Cfriends;
7              end
8          end
9          /* compute cosine weight for edge(u,v) */
10     end
11 end
```

Algorithm 1: Simple Algorithm (for edge weighting)

To find the number of common friends for all pairs of vertices of the edges, there is a simple algorithm (Algorithm 1). In this algorithm, for every vertex v and for every neighbor vertex u of v (i.e for each edge(u, v)), first, the number of common friends between vertices u and v is counted and then cosine weight is assigned to that edge. Using the adjacency matrix data structure, checking neighboring of vertices z and v (line 5) can be done in $O(1)$ time and thus finding the number of common friends between vertices u and v of an edge(u, v) can be done in $O(d_u)$ time where d_u is the number of neighbors of vertex u(line 3-8). After that, we can compute cosine weight of edge(u, v). Because each vertex u has d_u neighbor edges, finding the weight of all neighbor edges of vertex u, has the time complexity of $O(d_u^2)$.

Therefore, time complexity of this simple method to find the number of common friends for all pairs of vertices of the edges of the graph will be: $\sum_{v=1}^{n} d_v^2$. We can find different examples of graphs, even sparse graphs, that this time complexity can simply reach $O(n^2)$. For instance in a case of having some vertices with high degrees such as n/k (a fraction of n), the time complexity will be $O(n^2)$. The space complexity for this algorithm, will be $O(n^2)$ too. Since checking neighbors of vertices z and v (line 5) costs d_v time, using adjacency list increases the total time complexity of finding common friends. We show that we can change this simple method to have a lower upper bound for the total time complexity, specially for sparse graphs and at the same time we can have the advantage of using adjacency list with space complexity $O(m)$.

Again, To obtain the number of common friends between v and one of its neighbor vertices (i.e. u), we must just count the neighbors of v that are neighbor to u as well. After that, we compute weight of edge (u, v) and mark it as "weighted". We call this process extension, and we say that vertex v has been "extended". If we constrain ourselves to not extend each vertex more than once and to find only the weight of an unweighted edge with each extension, we will have these results: First, the time complexity for extension in the worst case would be $\sum_{v=1}^{n} d_v = O(m)$, when each vertex is extended exactly once. Second the maximum number of weighted edges will be n and that is because the number of weighted edges equals with the number of extensions. This approach is presented in Algorithm 2 which is called extension algorithm. Extension algorithm is the modified version of simple algorithm (Algorithm 1) with considering two mentioned constraints as two conditions in line 3 of algorithm 2 .

```
1  foreach vertex $v$ do
2      foreach neighbor $u$ of $v$ do
3          if $u$ is "unextended" And edge$(u, v)$ is
           "unweighted" then
4              Cfriends=0;
5              foreach neighbor $z$ of $u$ do
6                  if $z$ and $v$ are neighbor then
7                      ++Cfriends;
8                  end
9              end
10             /* compute cosine weight for edge(u,v) */
11             /* mark vertex $u$ as "extended" */
12             /* mark edge $(u, v)$ as "weighted" */
13         end
14     end
15 end
```

Algorithm 2: Extension Algorithm

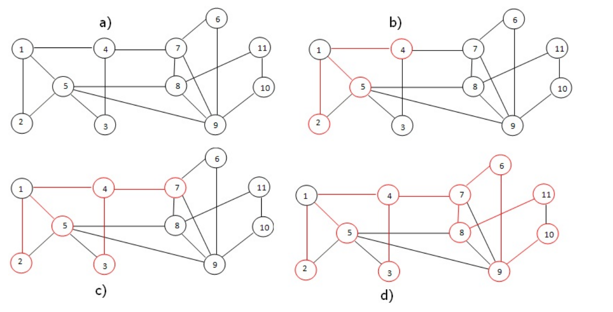

Figure 1. (a) shows a sample graph with 11 vertices (n=11). (b) and (c) are an attempt to find the weights of all neighbor edges of vertices 1 and 4 respectively. First run of extension algorithm has been illustrated in (d).

So, if each vertex is extended once, the corresponding time complexity will be $\sum_{u=1}^{n} d_u = O(m)$. Therefore, time complexity of extension algorithm is

$$O(m) \tag{19}$$

To clarify how extension algorithm works, we run this algorithm on a sample graph which is presented in Figure 1.a. At first run, all edges are considered as "unweighted" and all vertices are considered as "unextended". If we consider natural order of vertices, the first vertex of the graph will be vertex 1. So, we make an attempt to find the weight of all of its neighbor edges. Since all of its neighbor vertices are "unextended" and all of its neighbor edges are "unweighted", all of its neighbor vertices become "extended" and therefore we can find the weight of all of its neighbor edges. The result has been illustrated in Figure 1.b. The weighted edges have been specified with red lines and the extended vertices have been determined with red circles. Second vertex is vertex 2. One of neighbor vertex of vertex 2 is vertex 5 which has already been extended . So, vertex 5 can not be extended and the weight of edge $(2,5)$ can not be found. The other neighbor vertex of vertex 2 is vertex 1. As we already have the weight of edge $(1,2)$, there is no need to extend vertex 1. Third vertex is vertex 3. Since all of its neighbor vertices have been extended before, we do nothing. Fourth vertex is vertex 4. There are three neighbor vertices of vertex 4: vertices 1, 3 and 7. We already have the weight of edge$(4,1)$, So there is no need to extend vertex 1. Since vertices 3 and 7 are "unextended" and edges $(4,3)$ and $(4,7)$ are "unweighted", both of them will be extended and will be marked as "weighted" (Figure 1.c). We continue this approach for the rest of the vertices (See Figure 1.d).

As it has been mentioned, in extension algorithm, the number of weighted edges will not be greater than n. Thus, this algorithm must be repeated several times in order to find the weight of the rest of unweighted edges (See Algorithm 3).

```
Consider all edges as "unweighted";
k=1;
while $k <= R$ do
    Consider all vertices as "unextended" ;
    Extension Algorithm();
    ++k;
end
```

Algorithm 3: Weighting Algorithm

Since weighting algorithm (Algorithm 3) has R iterations and each iteration has time complexity $O(m)$ (cost of extension algorithm), total time complexity of the proposed weighting algorithm will be

$$O(R.m) \tag{20}$$

An important question that needs to be answered is how many iterations our proposed weighting algorithm needs to find the weights of all edges of the graph? In other words, what would be the best value for R?

Suppose that D is the average degree of the graph. If all vertices of the graph have equal degree D we need maximum D iterations to find for each vertex, the weight of all of its neighbor edges. Consequently, in this situation, we need D iterations to find the weight of all edges of the graph. In reality, the degree of each vertex of the graph in average, not exactly, equals to D. So we expect that in average we find the weights of all edges in maximum $\lceil D \rceil$ iterations. As a result $\lceil D \rceil$ would be a good fixed value for R. If we set R to D the weighting algorithm will be an approximate algorithm. the results showed that the rate of weighted edges found by this weighting algorithm is generally high (see Table I). It is of importance to note that the goal of the proposed weighting algorithm, as we will see later, will not be finding the weight of all edge of the graph. As the weighting algorithm will only be used in preliminary community detection stage and in real graphs which are sparse, D is very small and sometimes even lower than $log(n)$, so we can set R to $log(n)$. In this situation, for each vertex the algorithm can find the weight of at least $log(n)$ of its neighbor edges. However, as the rate of weighted edges gets higher, we would have more accurate preliminaries and as the result, the output of the community detection algorithm will have better modularity value.

Network	n	%weighted
Karate	34	100%
Jazz	198	99.9%
Metabolic	453	99.6 %
Email	1133	99.9%
Key Signing	10680	82.1%
Physicists	27519	95.9 %

Table I
THE RESULT OF WEIGHTING ALGORITHM, IF R=$\lceil D \rceil$.

Next step is reducing space complexity of the algorithm using adjacency list and without increasing the time complexity. Although we could have better space complexity $O(m)$ using adjacency list, checking neighboring vertices z (vertex z is neighbor vertex of u) and v (line 6 of extension algorithm) could not be possible with $O(1)$ time and this increases the total time complexity again. To avoid that, we use labeling technique. That is, in each iteration and for every vertex v, we assign all of its neighbor vertex u, a label v. Then, to find the number of common friends between v and each neighbor u of v, it is just enough to count all u's neighbors which have label v.(See Algorithm 4)

```
1  foreach vertex $v$ do
2      foreach neighbor $u$ of $v$ do
3          assign label $v$ to $u$
4      end
5      foreach neighbor $u$ of $v$ do
6          if $u$ is "unextended" And $edge(u,v)$ is
           "unweighted" then
7              Cfriends=0;
8              foreach neighbor $z$ of $u$ do
9                  if $z$ has label $v$ then
10                     ++Cfriends;
11                 end
12             end
13             /* compute cosine weight for edge(u,v) */
14             /* mark vertex $u$ as "extended" */
15             /* mark edge $(u,v)$ as "weighted" */
16         end
17     end
18 end
```

Algorithm 4: Revised Extension Algorithm. We used labeling technique in order to reduce the space complexity without increasing the time complexity.

Using labeling technique, checking neighboring vertices z and v can be done in $O(1)$ time, even with adjacency list. Thus, to know whether vertices z and v are neighbors or not, it is enough to check if the label of vertex z equals to v or not (line 9). The cost of extension of the vertices in revised extension algorithm is the same as extension algorithm (i.e. $O(m)$). But we have labeling cost. A label will be assigned to every neighbor vertices of each vertex u in d_u times. So, the time complexity of labeling will be $\sum_{u=1}^{n} d_u = O(m)$. Thus, the total time complexity of revised extension algorithm is $O(2m)$ or

$$O(m) \tag{21}$$

We need to set all vertices as "unlabeled" before revised extension algorithm runs. Revised weighting algorithm (Algorith 5) is our final weighting algorithm with time complexity $O(R.m)$.

```
1 mark all edges as "unweighted";
2 k=1;
3 while $k <= R$ do
4     mark all vertices as "unextended" ;
5     mark all vertices as "unlabeled" ;
6     Revised Extension Algorithm();
7     ++k;
8 end
```

Algorithm 5: Revised Weighting Algorithm.

D. Preliminary Community Detection

After weighting edges, we are going to find preliminary communities (i.e. subcommunities). We first put all weighted edges of the graph in an array and then sort them based on their weight in descending order. At first, all vertices are considered unassigned. Then the edges in the array are picked one by one and for each edge an attempt is made to assign its vertices to a new community, if both of them were unassigned. That means, if we have an edge (u, v) that both of u and v have not already been assigned to a community, we create a new community and assign both vertices u and v to it . If at least one of them were already assigned to a community,we do nothing. Thus we will have some subcommunities which have only two vertices. We consider the rest of the unassigned vertices as single member subcommunities. As a result and after running the preliminary community detection algorithm, graph will be divided into small communities which have only one or two vertices. In the two-members communities, vertices are assigned according to their similarity and we hope that they can be used as a more appropriate and reasonable preliminary subcommunities in our proposed modularity maximization algorithm. As an example the result of this stage on Karate graph is illustrated in Figure 3.

The space needed for this stage is an array with the size of the number of the edges of the graph(i.e. m). Required space for preliminary community detection is $O(m)$.

Running time of this stage is mainly concerned with the sorting of the weighted edges. Time complexity for sorting an array is $O(nlogn)$ using merge sort. The maximum number of weighted edges is m and therefore, running time of this stage is: $O(mlogm)$.

E. Merging Communities

Having found preliminary communities, we are going to merge communities with the objective of having better modularity value. First, for each community c_i, we find a neighbor community c_j which merging them will result in maximum ΔQ. If this maximum ΔQ is positive then we draw an arrow from community c_i to community c_j.

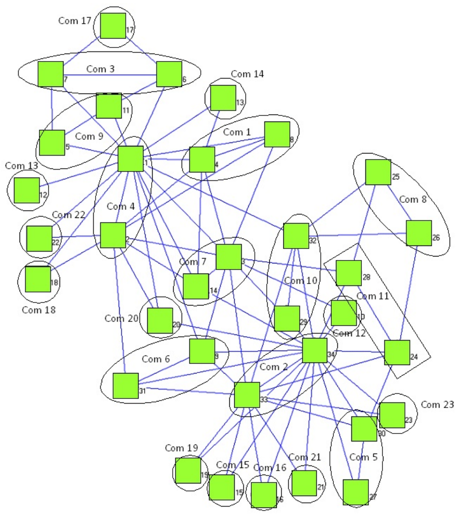

Figure 2. Result of preliminary community detection algorithm on Karate Network.

If each pair of these connected communities are merged together, we will have an increase in the modularity value. It is obvious that if it exists a bi-directional arrow between two communities c_i and c_j, merging this pair of communities will lead to more increase in modularity value in comparison with merging one of them with another single neighbor community. This kind of merging is referred to as "pairwise" merging and will definitely increase modularity value though has lower speed. It must be used with a more quick merging which we call it "single neighbors" merging . In this type of merging, all communities which are connected with only one community(with an arrow), will be merged to that corresponding community. Although it has more speed in merging, this approach will not guarantee to increase the modularity value. Consequently, we use both of them and call this approach "Hybrid" merging. In "Hybrid" merging, we start with "pairwise" merging for some iterations. Having had R_m as the number of iterations for merging, we choose a fraction of it(i.e. $Frac$) to start with pairwise merging. The rest iterations(i.e. $(1 - Frac)R_m$) will be devoted to "single neighbors" merging. We will see that a reasonable value for R_m is $log(n)$. $Frac$ can be simply set to 0.5. The experiments will show that more increase in R_m and $Frac$ will result in better modularity values.

An important question to answer is, how far the number of iterations go if the merging communities algorithm needs to find all the communities as they are. Assume that we assign each vertex to a community; so we have n communities with size one. We also suppose that in each iteration of merging methods we can merge only two communities together. With the goal of having only one resulted community that consists of all vertices of the graph, we will need log(*n*) iterations to merge all communities to form one communities including all vertices. Hance, a reasonable value for number of iterations of merging(i.e. R_m) is log(n). At the end of each iteration, we compute modularity and at the end of program, we select the maximum modularity.

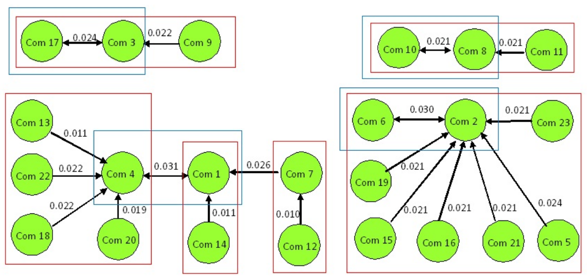

Figure 3. Scheme of merging subcommunities on subcommunities obtained by preliminary community detection algorithm on the Karate graph. the number above arrows refers to the value of ΔQ if the corresponding two communities are merged together. Communities which are specified as red squares are the result of "single neighbors" merging and those communities which are specified with blue squares are the result of "pairwise" merging

In merging process we have $\log(n)$ iterations and each iteration can be implemented with time complexity $O(m)$. Therefore, time complexity of merging stage would be $O(m.logn)$.

IV. Complexity Analysis

As we know $D = \frac{\sum_{i=1}^{n} d_i}{n}$, so $D = \frac{2m}{n}$, and $m = n\frac{D}{2}$. Our proposed algorithm has three parts: weighting algorithm with time complexity $O(R.m)$ and space complexity $O(m)$, preliminary community detection with time complexity $O(m.log(m))$ and space complexity $O(m)$ and finally merging stage with time complexity $O(m.log(n))$. The total time complexity of the proposed community detection is:

$$R.D.n + D.n.log(n) + log(n).D.n \tag{22}$$

and the total space complexity of the purposed algorithm is

$$O(2m)) \tag{23}$$

In the weighting algorithm, we set the number of iterations $R = \lceil log(n) \rceil$. So, the time complexity of the the proposed community detection algorithm will be $O(D.n.log(n))$. For real graphs which are sparse ($D \ll n$), this time complexity will be

$$O(n.log(n)) \tag{24}$$

The space time complexity of the proposed algorithm for sparse graph is also

$$O(n) \tag{25}$$

If we want to extract members of the cluster of the partition with highest modularity, we must allocate an array with size n for each iteration whose elements specify the communities of the vertices. In this situation, space complexity will be: $O(n.log(n))$.

Network	n	GN	CNM	EO	N_{eig}	this paper
Karate	34	.401	.381	.419	.419	.420
Jazz	198	.405	.439	.445	.442	.425
Metabolic	453	.403	.402	.434	.435	.435
Email	1133	.532	.494	.574	.572	.560
Key Signing	10680	.816	.733	.846	.855	.876
Physicists	27519	-	.668	.679	.723	.743

Table II
COMPARISON OF ALGORITHMS FOR THE FIRST GROUP OF NETWORKS

Network	n	$\lceil log(n) \rceil$	$\lceil D \rceil$	FKcd	LM	this paper
CA-GrQc	5,242	13	12	.786	.816	.860
CA-HepTh	9,877	14	12	.648	.768	.762
CA-HepPh	12,008	14	40	.598	.659	.608
CA-AstroPh	18,772	15	44	.568	.628	.607
CA-CondMat	23,731	15	18	.599	.731	.719

Table III
COMPARISON OF ALGORITHMS FOR THE GROUP OF COLLABORATION NETWORKS

V. Experimental Result

We evaluated our method on two different groups of networks: A group of six benchmark networks of varying sizes [18] and a group of five collaboration networks [19].

As it is presented in table II our method mainly outperforms other well known methods which are reviewed in Section II. For instance; for Karate, Key Signing and Physicists networks, the proposed algorithm outperforms other methods. In Metabolic network, our strategy and N_{eig} method obtained maximum modularity value 0.435. It is of value to note that time complexity of N_{eig} and EO are $O(n^2 logn)$ and $O(n^2 log^2 n)$, while our proposed algorithm has time complexity $O(n.log(n))$.

In table III, the comparison between the proposed algorithm, Louvain methods (LM)[3] and generalized Louvain method(i.e. FKcd) [17] is presented. Although our algorithm has better modularity value for the first network(i.e. CA-GrQc), for the rest of the networks LM has better modularity value. However the modularity values obtained by two methods specially for CA-HepTh and CondMat networks are very close. It is worthy to note that in a more sparse network specially with $log(n)$ greater than the average degree of the graph, our proposed algorithm has better performance.

We conducted another experiment to see the impact of choosing different values for the parameters of our algorithm. As it is stated before, R is the number of iterations in weighting algorithm, R_m is the number of iterations for merging communities and $Frac$ is the fraction of R_m that we carry out pairwise merging. Using similarity measure for detecting preliminary communities, we evaluated our algorithm with three different settings for $(R, R_m, Frac)$. These settings are: $(log(n), log(n), .5)$, $(log(n), 2log(n), .75)$ and $(4log(n), 4log(n), .875)$ which are presented in Table IV and in A, B and C column respectively. We also run our algorithm not using similarity measure (i.e. do not using weighting algorithm). The setting for $(R_m, Frac)$ is $(4log(n), .875)$ which is

Network	n	A	B	C	D	$W_{A,B}$
Karate	34	.420	.420	.420	.383	100%
Jazz	198	.409	.428	.425	.422	52%
Metabolic	453	.410	.425	.435	.426	100%
Email	1,133	.542	.558	.560	.547	100%
Key Signing	10,680	.868	.870	.876	.863	96%
Physicists	27,519	.736	.738	.743	.733	100%
CA-GrQc	5,242	.841	.848	.860	.851	91%
CA-HepTh	9,877	.750	.755	.762	.752	100%
CA-HepPh	12,008	.578	.574	.608	.612	53%
CA-AstroPh	18,772	.590	.596	.607	.582	75%
CA-CondMat	23,731	.710	.716	.719	.715	91%

Table IV
IMPACT OF $(R, R_m, Frac)$ IN MODULARITY VALUES IN DIFFERENT NETWORKS

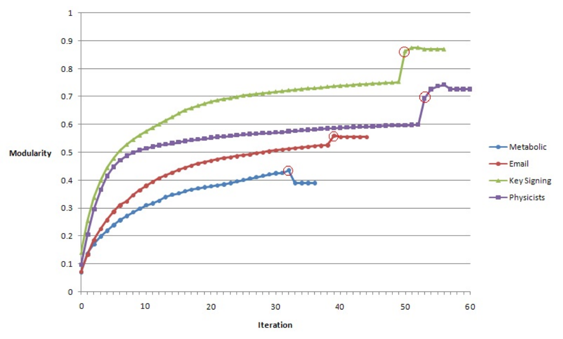

Figure 4. Modularity value in each iteration for four benchmark graphs. Points specified with the red circles are the start point of "single neighbors" merging. That is, $(\lfloor Frac.R_m \rfloor + 1)$th iteration . Before this point, "pairwise" merging is carried out.

presented in column D. The percentages of weighted edges in states A or B (which are equal) is denoted as $W_{A,B}$. The edges are almost fully weighted for all networks for state C.

As it is clearly observable from Table IV by increasing the value of $(R, R_m, Frac)$, the obtained modularity value will generally increase. Comparing states C and D shows that using similarity value for finding preliminary communities (column C), except for one network (i.e. CA-HepPh), will result in more modularity value. The stages of merging communities for four benchmark graphs with settings (R, R_m, $Frac$)=($4log(n)$, $4log(n)$, 0.875) is depicted as figure 4. The sudden change in modularity value is due to change from pairwise merging to single neighbor merging method.

VI. CONCLUSION

We proposed a modularity maximization algorithm for community detection with time complexity $O(n.log(n))$. The algorithm utilized a vertex similarity measure to find small preliminary communities to be used as start configuration in merging stage. As we compared our algorithm with some of well-known algorithms on several benchmark graphs, our algorithm had better performance. For some networks, while the proposed algorithm has lower time complexity, the performance was comparable with other algorithms in terms of quality of discovered communities modularity value. Its low cost and good accuracy enables this method to be applied on possibly very large networks.

REFERENCES

[1] M. Girvan and M. E. Newman, Proc. Natl. Acad. Sci. USA 99, 7821 (2002).

[2] A. Clauset, M. E. Newman, and C. Moore, Phys. Rev. E 70, 066111 (2004).

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech. P10008 (2008).

[4] S. van Dongen, Ph.D. thesis, Dutch National Research Institute for Mathematics and Computer Science, University of Utrecht, Netherlands (2000).

[5] L. Donetti and M. A. Munoz, J. Stat. Mech. P10012 (2004)

[6] M. E. J. Newman and E. A. Leicht, Mixture models and exploratory analysis in networks, Proc. Natl. Acad. Sci. USA 104, 9564-9569 (2007).

[7] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435, 814 (2005).

[8] S.Fortunato, Community detection in graphs, Physics Reports, 486, 75-174 (2010).

[9] M. E. J. Newman, Fast algorithm for detecting community structure in networks, Phys. Rev. E 69, 066133 (2004).

[10] M. E. J. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. USA 103, 8577-8582 (2006).

[11] J. Duch and A. Arenas,Community identification using Extremal Optimization, Physical Review E, vol. 72, 027104, (2005)

[12] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69, 026113 (2004).

[13] M. E. J. Newman, Detecting community structure in networks, Eur. Phys. J. B 38, 321-330 (2004).

[14] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, Self-contained algorithms to detect communities in networks, Proc. Natl. Acad. Sci. USA 101, 2658 (2004).

[15] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas. Comparing community structure identification, J. Stat. Mech. P09008 2005.

[16] M. Rosvall and C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proc. Natl. Acad. Sci. USA 104, 7327 (2007).

[17] P. De Meo, E. Ferrara, G. Fiumara, A. Provetti, Generalized Louvain Method for Community Detection in Large Networks. Proc. of the 11th International Conference On Intelligent Systems Design And Applications, pp. 88-93, 2011.

[18] http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml

[19] http://snap.stanford.edu/data/index.html