Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 2190 – 2205
© TÜBİTAK
doi:10.55730/1300-0632.3933

A new network-based community detection algorithm for disjoint communities

Pelin ÇETİN[*], **Şahin EMRAH AMRAHOV**
Department of Computer Engineering , Ankara University, Ankara, Turkey

Received: 09.09.2021 • **Accepted/Published Online:** 28.07.2022 • **Final Version:** 28.09.2022

Abstract: A community is a group of people that shares something in common. The definition of the community can be generalized as things that have common properties. By using this definition, community detection can be used to solve different problems in various areas. In this study, we propose a new network-based community detection algorithm that can work on different types of datasets. The proposed algorithm works on unweighted graphs and determines the weight by using cosine similarity. We apply a bottom-up approach and find the disjoint communities. First, we accept each node as an independent community. Then, the merging process is applied by using the modularity value as a stopping criterion. We use real datasets and evaluate the algorithm with modularity, normalized mutual information, and performance metrics. In addition, we test our algorithm by using central nodes. We also take into consideration the number of communities in the case they are known. The proposed algorithm has high modularity and accuracy in different datasets.

Key words: Community detection, disjoint communities, cosine similarity, modularity, network-based

1. Introduction

Social networks are the most well-known application area of community detection. The number of social network users is increasing every day. Growth in social networks makes it harder to manage networks, cope with the increasing data, get access to reliable and accurate information. Today, social network users have a thousand friends, so manually grouping them is costly time. At this point, community detection can be used to determine the hierarchy of complex structures, modularize the system and manage the network easily.

An organized social network that is divided into subgroups provides advantages in content filtering, data privacy, and data sharing. For example, it is possible to block messages from certain groups by content filtering. Similarly, data privacy can be guaranteed by dividing the network into smaller groups and allowing some messages or information to be shared with the related group. Furthermore, the organization of social networks is a difficult problem because the network is large, and the organization requires effort in updating each new data. For these reasons, efforts are being made to automate the organizing of social networks [6, 22, 32].

In addition, community detection can be used in various areas such as identifying similar proteins for medicines, finding similar network users for the Internet service, detecting users with similar interests for recommendation systems, keeping data more efficiently, searching routes in large networks, etc. [20, 23]. It is important for drug production, determining the communication paths in ad-hoc networks, identifying related websites in web page search engines, querying the data, detecting anomalies, minimizing the communication while distributing tasks to processors in a parallel calculation, and many other areas.

[*]Correspondence: pelincetin.cs@gmail.com

The main problem in community detection is the definition of the community [10]. The definition of the community may vary based on the domain in which the problem is considered [12]. Therefore, there are various solutions in the literature. We examine these solutions under two headings: content-based and network-based methods. Content-based methods depend on the principle that members of a community have similar profiles. Although this method gives successful results, it is specific to the problem. Also, finding the right communities can be troublesome since it is hard to find similar people in the case of insufficient profile data. The network-based methods use graph structure. In a graph $G = (V, E)$, V represents the members, and E represents the existing connection between two members. Since network-based methods use graph data such as the degree of nodes, the weight of edges, centrality, etc., the calculation time generally depends on the size of the network, and it can be costly. Some studies use the advantages of both methods.

In this study, we propose a new algorithm to detect disjoint communities in complex networks. These complex networks can be from different fields such as biology, social, technology, information science, and so on. The input of our algorithm is an undirected and unweighted graph, and the outputs are the disjoint communities. We were inspired by label propagation and Louvain algorithms. However, the proposed algorithm is not a hybrid of these two algorithms. We use a bottom-up approach as in these algorithms. Important differences between our algorithm and the label propagation algorithm are converting the input graph to a weighted graph and performing the merging process in order of weight. Moreover, our algorithm is deterministic, unlike the label propagation algorithm. The similarity of the proposed algorithm to the Louvain algorithm is that both algorithms try to increase modularity while merging. The Louvain algorithm first finds the communities by increasing modularity. Then it creates a new graph and renews the weights based on the connections inside and between the founded communities. The algorithm repeats these two steps as long as there exist communities that modularity increases when we merge them. Our algorithm first determines the weights according to the similarity of the neighbor nodes. Then, it looks at the edges in descending order of the weights. After that, it merges the communities of the ends of the edges if the modularity increases. In addition, we perform these operations on only one graph, which uses less memory and reduces the number of operations performed, unlike the Louvain algorithm. We compare our algorithm with known community detection algorithms by applying it to benchmark datasets. We measure the success of algorithms by various metrics such as modularity, performance, and normalized mutual information. We emphasize that the proposed algorithm does not take the number of communities as an input, unlike many algorithms. For this reason, we also analyze the results of our algorithm on depending the number of communities.

The rest of the paper is organized as follows: In the next section, we summarize the related works. In Section (3), we give information about similarity metrics, evaluating metrics, and algorithms used in this study. In Section (4), we explain the proposed algorithm. In (5), we test the proposed algorithm on benchmark datasets. We compare it with other algorithms. Finally, in Section (6), we discuss our results and make the concluding remarks.

2. Related work

In this study, we focus on disjoint community detection on unweighted graphs. We review the network-based disjoint community detection algorithms in the literature. In these algorithms, the communities are detected by using node or edge properties, and there are two approaches: bottom-up and top-down.

The top-down approaches generally use edge properties such as bridges, weak-edges, edge-betweenness, etc. They start by accepting all graphs as one community and divide them by removing some edges using these properties. On the other hand, the bottom-up approaches generally use node properties and start by accepting each node as an independent community. Then the nodes are merged by using some properties such as degree centrality, closeness centrality, eigenvector centrality, etc. In both methods, there must be some stopping criteria if the number of communities is not known.

Girvan and Newman's algorithm [14] is one of the well-known algorithms in the literature. They applied a link-based method and defined the edge-betweenness metric by generalizing Freeman's betweenness centrality [13] to edges. They find communities by repeatedly removing the edge with the highest edge-betweenness score until none of the edges remains. Therefore, this algorithm has a high computational complexity. However, it generates a hierarchical tree that helps us see the whole structure in the network. Fang-ju [11] also uses the similarity between nodes. However, they assume that a community must have at least four nodes, which is a very restricted condition. In another study, Arasteh and Alizadeh [2, 3] propose a method that allows removing multiple edges at the same time to improve the running time of Girvan and Newman's algorithm. In this study, we also transform the graph into a weighted graph as in Girvan and Newman's algorithm. Unlike them, we do this by using the similarity between the nodes. We also add each node to its adjacent list, which increases the success of our algorithm.

After Newman [26] defines the modularity for the network to measure the quality of the division, modularity-based algorithms are proposed. One of them is the Louvain algorithm that is proposed by Blondel et al. [4]. Since it uses modularity, the division of the algorithm is better. Then, Mohammadi et al. [24] propose an adaptive CUDA Louvain method algorithm for acceleration. The Louvain algorithm evaluates the neighbors of a node and decides to merge by using modularity. Then, it generates a new network and continues iteratively until the modularity doesn't increase. This algorithm has a memory problem due to the step of generating new networks. The similarity of the algorithm we propose in this study to the Louvain algorithm is that both algorithms are designed to increase modularity. However, unlike the Louvain algorithm, we only do this for one graph. It allows us to achieve similar results with less processing time and memory.

Modularity is also used in metaheuristic optimization-based solutions. Hosseini and Rezvanian [15] use an ant colony optimization algorithm for modularity maximization with label propagation. Sani et al. [34] use a multiobjective ant colony optimization algorithm to optimize the density of the clusters and the community fitness. Then they select the solution with the highest modularity. Imtiaz et al. also use multilayer ant colony optimization [18]. Zhang et al. [38] propose whale optimization-based algorithm. Abduljabbar et al. [1] define the detection of the community as the optimization of both neighborhood relations and signature similarity. They use degree vectors of subgraphs as a signature. Then, they propose a new heuristic mutation operator to optimize neighborhood relations in terms of inter and intracommunities. The main problem of optimization-based solutions that use modularity is the resolution limit of modularity. Because of that, they can miss some subcommunities. Unlike these algorithms, the proposed algorithm is not a metaheuristic algorithm. As it is known, the biggest problem of metaheuristic algorithms is artificially defining the stopping criterion. Usually, the algorithm stops when the number of hops is greater than a threshold value. This situation makes it harder to divide the network into real communities. In addition, since metaheuristic algorithms have randomness, they can produce different communities every time they are applied. On the other hand, since the algorithm we propose is deterministic, it always produces the same communities.

One of the most used node-based algorithms in the literature is the label propagation algorithm that is proposed by Raghavan et al. [30]. There are three types of label propagation algorithms: synchronous, asynchronous, and semisynchronous. In synchronous methods, since the label of the previous step is used in each step, updating them can cause an endless loop problem. On the other hand, asynchronous methods can prevent the loop, but they perform poorly, and labeling cannot be completed unless the labeling of all their neighbors is completed. In order to overcome this problem, semisynchronous algorithms are proposed. One of them belongs to Cordasco and Gargano [8]. They first apply a coloring and then a labeling procedure. By prioritizing the existing label, they prevent infinite loop. One of the disadvantages of the label propagation algorithm is that it does not work on directional graphs. Another disadvantage is that it is nondeterministic since the nodes are selected randomly. To solve uncertainty and instability in outcomes of label propagation algorithm, Jokar and Mosleh [19] define weights, and the label with the highest weight is used instead of random selection. Yang et al. [35] use similarity to reduce instability. Hu et al. [16] define a measure for node selection according to the role of a node (such as core, peripheral, tight-knit, bridge, etc.) in the community. Zhang et al. [37] define a rule for label updating, label launch, and label acceptance to solve instability problems and increase accuracy. Bouyer and Roghani use a different method, and they find communities by starting from the nodes with the lowest degrees. Then they diffuse the labels to the neighbors of these nodes, and they merge the communities [5]. In the label propagation algorithms, selection of the majority causes to tend to integrate small communities into large communities, and the randomness causes instability. Since we do not look at the label in the majority, our algorithm does not have this disadvantage, and it is deterministic.

Fortuno [12] points out defining communities is only possible if the graph is sparse. He indicates that if the number of edges in the graph is greater than the number of nodes and the distribution of the edges is homogeneous, the identification of the communities is impossible. In this case, communities can be determined by using similarities between the nodes instead of edge density. Huang et al. [17] measure relationships of links by cosine similarity and cluster them by using the fast search clustering algorithm. Then, they transform the link communities into node communities. Okuda et al. [27] use a random walk to find similarities of the nodes. You et al. [36] use Sφrensen similarity and label propagation algorithm. In this study, we propose a new algorithm using the cosine similarity of the nodes. We add each node to the neighbor list before calculating the similarities of the nodes. This increases the success of cosine similarity in our algorithm.

Walk-based algorithms such as Spinglass, Walktrap, or Infomap require additional information for a walk such as spin or number of steps. Otherwise, it has to introduce a parameter and tries to minimize it as in the Infomap algorithm. Changing these parameters affects the results of the algorithms, and the user has to decide the parameter. The starting node for a walk also changes the result of the algorithm. We solve these issues by ordering edges and using modularity as a decision criterion which makes our algorithm independent from a parameter and additional decision process.

Another algorithm that we used in this study is Fluid Communities. It works on connected graphs and requires the number of communities as input. Since it starts from a random node, it is nondeterministic. In our algorithm, the order of the nodes is certain, and it terminates when all node pairs connected by an edge are evaluated. Additionally, there is no randomness in the proposed algorithm. It can work on disconnected graphs and in a situation where the number of communities is unknown.

In this study, our purpose is to develop a disjoint community detection algorithm that is applicable to sparse and dense networks such as protein-protein interactions, social, web pages, etc. that can represent by graphs and gives the best results without requiring any additional information even the weights and the direction of the relations are unknown. For this purpose, we apply a bottom-up approach and propose a new disjoint community detection algorithm. In this algorithm, we only use cosine similarity and modularity metrics. Then, we test our algorithm on real networks and compare the results with some known algorithms. Additionally, we also compare the results of the algorithm when the number of communities is known. The proposed algorithm does not have an instability problem. We believe that the results of this study throw light on community detection and our future works. Our contributions are as follows:

- We propose a new algorithm that is independent of existing algorithms and prior knowledge.
- We compare the similarity metrics and observe their effects.
- The effects of using central communities are investigated.
- The problem of modularity metrics is explained.
- We apply the proposed algorithms to real benchmark datasets. We find exactly the same communities in the karate and dolphins networks.

3. Theoretical background

3.1. Similarity metrics

Similarity metrics are used for collecting the most similar nodes in a cluster. Cosine and Jaccard similarities are the most common similarity metrics. Jaccard similarity is the ratio of the common neighbors between two nodes to all the adjacents of the two nodes. In cosine similarity, nodes are represented by neighbor vectors corresponding to the respective row in the adjacency matrix. The scalar product of the two vectors represents the number of common nodes. The square root of the degree of a node indicates the size of the vector. However, although nodes are connected with a bridge, the cosine similarity of the nodes equals 0. Nodes themselves are added to the list of neighbor nodes to increase this similarity. We also compare the results of the cosine similarity by adding the node itself to the neighbor vector.

Some metrics are based on distance like Euclidean, Manhattan, Minkowski, Gaussian, etc. Since the distance-based metrics use the number of different neighbors of two nodes, and we use these metrics just for sorting the edges, they produce the same outputs. For the simplicity of computation, we use Manhattan distance. Manhattan distance is equal to the number of different neighbors of two nodes. For normalization, we divide each distance by the number of nodes. Then we subtract normalized distances from 1 to find similarities.

3.2. Evaluation metrics

We use performance and modularity, which already exist in a free Python package for study on graphs and networks, NetworkX. Since these metrics do not require ground truth knowledge, they are suitable for unlabeled data. For the labeled data, we used normalized mutual information (NMI).

Modularity, which is developed by Newman and Girvan, is one of the most widely used evaluation criteria. It is based on the idea that a random graph is not expected to contain a cluster structure. In this method, the density of the given graph is compared with the density of the random one. While constructing a random graph, some structural features other than the community structure of the original graph are tried to be preserved. In the Bernoulli random graph, the probability of an edge between node i and j, P_{ij} is defined as given in Equation (1), where n is the total number of nodes in the graph, m is the total number of edges.

$$P_{ij} = \frac{2m}{n(n-1)}, \forall i, j \tag{1}$$

The formula of modularity is given in Equation (2). A is the adjacency matrix and C_i is the community of the node i. $\delta(C_i, C_j)$ is a function which produces 1 if node i and node j belong to the same community, 0 otherwise.

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \tag{2}$$

The total number of edges is maintained in the Bernoulli random diagram, and the edges are placed with the same probability between the nodes. However, this distribution is not consistent with real-life graphs. For this reason, random diagrams with the same degree distribution as the original diagram are preferred. Nodes i and j with the k_i and k_j degree, are connected with probabilities $p_i = k_i/2m$ and $p_j = k_j/2m$. For two independent events p_i and p_j, the probability of an edge being between i and j is equal to $p_i p_j$. Probability function P_{ij} can be defined with $P_{ij} = 2mp_i p_j = k_i k_j/2m$. In this case, the modularity formula becomes Equation (3) [7, 25].

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j) \tag{3}$$

Since addition is performed only for the nodes in the same group, the modularity function can be expressed by Equation (4), where n_c specifies the total number of clusters, l_c represents the total number of edges in c, and d_c represents the total number of degrees in c.

$$Q = \sum_{c=1}^{n_c} \left[\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right] \tag{4}$$

The modularity between 0.3 and 0.7 indicates that clustering is successful [21]. High values are very rare. However, as modularity increases with increasing graph size, it is not suitable for comparing graphs of different sizes.

Performance of our algorithm is also calculated by using the ratio of the number of correctly interpreted node pairs to the number of possible node pairs [12]. The correct interpretation of the nodes means that there is a direct edge between the nodes of the same set, and there is no direct edge for the nodes of different sets.

Normalized mutual information (NMI) is derived from mutual information (MI). MI measures the agreement of the two assignments by ignoring permutations. It is calculated by Equation (5), where u and v are the nodes, and P is the probability function.

$$MI(u, v) = \sum_{(i=1)}^{|u|} \sum_{(j=1)}^{|v|} P(i, j) log \left(\frac{P(i, j)}{P(i)P(j)} \right) \tag{5}$$

NMI is independent of the actual amount of mutual information between the labels Equation (6). H is the entropy. We used NMI with weighted-average remaining maturity.

$$NMI(u, v) = \frac{MI(u, v)}{mean(H(u), H(v))} \tag{6}$$

3.3. Algorithms

In this study, we use the algorithms to find discrete communities in NetworkX and igraph libraries. In Table 1, there is a summary of graph types that these algorithms work on and the ground truth knowledge that is needed.

Table 1. Summary of algorithms.

Algorithm	**Worked on disconnected graph**	**Worked on directed graph**	**Independent of prior knowledge**
Girvan–Newman	✓	✓	Number of communities
Greedy modularity	✓	✓	✓
Walktrap	✓	✓	Number of steps
Louvain	✓	×	✓
Spinglass	×	✓	Number of spins (upper limit for the number of communities)
Label propagation	✓	×	✓
Infomap	✓	✓	✓
Fluid Communities	×	✓	Number of communities
Proposed algorithm	✓	✓	✓

Girvan–Newman applies the centrality betweenness to edges of a graph [14]. In other words, the weights of edges are determined according to the number of shortest paths on this edge. First of all, the loops on nodes are removed so that they will not affect the community structure. Then, the edges with the highest weights are removed from the graph, and clusters are obtained. This process continues until all nodes become a separate community. The number of communities to be found is determined by the user.

Greedy modularity starts from the state where each node is a stand-alone community and progresses by combining the nodes that increase the modularity most [7]. Since merging communities that are not connected will not increase the modularity, only connected communities are merged. Therefore, the edges between the communities i and j are kept in the matrix e_{ij}. Since this matrix is very sparse, data is kept in a max-heap structure to get rid of unnecessary operations. The algorithm calculates the initial values of ΔQ_{ij} and a_i according to Equations (7), (8) and keeps them in the max-heap structure. It updates ΔQ_{ij} and a_i by combining communities with the highest ΔQ_{ij} value. Since both $i-j$ and $j-i$ are evaluated separately in the NetworkX library, fractions in this equation are multiplied by 2. This process continues until all communities are merged and form a single community.

$$\Delta Q_{ij} = \begin{cases} \frac{1}{2m} - \frac{k_i k_j}{(2m)^2} & \text{, if i and j are connected.} \\ 0 & \text{, otherwise.} \end{cases} \tag{7}$$

m denotes the number of edges in the line, k degrees, and a_i edges connected to the nodes in the community i.

$$a_i = k_i/2m \tag{8}$$

Walktrap algorithm starts with the state in which every node is a community of its own [29]. First, the distances between all adjacent nodes are computed. Then, two communities are selected according to the distance criteria and merged. The distances are determined by using a specified length of random walks. After these steps, the distances are recalculated. The optimum number of steps is 4 or 5. In very dense graphs, the number of steps should be smaller.

Louvain consists of two phases. In the first phase, each node is assigned to a different community. Then, every node is merged with each of its neighbors. The gain of modularity is calculated. If the gain is positive, the neighbor giving the maximum modularity is assigned to the same community as the current node. The gain is calculated by using Equation (9).

$$\Delta Q = \left\lceil \frac{\sum_{in} + 2k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2 \right\rceil - \left\lceil \frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2 \right\rceil \tag{9}$$

If the given graph is unweighted, it takes the weights of all edges as 1. $\sum_{in}$ is the sum of the weights in community C, $\sum_{tot}$ is the sum of the weights between nodes in C and neighbors, k_i is the sum of the weights of the edges between node i and neighbors, $k_{i,in}$ is the sum of the weights of edges between node i and nodes in C, and m is the sum of the weights of all the edges in the network. The first phase continues iteratively until the improvement stops [4]. In the second phase, the nodes in the same community are merged and accepted as a new node, and a new graph is generated. In this new graph, the weights of self-loops give the number of edges in the community that the node represents. The weights of edges between new nodes in the new graph are calculated by adding the weight of edges between the communities. Then the first step is repeated with the new nodes and weights. The algorithm stops when the modularity does not increase for any pair of nodes in the graph.

Spinglass is based on the idea of spinning around the network by using statistical mechanics [31]. They interpret the network community structure as a spin configuration. A spin configuration, which minimizes the energy of the spin glass with the number of possible spin states, is sought. The nodes in the same spins are assigned to the same community. However, since this algorithm requires too much computation time, it is suitable for small networks.

Label propagation algorithm that is proposed by Raghavan et al. gives a unique label to each node at the beginning [30]. In each step, labeling is done according to the most common label among neighbors. If we call l_v to the label of node v, and $N(v)$ to the set of neighbors, the new labels are determined according to Equation (10).

$$l_v = argmax_l \sum_{u \epsilon N(v)} [l_u == l] \tag{10}$$

If more than one node meets this condition, the node selection is made randomly, and this step continues until the stop criterion is met. For instance, the case in which labels of the nodes do not change can be accepted as a stop criterion.

InfoMap uses information flow on the network to detect communities [33]. It uses a random walker on the network. The random walker uses Markov transition matrix. The trajectory of the random walker is described with unique names. These unique names are obtained by Huffman coding. It gives unique names for the regions that random walker explores, but the names of nodes in different regions can be the same. It uses the same optimization method as the Louvain algorithm.

Fluid Communities algorithm which is proposed by Pares et al. [28] works for connected graphs. The number of communities should be stated at the beginning of the algorithm. In the first step, k nodes are selected randomly, and these nodes are assigned to each community respectively. The densities of the created communities are calculated with the formula $1/numberofnodes$ in the community. The community consisting of one node is the community with the maximum density, and its density is 1. The algorithm visits all the nodes randomly by starting from a random node. Candidate communities are the most frequent communities among neighbor nodes for each node, and the community with the highest density among these communities is determined as the best community. If there are many best communities, random selection is made among the communities. Until there is no change in the communities, the nodes are mixed again, and the processes are repeated.

4. The proposed algorithm

We accept the community as a collection of nodes that have more common nodes with each other than the nodes in the other communities. The proposed algorithm first calculates the similarity of two adjacent nodes and assigns it as a weight of the edge connecting these nodes. In this step, we transform the initially unweighted graph into a weighted graph. Experimenting with various similarity metrics, we found that cosine similarity is more suitable for the next steps of our algorithm and gives better results. The first important difference of our algorithm from other similarity-based algorithms is that we take the numbers in the principal diagonal of the adjacency matrix as one instead of zero. So we add the node itself to its neighbor list. It allows us to make the similarity score of two nodes more than zero if there is no common neighbor between two nodes, but they are connected.

In the next step, we sort the edges according to the weights we assign according to similarities. Then, we assume that modularity is equal to -1, and each node is a stand-alone community. We keep all communities on a community list. We evaluate the nodes at the end of each edge by starting from the one with the highest weight. If the nodes at the end of the considered edge are in different communities, we merge the communities of these nodes in the new community list. Then we calculate a new modularity score by using this new community list. If the new modularity is higher than the previous one, we accept the new modularity and the new community list as our modularity and community list. Otherwise, we do not change the list and the modularity. These steps are terminated when all the edges are evaluated. The pseudo-code of our algorithm is given by Algorithm 1. We use NumPy and NetworkX Python libraries for implementation.

In Figure 1, we give an example graph and show the steps of our algorithm. First of all, we add each node to its adjacency list and calculate cosine similarities. Then, the edges are sorted in descending order according to the cosine similarity of node pairs that are connected by the edge. The algorithm merges the nodes with the highest similarity. Then, it calculates modularity. If the modularity increases, the nodes are merged. Otherwise, the merge process is canceled. For nodes 1 and 3, the merge process does not apply since they already belong to the same community. The same situation is also valid for the nodes pair (6,7) and (3,4). Since the modularity decreases, the merge process is not applied for the nodes pair (4,6). Finally, the algorithm terminates and returns the community list when there is no pair of nodes.

Accepting n is the number of nodes and m is the number of edges, cosine similarities are calculated in $O(n^3)$, sorting is completed in $O(m \log m)$. for loop works in $O(m)$, for loop completed in $O(mn)$. The total complexity of the algorithm equals to $O(n^3) + O(m \log m) + O(mn)$. Hence in dense graphs, m equals $O(n^3)$, the complexity of our algorithm is $O(n^3)$.

Algorithm 1: The proposed algorithm.

```
Define each node as an independent community
Calculate weights by using cosine similarities, considering each node as a neighbor to itself
Sort edges by descending order according to their weights
Set modularity and new modularity to –1
for each edge form the sorted edge list do
    Assign communities to temp communities
    if nodes u and v at both ends of the current edge belong to different communities then
        Merge the community of u and v, then assign it to the temp communities
        Remove the community of v from the temp communities, since it is merged
        Calculate new modularity of the temp communities
        if new modularity increases or does not change then
            Assign temp communities as communities
            Assign new modularity as modularity
return list of communities
```

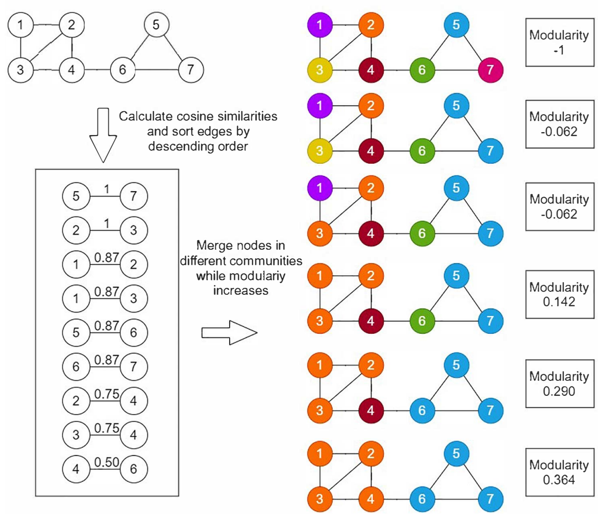
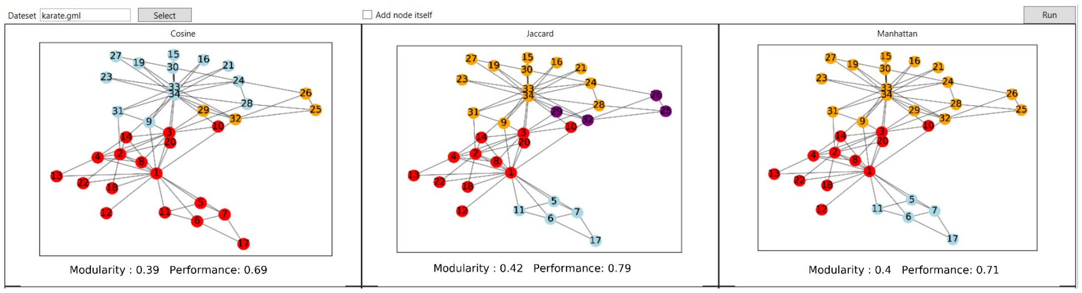

Calculate cosine similarities and sort edges by descending order
Merge nodes in different communities while modulariy increases
Modularity -1
Modularity -0.062
Modularity -0.062
Modularity 0.142
Modularity 0.290
Modularity 0.364
1
0.87
0.75
0.50

Figure 1. Steps of the proposed algorithm.

After we use cosine similarity, we compare the success of this metric with Jaccard and Manhattan. Results for the karate dataset are given in Figure 2. It is seen that Jaccard similarity is better than the others. But when we add the node itself to the neighbor list, we observe that cosine similarity gives the same result as Jaccard similarity, that the node itself is not included in the neighbor list. For cosine similarity, since it considers the connections ignored in the default version, adding the node itself to the neighbor list causes a more detailed division. Since it uses different neighbors, adding the node itself does not affect the result of the Manhattan. Because of this situation, we add the node itself to the adjacency list only for cosine similarity.



Dateset: karate.gml

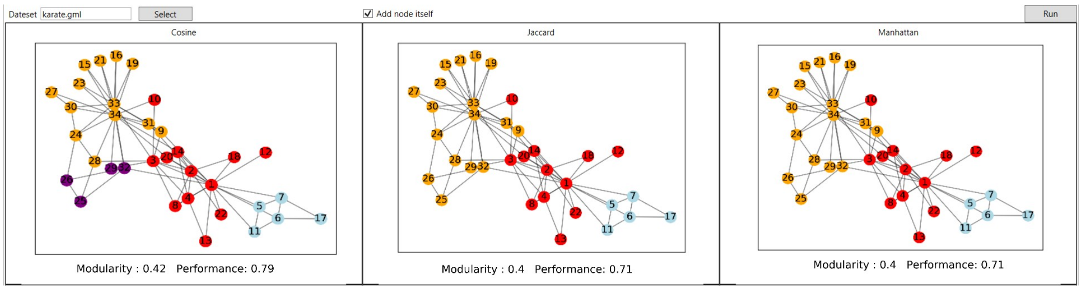
Add node itself: ☐

Modularity : 0.39   Performance: 0.69

Modularity : 0.42   Performance: 0.79

Modularity : 0.4   Performance: 0.71

Results of similarity metrics.



Dateset: karate.gml

Add node itself: ☑

Modularity : 0.42   Performance: 0.79

Modularity : 0.4   Performance: 0.71

Modularity : 0.4   Performance: 0.71

Adding node itself to the neighborhood list.

Figure 2. Comparision of similarity metrics on Zachary's karate club.

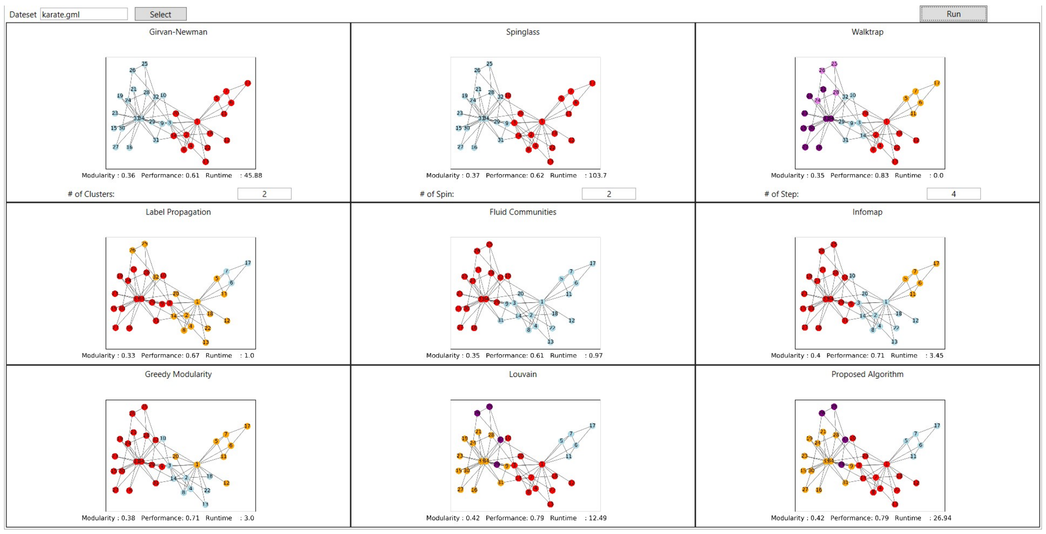
We compare these metrics with the datasets that are labeled in Table 2. In the football dataset, all metrics produce the same outputs. In polbooks and dolphins datasets, cosine similarity has the best scores according to modularity and performance. On the other hand, Jaccard has the best score in NMI. In the polbooks dataset, cosine similarity is better in modularity and NMI, but Manhattan is better in performance. Results indicate that none of the metrics outperforms the other based on all evaluation criteria. Since cosine similarity has the highest scores in modularity for all of the four datasets, we decide to use cosine similarity in our algorithm.

Table 2. Comparison of similarity metrics.

Dataset	**Similaritymetric**	**Modularity**	**Performance**	**NMI**	**Number of communities**
Karate	Jaccard	**0.4156**	**0.7861**	0.6301	4
	Cosine	**0.4156**	**0.7861**	0.6301	3
	Manhattan	0.4020	0.7112	**0.7112**	3
Football	Jaccard	0.6044	0.9246	0.8734	9
	Cosine	0.6044	0.9246	0.8734	9
	Manhattan	0.6044	0.9246	0.8734	9
Dolphins	Jaccard	0.5153	0.7911	**0.6831**	4
	Cosine	**0.5203**	**0.8091**	0.6570	4
	Manhattan	0.5034	0.8006	0.5762	4
Polbooks	Jaccard	0.5223	0.7286	0.5585	4
	Cosine	**0.5267**	0.7313	**0.5930**	4
	Manhattan	0.5157	**0.7476**	0.5414	4

5. Experimental results

We calculate the success of our algorithm by using performance, modularity, and NMI metrics when the number of communities is known. The results for the karate dataset are given in Figure 3. The modularity and performance scores increase when the proposed algorithm divides the karate dataset into two more communities. On the other hand, there are only two communities in the actual dataset.



Dateset karate.gml

Girvan-Newman: Modularity : 0.36 Performance: 0.61 Runtime : 45.88
# of Clusters: 2

Spinglass: Modularity : 0.37 Performance: 0.62 Runtime : 103.7
# of Spin: 2

Walktrap: Modularity : 0.35 Performance: 0.83 Runtime : 0.0
# of Step: 4

Label Propagation: Modularity : 0.33 Performance: 0.67 Runtime : 1.0

Fluid Communities: Modularity : 0.35 Performance: 0.61 Runtime : 0.97

Infomap: Modularity : 0.4 Performance: 0.71 Runtime : 3.45

Greedy Modularity: Modularity : 0.38 Performance: 0.71 Runtime : 3.0

Louvain: Modularity : 0.42 Performance: 0.79 Runtime : 12.49

Proposed Algorithm: Modularity : 0.42 Performance: 0.79 Runtime : 26.94

Figure 3. Outputs for Zachary's karate club dataset.

We also make an experiment to see the success of our algorithm when the number of communities is known. After finding the communities with the proposed algorithm, we merge a community pair in each step and calculate the modularity. Then the modularity values are sorted by decreasing order and the community pair which produces maximum modularity is used for merging. The algorithm stops when it reaches the given number of communities.

When the number of communities is known, our algorithm produces the same results as the original communities in the karate dataset except node 10 (Figure 4). There is also only one error, which is on node 3, in the Girvan–Newman algorithm. The modularity, performance, and NMI results of the Girvan–Newman algorithm are 0.402, 0.7112, and 0.8365, respectively. The proposed algorithm gives a higher NMI score when the number of communities k is given. If we want higher modularity, the number of communities must be higher in this dataset.

In the dolphins dataset, our algorithm produces the same result as the original. It also increases the NMI value of the polbooks. We could not apply this procedure to the football dataset since the algorithm produces fewer communities than the original ones. Although knowing the number of communities increases the NMI, it decreases the modularity and performance, as seen in Table 3. Since lots of real-world datasets have no label for the communities, we pay more attention to modularity and performance metrics.

We compare our algorithm with some basic algorithms, and the results are given in Figure 5. The proposed algorithm gives high modularity and performance values in each dataset. Based on modularities, Infomap, Louvain and our algorithm give the best results in four datasets. While Infomap gives the best performance for three datasets, Walktrap, Louvain, and the proposed algorithm give better results in the karate dataset. The proposed algorithm performs the same success in four datasets as Louvain.

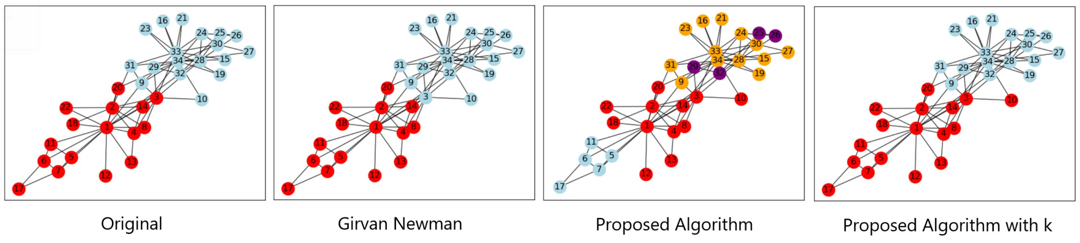


Figure 4. Results for Zachary's karate club dataset.

Table 3. Proposed algorithm with k-communities.

Data set	**Modularity**	**Performance**	**NMI**	**k**
Karate	0.3718	0.6185	0.8371	2
Dolphin	0.3735	0.5219	1.0	2
Polbooks	0.5216	0.7119	0.5934	3

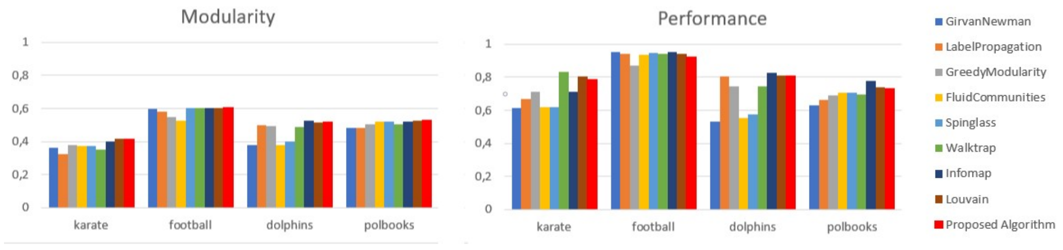


Figure 5. Modularity and performance of algorithms.

Since the complexity of cosine similarity calculation is O(n^3), our algorithm also works in O(n^3). For the case of sparse networks, an adjacency list can be used instead of an adjacency matrix to decrease the running time of the algorithm. To speed up our algorithm, we decide to use base communities instead of starting from the community list in which each node is an independent community. For this purpose, we find the central nodes by using Algorithm 2. Firstly, all nodes are sorted in descending order by degrees and kept on a possible central nodes list. The node with the highest degree in this list is accepted as a central node. This node is put into the central nodes list. Then, the neighbors of the node are eliminated from the possible central nodes list. The algorithm continues from the next node with the highest degree in the possible central node list and terminates when this list is empty. Finally, it returns the list of central nodes with k communities.

Algorithm 2: Find central nodes.

```
Sort nodes by descending order according to their degrees and keep them on the possible central nodes list
Add the node with the highest degree from the possible central node list to the central nodes list
Remove the node with the highest degree from the possible central node list
while possible central node list is not empty do
    Find the neighbors of the central node
    Remove the neighbors from the possible central node list
    if possible central node list is not empty then
        Add the node with the highest degree from the possible central node list to the central nodes list
        Remove the node with the highest degree from the possible central node list
return central nodes list
```

After we find the central nodes, we change the first step of Algortihm 1 (creating initial communities). We assume that each central node with its neighbors forms a community. Naturally, in this case, nodes that are neighbors to more than two central nodes will be in many communities at the same time. Then we remove such nodes from all the communities in which they are, and we apply the other steps of Algorithm 1 to the edges where one of the ends is such node. The results are given in Table 4. We find the same communities with the original data in the karate and the dolphins datasets by using the central nodes.

Table 4. Proposed algorithm with k-communities and central nodes.

Data set	**Modularity**	**Performance**	**NMI**	**k**
Karate	0.3715	0.6168	1.0	2
Dolphin	0.3735	0.5219	1.0	2
Polbooks	0.4881	0.6471	0.5480	3

After these experiments, we test proposed algorithms on real datasets. Since the number of communities is unknown in some datasets, we run our algorithm without using this information. Results in Table 5 indicate that the proposed algorithm has high modularity and performance. Using central nodes increases the number of communities except for the football dataset. This increase also gives better modularities except for the yeast and the router datasets. While both versions of the algorithm produce the same number of communities, using central nodes decreases the modularity. Another interesting point is that producing 112 and 1240 communities give very close modularity values in the PGP dataset. Because of that, using central nodes when the number of communities is unknown is not recommended, although it increases the speed of our algorithm.

Table 5. Results of the proposed algorithms.

Data set	**Number of nodes**	**Number of edges**	**k**	**Proposed algorithm**		**Proposed algorithm with central nodes**	
				Modularity	**k**	**Modularity**	**k**
Karate	34	78	2	0.4156	4	0.3774	4
Football	115	613	12	0.6044	9	0.5395	8
Dolphin	62	159	2	0.5203	5	0.5188	5
Polbooks	105	441	3	0.5267	4	0.5141	4
Les Misérables	77	254	-	0.5555	7	0.5225	7
NetScience	379	914	-	0.7585	10	0.7989	31
Email	1133	5451	-	0.3936	5	0.5130	27
Yeast	2375	11693	-	0.7213	22	0.7181	168
Web spam	4767	37375	-	0.3880	58	0.4336	209
Router	5022	6258	-	0.8777	36	0.7850	623
Bio dmela	7393	25569	-	0.3125	26	0.3277	394
PGP	10680	24316	-	0.8020	112	0.8065	1240

6. Conclusion

Using cosine similarity, we propose a community detection algorithm that is easy to implement. The algorithm gives the output in one iteration on edges. Since it does not use random generation, it gives the same output for the same input. In addition, it does not require any prior knowledge. It can work without prior data, such as the number of clusters, iteration, content data, etc., and produces more modular communities. Also, it has high performance and modularity in different datasets. Besides, we test the NMI values of our algorithms with the given number of communities. Although knowing the number of communities increases the success of the algorithm according to the NMI, it decreases the modularity. On the other hand, we also run our algorithm using base clusters. We decide on the base clusters by using central nodes. Since this method makes our algorithm faster, it is not recommended when the number of communities is unknown, because of increasing the number of communities too much. Additionally, one of the advantages of our algorithm is that it is easy to modify and suitable for improvement. In the next study, we plan to improve our algorithm to detect overlapping communities [9].

References

[1] Abduljabbar DA, Hashim SZM, Sallehuddin R. An improved multi-objective evolutionary algorithm for detecting communities in complex networks with graphlet measure. Computer Networks 2019; 169. doi: 10.1016/j.comnet.2019.107070

[2] Arasteh M, Alizadeh S. A fast divisive community detection algorithm based on edge degree betweenness centrality. Applied Intelligence 2019; 49 (2): 689–702. doi: 10.1007/s10489-018-1297-9

[3] Arasteh M, Alizadeh S. Community detection in complex networks using a new agglomerative approach. Turkish Journal of Electrical Engineering and Computer Sciences 2019; 27 (5): 3356–3367. doi: 10.3906/elk-1902-163

[4] Blondel VD, Guillaume J, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008; 10. doi: 10.1088/1742-5468/2008/10/p10008

[5] Bouyer A, Roghani H. LSMD: A fast and robust local community detection starting from low degree nodes in social networks. Future Generation Computer Systems 2020; 113: 41–57. doi: 10.1016/j.future.2020.07.011

[6] Choudhary N, Minz S, Bharadwaj KK. Circle-based group recommendation in social networks. Soft Computing 2021; 25 (22): 1433–7479. doi: g/10.1007/s00500-020-05356-y

[7] Clauset A, Newman MEJ, Moore C. Finding community structure in very large networks. Physical Review E 2004; 70 (6): 1–6. doi: 10.1103/PhysRevE.70.066111

[8] Cordasco G, Gargano L. Community detection via semi-synchronous label propagation algorithms. In: 2010 IEEE International Workshop on: Business Applications of Social Network Analysis (BASNA); Bangalore, India; 2010. pp. 1-8.

[9] Çetin P, Emrah Amrahov Ş. A new overlapping community detection algorithm based on similarity of neighbors in complex networks. Kybernetika 2022; 58 (2): 277-300. doi: 10.14736/kyb-2022-2-0277

[10] Emrah Amrahov S, Tugrul B. A community detection algorithm on graph data. In: 2018 International Conference on Artificial Intelligence and Data Processing (IDAP); Malatya, Turkey; 2018. pp. 1-4.

[11] Fang-ju AI. Research on a large-scale community detection algorithm based on non-weighted graph. Cluster Computing 2017; 22 (2): 2555–2562. doi: 10.1007/s10586-017-1326-1

[12] Fortunato S. Community detection in graphs. Physics Reports 2010; 486 (3): 75–174. doi: 10.1016/j.physrep.2009.11.002

[13] Freeman LC. A set of measures of centrality based on betweenness. Sociometry 1977; 40 (1): 35–41. doi: 10.2307/3033543

[14] Girvan M, Newman MEJ. Community structure in social and biological networks. National Academy of Sciences 2002; 99 (12): 7821–7826. doi: 10.1073/pnas.122653799

[15] Hosseini R, Rezvanian A. AntLP : Ant-based label propagation algorithm for community detection in social networks. CAAI Transactions on Intelligence Technology 2020; 5 (1): 34–41. doi: 10.1049/trit.2019.0040

[16] Hu X, He W, Li L, Lin Y, Li H et al. An efficient and fast algorithm for community detection based on node role analysis. International Journal of Machine Learning and Cybernetics 2017; 10 (4): 641–654. doi: 10.1007/s13042-017-0745-x

[17] Huang L, Wang G, Wang Y, Pang W, Ma Q. A link density clustering algorithm based on automatically selecting density peaks for overlapping community detection. International Journal of Modern Physics B 2016; 30 (24). doi: 10.1142/S0217979216501678

[18] Imtiaz ZB, Manzoor A, Islam S, Judge MA, Choo KKR et al. Discovering communities from disjoint complex networks using multi layer ant colony optimization. Future Generation Computer Systems 2020; 115: 659–670. doi: 10.1016/j.future.2020.10.004

[19] Jokar E, Mosleh M. Community detection in social networks based on improved label propagation algorithm and balanced link density. Physics Letters A 2018; 383 (8): 718–727. doi: 10.1016/j.physleta.2018.11.033

[20] Karataş A, Şahin S. Application areas of community detection: a review. In: 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT); Ankara, Turkey; 2018. pp. 65-70.

[21] Lu H, Yao Q. Modularity maximization for community detection using genetic algorithm. Neural Information Processing 2018; 11 (302): 463–472. doi: 10.1016/j.physa.2012.11.003

[22] Mcauley J, Leskovec J. Discovering social circles in ego networks. ACM Transactions on Knowledge Discovery from Data 2014; 8 (4): 1–28. doi: 10.1145/2556612

[23] Meena P, Pawar M, Pandey A. A survey on community detection algorithm and its applications. Turkish Journal of Computer and Mathematics Education (TURCOMAT) 2021; 12 (6): 4807-4815.

[24] Mohammadi M, Fazlali M, Hosseinzadeh M. Accelerating Louvain community detection algorithm on graphic processing unit. The Journal of Supercomputing 2020; 77 (6): 6056–6077. doi: 10.1007/s11227-020-03510-9

[25] Newman M. Networks an Introduction. Oxford, NY, USA: Oxford University Press, 2010.

[26] Newman MEJ. Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America 2006; 103 (23): 8577–8582. doi: 10.1073/pnas.0601602103

[27] Okuda M, Satoh S, Sato Y, Kidawara Y. Community detection using restrained random-walk similarity. IEEE Transactions on Pattern Analysis and Machine Intelligence 2021; 43 (1): 89–100. doi: 10.1109/TPAMI.2019.2926033

[28] Pares F, Garcia-Gasulla D, Vilalta A, Moreno J, Ayguade E et al. Fluid communities: A competitive, scalable and diverse community detection algorithm. Complex Networks and Their Applications 2017; (689): 229–240.

[29] Pons P, Latapy M. Computing communities in large networks using random walks. In: ISCIS 2005: Computer and Information Sciences; Istanbul, Turkey; 2005. pp. 284–293.

[30] Raghavan UN, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E 2007; 76 (3). doi: 10.1103/PhysRevE.76.036106

[31] Reichardt J, Bornholdt S. Statistical mechanics of community detection. Physical Review E 2006; 74 (1). doi: 10.1103/PhysRevE.74.016110

[32] Rizi FS, Granitzer M, Ziegler K. Global and local feature learning for ego-network analysis. In: 2017 28th International Workshop on Database and Expert Systems Applications (DEXA); Lyon, France; 2017. pp. 98–102.

[33] Rosvall M, Bergstrom CT. Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences 2008; 105 (4): 1118–1123. doi: 10.1073/pnas.0706851105

[34] Sani NS, Manthouri M, Farivar F. A multi-objective ant colony optimization algorithm for community detection in complex networks. Journal of Ambient Intelligence and Humanized Computing 2020; 11 (1): 5–21. doi: 10.1007/s12652-018-1159-7

[35] Yang G, Zheng W, Che C, Wang W. Graph-based label propagation algorithm for community detection. International Journal of Machine Learning and Cybernetics 2020; 11 (6): 1319–1329. doi: 10.1007/s13042-019-01042-0

[36] You X, Ma Y, Liu Z. A three-stage algorithm on community detection in social networks. Knowledge-Based Systems 2020; 187. doi: 10.1016/j.knosys.2019.06.030

[37] Zhang Y, Liu Y, Jin R, Tao J, Chen L et al. GLLPA: A graph layout based label propagation algorithm for community detection. Knowledge-Based Systems 2020; 206. doi: 10.1016/j.knosys.2020.106363

[38] Zhang Y, Liu Y, Li J, Zhu J, Yang C. WOCDA: A whale optimization based community detection algorithm. Physica A: Statistical Mechanics and its Applications 2020; 539. doi: 10.1016/j.physa.2019.122937