IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 12, 2021, accepted May 24, 2021, date of publication May 26, 2021, date of current version June 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3083971

Large-Scale Network Community Detection Using Similarity-Guided Merge and Refinement

VOLKAN TUNALI, (Member, IEEE)
Department of Software Engineering, Faculty of Engineering and Natural Sciences, Maltepe University, 34857 Istanbul, Turkey
e-mail: volkan.tunali@gmail.com

ABSTRACT It is possible to extract valuable insights about the functional properties of a system by identifying and inspecting the community structure in the network that models the system. Community detection aims to extract these community structures from networks. Many community detection methods have been proposed that consider the problem from different perspectives. However, with the emergence of very large and complex networks from variety of domains, there has been a growing need for community detection methods that can operate at scale effectively and efficiently. Considering this, we propose a novel algorithm for large-scale community detection, based on two novel similarity indices we propose as well. In the first stage of our proposed algorithm, we generate candidate communities using a mechanism similar to information propagation very rapidly. Then, we merge small candidates that have fewer nodes than a calculated threshold with the larger ones using similarity between nodes and communities. Next, we engage a refinement operation on the candidates by moving all nodes to the candidates to which they are most similar using the same similarity index again. After that, we merge small communities with larger ones by using the similarity between communities until no gain in the modularity is obtained. Finally, in the last stage, we employ the same refinement operation as in the third stage. With an extensive experimentation on real-world and artificially-generated benchmark networks, we demonstrate and verify the performance and effectiveness of the proposed algorithm comparing it with the state-of-the-art methods. Experimental results indicate that our algorithm scales very well with growing size and complexity of networks. Besides, our algorithm outperforms most state-of-the-art community detection methods both in detection performance and computation time.

INDEX TERMS Network community detection, network clustering, network science, network analysis, complex networks.

I. INTRODUCTION

Any complex system that is made of a number of interacting components can be modeled with a network structure, where nodes represent the components and edges represent the interactions among these components [1]. This makes network models a highly useful tool to study many phenomena in social sciences, economics, biology, ecology, logistics, communication, and so on along with the advances in computing technology and the techniques developed to analyze especially large and complex networks under the interdisciplinary research field of network science [2]. In many complex networks, we observe a natural tendency to form dense groups of nodes that are typically denser than the whole network. These dense structures are called network communities, and these communities are usually related to the functional units of the systems modeled by networks [3], [4]. For example, in social networks, communities mostly correspond to real social professional or interest groups. In book co-purchase networks, they probably represent the group of customers with similar reading taste or interest. Communities in protein interaction networks and metabolic networks, for instance, correspond to the functional modules. Therefore, by identifying and studying the community structures in networks, it is possible to extract valuable insights about the functional properties of the underlying systems. The problem of identifying and extracting communities embedded in networks is called community detection. It is a very active research area where many methods and tools have been developed [3].

Beginning from early 2000's, the development of computational technologies has enabled us to generate, collect, transfer, and store large amounts of data in almost every aspect

The associate editor coordinating the review of this manuscript and approving it for publication was Zhan Bu.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

of daily life including but not limited to business operations, scientific studies, and social interactions. Network data is no exception to this. When compared to past ones, scale and complexity of new networks that researchers and practitioners need to deal with have increased tremendously. Computational power of modern hardware can help analyze and mine these networks only to some extent unfortunately. There is still a growing need for community detection methods that can deal with today's large-scale and complex networks fast and efficiently.

In this paper, we propose a novel community detection method to detect communities especially in large-scale and complex networks effectively and fast. Our method is largely based on two different concepts of similarity: (i) similarity between a node and a community, and (ii) similarity between two communities.

The main contributions and novelty of our work are threefold:

1) We propose two similarity indices that have not been introduced in the network science domain before to the best of our knowledge, and show that they can be utilized highly effectively in community detection.
2) We propose a fast and effective community detection algorithm that utilizes the similarity indices we propose.
3) Finally, with an extensive experimentation on real-world and artificially-generated benchmark networks, we demonstrate and verify the performance and effectiveness of the proposed algorithm comparing it with the state-of-the-art methods.

Experimental results show that our algorithm scales very well with growing size and complexity of networks. In addition, our algorithm outperforms most state-of-the-art community detection methods both in detection quality and computation time.

The rest of this paper is organized as follows. In Section II, we provide an essential background on community detection methods. In Section III, we elaborate the design of our proposed algorithm including the proposed similarity indices and the algorithm stages. We give the details of our experimental methodology in Section IV. We present our experimental results and discussion in Section V. Finally, in Section VI, we conclude the paper with some directions for future research.

II. RELATED WORK

There are numerous community detection methods in the literature that have been developed over the years, and these methods address the community detection problem from various perspectives. In this section, first, we succinctly review the main approaches to community detection problem, which are regarded as the state-of-the-art in this field. In our research, we compare our proposed method with all these. Second, we briefly review some more recent community detection studies.

Clauset-Newman-Moore (CNM) [5] is one of the earliest attempts to find community structure in very large networks. It performs a greedy optimization of the quantity known as Modularity [6], and produces a hierarchical community structure that can be visualized in a dendrogram. It employs a bottom-up agglomerative strategy where communities are merged greedily if their merging results in an increase in modularity.

Louvain [7] is one of the most widely-known community detection algorithms due to its effectiveness and speed, and it is implemented in many network analysis software packages and tools. Louvain is also an agglomerative hierarchical algorithm based on modularity optimization. It initially creates communities from every single node. Then, in a node-moving procedure it seeks for modularity gain by moving nodes between communities. In the next round, the same procedure is repeated but instead of nodes, whole communities are moved as if they were the nodes of the network of that round. The algorithm performs this operation repeatedly until no more modularity gain is possible. In each round, as communities get larger, the output of the algorithm becomes a hierarchical partitioning as in CNM.

Label Propagation (LP) algorithm [8], as its name implies, uses the information propagation mechanism to find communities in networks. Initially, every node is assigned a unique label representing a unique community. Then, in an iteration, every node is assigned the label of the majority of its neighbors. The iteration ends when the label of every node is the same as the label of the majority of its neighbors. LP is known to be one of the fastest algorithms in community detection due to its simple design. It is, however, not very stable when the network is not sparse and communities are not clear.

Infomap algorithm [9] uses the principles from information theory and exploits random walk dynamics. It considers the community detection problem as a problem of finding a description of minimum information from a random walk within the network by trying to optimize what is called minimum description length as an objective function. Infomap is also known to be both very fast and very effective.

Order Statistics Local Optimization Method (OSLOM) algorithm [4] is specifically designed to handle directed and weighted networks with overlapping communities. It is based on local optimization of a fitness function that expresses the statistical significance of communities with respect to global null model. OSLOM can be used as a stand-alone community detection algorithm, or it can be used to refine the communities found by other methods. It is known to identify very high-quality communities but it suffers from high computational cost.

In a study by Žalik [10], a community detection method based on maximal similar neighborhood concept was proposed. Communities are detected by starting from communities formed by adjacent pairs of nodes and their maximal similar neighbors. Using several different similarity indices and depending on a k parameter, they show that their method can extract communities of different resolutions based on the

experiments on both real-world and relatively small artificial benchmark networks ($N_{max} = 5{,}000$).

Chen *et al.* [11] proposed the concept of ''community centrality'' for finding communities in networks. Their idea considers the communities as high-density nodes surrounded by lower-density neighbors. Besides, according to their community definition, community centers should be far away from each other. They proposed two different density indices calculated for a node and used these indices to identify central nodes. Then, using these central nodes, they employ a label propagation to recover the communities around them. They show the effectiveness of their method on both real-world and relatively small artificial benchmark networks ($N_{max} = 1{,}000$).

In a recent work by Shang *et al.* [12], a new community integration strategy was proposed to improve network community detection for large-scale networks. The main idea in this work is to merge candidate communities by considering an improved concept called modularity density increment. The paper empirically demonstrates the superiority of the proposed approach on small to large-sized real-world networks ($N_{max} = 500{,}000$).

In another work by Shang *et al.* [13], K-nearest neighbor and label propagation algorithms are combined to generate sub-communities initially. Then, these sub-communities are merged depending on modularity density rather than classical modularity. Effectiveness of the approach is demonstrated on both small to moderate-sized real-world networks and moderate-sized artificial benchmark networks ($N_{max} = 10{,}000$).

In a recent study by Meng *et al.* [14], a robust distance dynamics model called Attractor++ was proposed to improve community detection. Attractor++ uses a dynamic membership degree to determine the influence of exclusive neighbors on the distance instead of using a cohesion parameter used in distance dynamic models in general. They conducted experiments on both real-world and moderate-sized artificial benchmark networks ($N_{max} = 60{,}000$), and demonstrated the effectiveness of their approach.

Cheng *et al.* [3] proposed a neighbor similarity based agglomerative method for network community detection. Their method called NSA is based on both similarity between nodes and similarity between communities. NSA first identifies the preliminary community structure by using the similarity between two nodes. Then, it tries to identify small and sparse communities based on the concepts of community sparsity and community scale they propose. Next, it tries to merge identified small and sparse communities by using the index they also propose to calculate the similarity between two communities. They show the effectiveness of NSA on both real-world and relatively small artificial benchmark networks ($N_{max} = 5{,}000$).

Luo *et al.* [15] proposed two signed modularity functions to evaluate the community properties in networks. Using the proposed modularity functions, they proposed a dynamic expansion algorithm based on local information for global community detection. Their algorithm gradually detects the community structure by dynamically expanding the community size and adding nodes through an initial node with community label. They demonstrated the effectiveness of the algorithm both on very large real-world benchmark networks and moderate-sized artificial networks ($N_{max} = 20{,}000$).

Cao *et al.* [16] proposed a method to efficiently detect communities specifically within the smart grid structure by considering the problem as a multi-objective optimization problem and offering a new dynamic game model. Their approach tries to find locally Pareto-optimal user community groups in the smart grid. The proposed approach is shown to be effective at detecting corresponding communities in several IEEE power grid model networks and artificial benchmark networks with ground truth communities ($N_{max} = 5{,}000$).

In a very recent work by Bu *et al.* [17], community detection in social media networks is translated into a multi-objective optimization problem by modeling a social media network as a discrete-time dynamical system. In such systems, the opinion matrix and the community structure are expected to affect each other mutually. The paper proposes a new graph K-means framework that employs a series of phases. The validity and the performance of the proposed approach is presented on several real-world benchmark datasets of varying sizes ($N_{max} = 4$M).

Hu *et al.* [18], also in a very recent study, proposed a new algorithm called Node2vec-SC that combines node2vec with spectral clustering. They use node2vec, which is an algorithm framework for learning continuous feature representation for nodes in networks, to learn representations that organize nodes based on their network roles or communities they belong to. Then, they use the spectral clustering algorithm to find the communities that belong to the networks. They show the effectiveness of Node2vec-SC on both real-world and moderate-sized artificial benchmark networks ($N_{max} = 10{,}000$).

Traag *et al.* [19] claimed that Louvain algorithm has an important flaw that it produces artbitrarily badly-connected, even disconnected, communities, and proposed a new algorithm to address this problem. They proved that their algorithm, called Leiden, is guaranteed to produce connected communities, and it is also faster than Louvain. They also demonstrated the performance of the Leiden algorithm on real-world and artificial benchmark networks ($N_{max} = 40$M). Since Leiden was a relatively new algorithm that could be applied on large-scale networks and it was claimed to be superior to Louvain, we included it in our experimental comparison.

In a very recent work, Choumane *et al.* [20] proposed a new community detection method, called Core Expansion (CoreExp), that finds communities independently from modularity. That is, they discover the number of communities and their members without calculating modularity. Instead, they first detect the core of each possible community in the network. Then, they expand each core by adding the nodes to

form the final communities based on a neighborhood overlap measure in an iterative manner. They conducted experiments on real-world and artificial benchmark networks to show the performance of their method ($N_{max} = 330{,}000$). Since CoreExp was also a very new algorithm that could be run on very large networks, we added it in our experimental comparison, too.

In the most recent study we review here, Xu *et al.* [21] proposed a community detection algorithm based on classical label propagation algorithm due to its low computational complexity. They improved the label propagation by employing a two-level neighborhood similarity measure called TNS. They empirically show that their approach produces promising results on both small to moderate-sized real-world networks and relatively small artificial benchmark networks ($N_{max} = 30{,}000$).

The recent methods we review here approach the community detection problem from different perspectives, and they propose different methods to solve it. Each method was shown to be effective at recovering community structures based on systematic experimentation. An important issue that attracts our attention is that these methods were mostly experimented on relatively small benchmark networks, real-world or artificial. Particularly, for the artificial benchmark network generation, they generally picked relatively small number of nodes ($N = 1{,}000$ to $60{,}000$). Moreover, some of them picked relatively small mixing parameter μ, which results in the generation of benchmark networks with easily identifiable communities. In our research, on the other hand, our primary goal is to develop a community detection method that can effectively address very large and complex networks. In order to test the effectiveness of our method, we generated a large number of artificial benchmark networks with N from 1,000 to 1,000,000 and mixing parameter μ from 0.1 to 0.9. Additionally, we set the other parameters of benchmark generation to the extreme values to generate really challenging networks for all the methods that we compare.

III. PROPOSED ALGORITHM: SimCMR

In this section, we introduce our algorithm that recovers high-quality partitions of very large networks very fast while requiring very little memory space. We call our algorithm SimCMR (**Sim**ilarity-Guided **C**ommunity **M**erge and **R**efinement) because it consists of several community merge and refinement stages based on similarity notion, which are described in great depth in this section. SimCMR algorithm is designed for detecting non-overlapping communities in unipartite, undirected, and unweighted networks.

Throughout this paper, we consider an undirected and unweighted network as a graph $G(V, E)$ such that V is the set of N nodes ($|V| = N$) and E is the set of M edges ($|E| = M$). SimCMR can find optimal non-overlapping community structure C such that $C = \{c_1, c_2, \ldots, c_k\}$ where $c_i \neq \emptyset$ and $c_i \cap c_j = \emptyset$ for $i, j = 1 \ldots k$.

A. SIMILARITY INDICES

SimCMR algorithm heavily relies on the notion of similarity. Unlike most previous studies in the literature, SimCMR does not use the similarity between nodes. Instead, it makes use of two different similarity indices that we define in this subsection. The first one is the similarity between a node and a community (S_{nc}), and the other one is the similarity between two communities (S_{cc}).

Similarity between a node v and a community c could simply be calculated as the number of edges that connect v to c. In other words, this is the number of neighbors of v that are members of c. This simple approach, however, tends to favor large communities over smaller ones. Therefore, we normalize it by the size of the community as seen in (1).

$$S_{nc}(v, c) = \frac{|N(v) \cap c|}{|c|} \tag{1}$$

where v is the node, c is the community, and $N(v)$ is the set of the neighbors of v. S_{nc} is essentially based on the principle that links within a community are more predictable than external links [22].

The second similarity index is the one that computes the similarity between two communities according the number of edges that connect two communities. Again, this simplistic approach does not take into consideration the sizes of the communities. However, it is a worthwhile idea to normalize this value by the total size of the communities as given in (2).

$$S_{cc}(a, b) = \frac{C(a, b)}{|a| + |b|} \tag{2}$$

where a and b are two communities, and $C(a, b)$ is the number of edges (links) connecting a to b. $C(a, b)$ is also known as the cut size between two communities a and b [23], and we calculate it in this study as in (3).

$$C(a, b) = \sum_{u \in a} |N(u) \cap b| \tag{3}$$

Number of connecting edges between two communities a and b can also be interpreted as the total number of nodes that are both members of b and are neighbors of nodes in a. It should be noted that both C and S_{cc} are symmetric indices; that is, $C(a, b) = C(b, a)$ and $S_{cc}(a, b) = S_{cc}(b, a)$.

Cut size is a very popular index in network community detection research. It has several variants that normalizes the cut size by the volume of communities defined as $Vol(a) = \sum_{i \in a} k_i$, where k_i is the degree of node i in community a [23]. In our research, however, rather than using the concept of volume, we normalize the cut size by the total size of the respective communities, which is a very simple yet fast and effective approach.

B. ALGORITHM DESCRIPTION

SimCMR algorithm mainly consists of five stages. We present these stages in Algorithm 1 as pseudocode. First, we generate candidate communities in one pass over the nodes of the network. Then, we merge small candidates,

Algorithm 1 SimCMR(G)

```
C ← Generate_Candidate_Comms(G)
C ← Merge_Candidates_by_N_to_C_Sim(G, C)
C ← Refine_Node_Comm_Assignments(G, C)
C ← Merge_Comms_by_C_to_C_Sim(G, C)
C ← Refine_Node_Comm_Assignments(G, C)
return C
```

which have fewer nodes than a calculated threshold, with the larger ones using S_{nc} similarity index. Next, we employ a refinement operation on the candidates by moving all nodes to the candidates to which they are most similar using S_{nc} similarity index again. After that, we merge small communities with larger ones by using S_{cc} similarity index until the modularity of communities does not increase. Finally, in the last stage, we employ the same refinement operation as in the third stage. In the following subsections, we describe each stage in detail.

1) CANDIDATE COMMUNITY GENERATION

In this first stage of SimCMR, we generate candidate communities simply by iterating through the nodes in the network in descending order of their degrees. The pseudocode for this stage is given in Algorithm 2. Beginning from the node with the highest degree, we create a community that contains only this node. Then, we iterate through the neighbors of this very same node and we put each of them into the same community if it has not been put into another community previously. We mark each node as visited if it has been visited previously either in the main iteration or in the neighborhood iterations. For the rest of the nodes, we repeat the same procedure if a node is not marked as visited, until no node is left unvisited. This stage of SimCMR is the fastest stage as only one pass over the nodes is required, and most of the nodes are visited in the neighborhood iterations, therefore, they do not need to be considered again in the main iteration. Depending on the topological structure of the network, this stage generates many large communities and too many small ones with varying sizes including lots of singletons.

2) MERGING CANDIDATES USING S_{nc} SIMILARITY INDEX

As the candidate generation stage of SimCMR generates too many candidate communities most of which are especially very small, in this stage we merge small ones with larger ones. The pseudocode for this stage is presented in Algorithm 3.

Before beginning merging, we calculate a threshold that represents the minimum number of nodes a community must contain if it is to stay persistently. We use a heuristic to calculate this threshold based on the size distribution of the candidate clusters. If the calculated threshold is less than two, then we set it two because a meaningful community must have at least two nodes and we also need to eliminate the candidate singletons. We developed this heuristic based on numerous experiments on a large variety of networks including both real-world and artificial ones. Our intuition in this heuristic is that if we have lots of very large candidates already from the previous stage, then it is more likely that we will have large communities in the end rather than very small ones because we never split candidate communities. Therefore, it is safe to assume a large threshold using the formula in Algorithm 3. Similarly, if there are lots of very small candidates, then we will probably have relatively small

Algorithm 2 Generate_Candidate_Comms(G)

```
C ← ∅
visited_nodes ← ∅
V ← sort(V)   by degree in descending order
for v ∈ V do
  if v ∉ visited_nodes then
    new_comm ← {v}
    C ← C ∪ {new_comm}
    for u ∈ N(v) do
      if u ∉ visited_nodes then
        new_comm ← new_comm ∪ {u}
        visited_nodes ← visited_nodes ∪ {u}
      end if
    end for
    visited_nodes ← visited_nodes ∪ {v}
  end if
end for
return C
```

Algorithm 3 Merge_Candidates_by_N_to_C_Sim(G, C)

```
h ← √((Σ_{c∈C} |c|^2) / |C|)
min_comm_size ← max(h, 2)
C ← sort(C)   by number of nodes
removed_comms ← ∅
for i ← 1 to |C| − 1 do
  if |c_i| ≥ min_comm_size then
    break
  end if
  for v ∈ c_i do
    max_sim_comm ← ∅
    max_sim ← −∞
    for j ← i + 1 to |C| do
      sim ← S_nc(v, c_j)
      if sim > max_sim then
        max_sim ← sim
        max_sim_comm ← c_j
      end if
    end for
    c_i ← c_i − {v}
    max_sim_comm ← max_sim_comm ∪ {v}
  end for
  if |c_i| = 0 then
    removed_comms ← removed_comms ∪ {c_i}
  end if
end for
C ← C − removed_comms
return C
```

Algorithm 4 Refine_Node_Comm_Assignments(G, C)

```
V ← sort(V)   by degree
for v ∈ V do
    max_sim_comm ← ∅
    max_sim ← −∞
    for c ∈ C do
        sim ← S_nc(v, c)
        if sim > max_sim then
            max_sim ← sim
            max_sim_comm ← c
        end if
    end for
    if c ≠ max_sim_comm then
        c ← c − {v}
        max_sim_comm ← max_sim_comm ∪ {v}
    end if
end for
return C
```

communities in the end. Thus, we calculate a small threshold. Without doubt, a network can naturally be made up of a collection of large and small communities, and it would be a mistake to assume a uniform community size distribution in this stage. Our heuristic formula takes into account this possibility and calculates an appropriate threshold.

We iterate through the candidate communities in descending order by the number of nodes they contain. In this iteration, if we encounter a community that has more nodes than or equal to the threshold, we stop the iteration because we are now at such a point that we have merged all very small candidates to the larger ones. In each iteration, we iterate through the nodes in the current candidate. By using S_{nc} similarity index, we find the larger candidates that are most similar to the nodes, and then we move them from the current candidate to the candidates found most similar. At the end of this stage, all candidates with fewer nodes than the threshold are eliminated, the total number of candidates rapidly drops, and finally real communities begin to emerge.

3) REFINING CANDIDATES USING S_{nc} SIMILARITY INDEX

The previous stage reduces the number of candidate communities dramatically by moving nodes in smaller ones to larger ones based on the similarity between nodes and communities. However, this does not guarantee the cohesion of the communities, and a further refinement process makes them more cohesive before merging them in the subsequent stage.

In this stage, we iterate through the nodes in the network in ascending order of their degrees. For each node, we find the community that is most similar to it, and then we move it from its current community to the one found most similar, of course if they are not the same. Note that this simple refinement process is employed twice in SimCMR, once in the third stage and then in the fifth stage, and it substantially improves the cohesiveness of the communities. The pseudocode for this stage is presented in Algorithm 4.

Algorithm 5 Merge_Comms_by_C_to_C_Sim(G, C)

```
C ← sort(C)   by number of nodes
removed_comms ← ∅
best_Q ← modularity(G, C)
for i ← 1 to |C| − 1 do
    max_sim_comm ← ∅
    max_sim ← −∞
    for j ← i + 1 to |C| do
        sim ← S_cc(c_i, c_j)
        if sim > max_sim then
            max_sim ← sim
            max_sim_comm ← c_j
        end if
    end for
    m ← max_sim_comm ∪ c_i
    K ← (C − {c_i, c_j}) ∪ {m}
    Q ← modularity(G, K)
    if Q > best_Q then
        max_sim_comm ← max_sim_comm ∪ c_i
        removed_comms ← removed_comms ∪ {c_i}
        c_i ← ∅
        best_Q ← Q
    end if
end for
C ← C − removed_comms
return C
```

4) MERGING COMMUNITIES USING S_{cc} SIMILARITY INDEX

Once we obtain cohesive communities in the previous stage, we continue merging small communities with larger ones in this stage using S_{cc} similarity index that computes the similarity between two communities. This time, we have one more metric that guides the merging process, which is modularity. Rather than simply merging most similar communities, we first check if merging two communities results in an increase in the modularity or not. The pseudocode for this stage is given in Algorithm 5.

From the smallest to the largest, for each community, we find the most similar other community that is larger than it. Then, we completely merge these two communities by moving the nodes from the small one to the large one. We calculate modularity with respect to this new community configuration, and if there is an increase in modularity, we make the merge persistent and eliminate the empty community. If there is no increase in modularity, we roll back to the previous community configuration. After all community pairs are considered, this stage ends. Now, we have the final candidate communities such that there will be no more merging of communities and the number of communities will stay the same.

At the end of this stage, even though we have obtained the highest modularity so far, there is still room for improvement. Therefore, we finally employ the refinement process once more, which almost always improves the community quality as well as modularity.

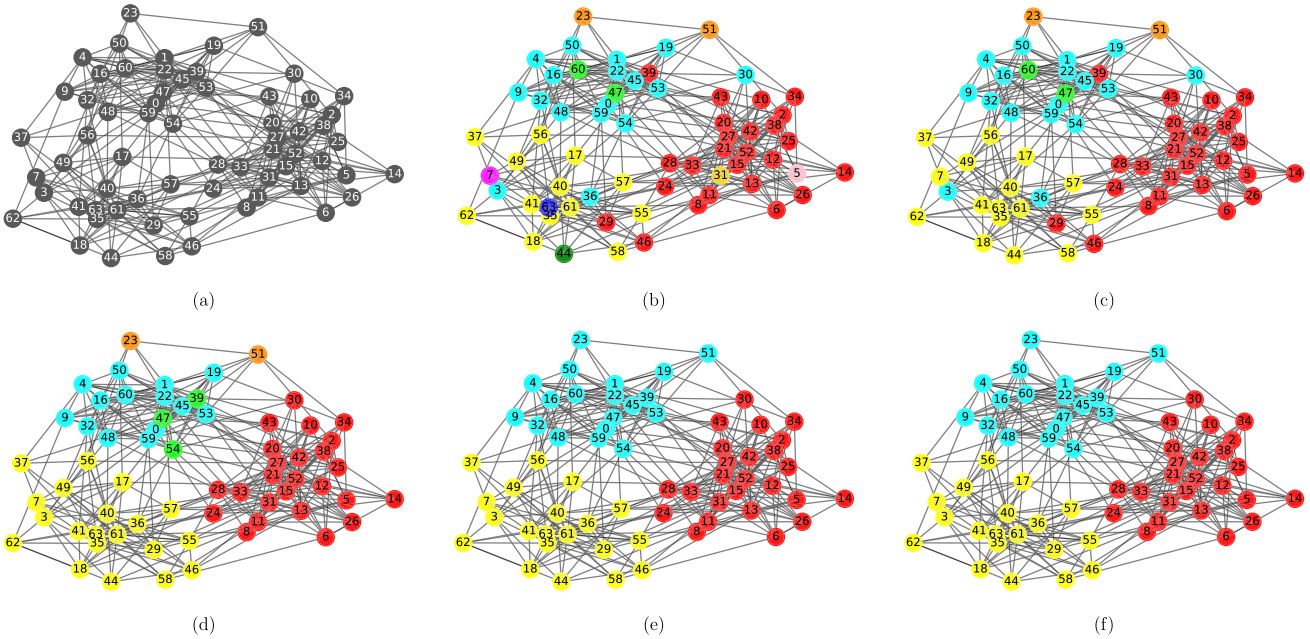

FIGURE 1. Demonstration of SimCMR stages on a sample network. (a) Original network with $N = 64$ nodes and $|C| = 3$ apparent communities (b) Candidate communities are generated in stage 1 (colors represent community membership of nodes) (c) Small candidates are merged using S_{nc} in stage 2 (d) Node community assignments are refined using S_{nc} in stage 3 (e) Communities are merged using S_{cc} and modularity in stage 4 (f) Node community assignments are refined again using S_{nc} in stage 5 (in the sample, no node changes its community).

Fig. 1 illustrates the effect of each stage on a sample network with $N = 64$ nodes and $|C| = 3$ apparent communities. Note that since the communities are very distinguishable in this sample network, they are fully recovered in stage 4 and the refinement process in the final stage has no effect. This, however, depends on the structure and the complexity of the network, and we experimentally observe that this final stage should definitely take place.

None of the stages of SimCMR involves randomness. As a result of this, SimCMR consistently produces the same communities on the same network at every run. Our experimental observations also support this situation.

C. COMPLEXITY ANALYSIS

SimCMR algorithm consists of five stages. In the first stage we generate candidate communities by passing over the nodes only once with a very simple operation with almost no computational cost. The time complexity of this stage is $O(N)$, where N is the number of nodes in the network.

In the second stage, we merge very small candidates with the larger ones using N times k calculations of S_{nc}, where k is the number of candidates. In this and the following stages of SimCMR, $N \gg k$, and assume that unit computation time of S_{nc} is F. Therefore, the time complexity of this stage is $O(N \cdot k \cdot F)$.

The third stage where we refine the node community assignments is equivalent of the second stage in terms of time complexity. We also perform N times k calculations of S_{nc}, therefore, the time complexity of this stage is also $O(N \cdot k \cdot F)$. In the fifth stage, exactly the same refinement is employed again, and its effect on the overall time complexity is also the same.

The fourth stage is the most computationally intensive stage of SimCMR. In this stage, we calculate pairwise similarities between candidate communities using S_{cc}. That is, we perform k^2 calculations of S_{cc}. In addition, for the most similar pair of candidates, we calculate modularity, which is the most costly calculation in SimCMR. However, modularity calculation is performed only for k times at most. Besides, the number of candidates considered in this stage is actually very small because the second stage of SimCMR greatly eliminates very small candidates. Assume that unit computation time of S_{cc} is G, and unit computation time of modularity is H, and $H \gg G$. As a result, the time complexity of this stage is $O(k^2 \cdot G + k \cdot H)$.

When we combine the time complexities of SimCMR stages, we find that overall time complexity of SimCMR is $O(N \cdot k \cdot F + k^2 \cdot G + k \cdot H)$. Since k is almost always much smaller than N, we can state that the dominant factor in the time complexity of SimCMR is the number of nodes N. As a result, SimCMR algorithm has a linear time complexity and it can efficiently be applied to networks of millions of nodes.

IV. EXPERIMENTAL SETTINGS

In order to demonstrate the effectiveness of SimCMR, we conducted a series of exhaustive experiments, where we applied SimCMR and several state-of-the-art community detection algorithms on a lot of benchmark network datasets for comparison. In this section, we describe the experimental settings and the experimental procedure of our research extensively.

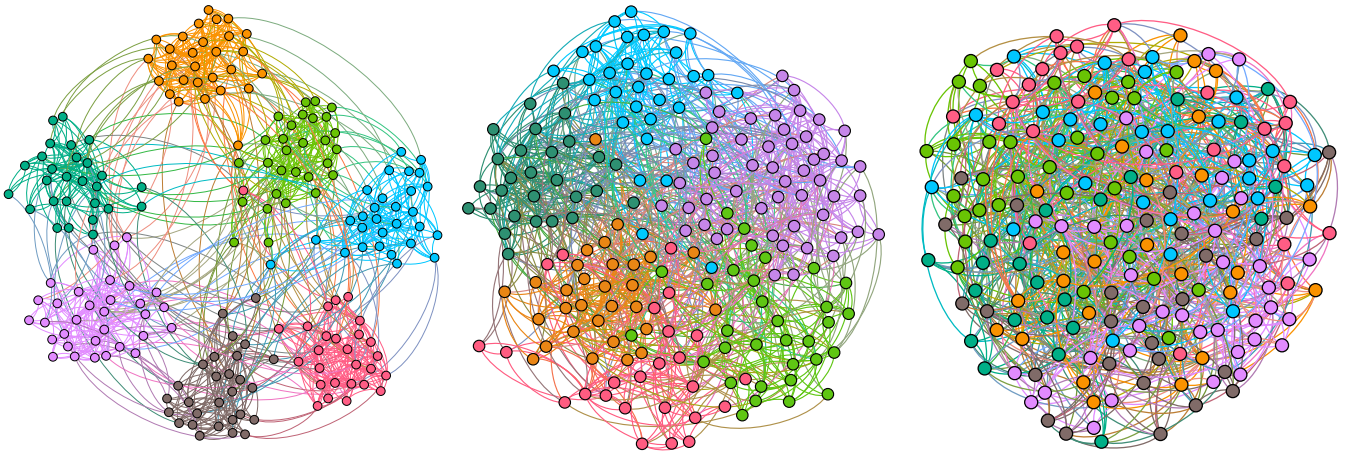

FIGURE 2. Illustration of the effect of μ on community formation and detectability (colors represent community membership of nodes). $N = 200$, $\mu = 0.1$, 0.3, and 0.5, left to right respectively.

A. NETWORK DATASETS WITH GROUND-TRUTH COMMUNITIES

The conventional approach to test the efficacy of a community detection algorithm is to test it against networks with known community structure, also called "ground-truth" or "gold-standard" communities. In this approach, we test the effectiveness of the algorithm by comparing the ground-truth communities with the ones that the algorithm reports. In order to do this, we need to use universally accepted benchmark networks. Unfortunately, in the literature, there are a small number of real-world networks with known communities, which are widely-used for benchmarking. In this research, we used the most common ones as described in Section IV-A1. Beside real-world networks, we used artificially-generated benchmark networks with ground-truth communities as it is easy to generate networks of various size and community distributions as explained in Section IV-A2.

1) REAL-WORLD NETWORKS

Although very small in general, we considered the use of five real-world networks with ground-truth communities, which are commonly used as benchmarks in network community detection research. Dolphins network is an undirected social network of frequent interactions between a group of bottlenose dolphins living off Doubtful Sound, New Zealand [24]. American College Football network is the network of American football games scheduled between Division IA colleges during regular season Fall 2000 [25]. Zachary's Karate Club network is a social network of friendships between the members of a karate club at a US university in the 1970s [26]. Political Blogs, compiled in 2005, is a directed network of hyperlinks between blogs on US politics [27]. Political Books network is a network of frequently co-purchased books about US politics that were published around the presidential election of 2004 and sold by Amazon.com [28]. General characteristics of these networks are given in Table 1.

TABLE 1. General characteristics of the real-world networks with ground-truth communities. N denotes the number of nodes, M is the number of edges, and $|C|$ is the number of real communities.

Network	N	M	$\|C\|$
Dolphins	62	159	2
Football	115	613	12
Karate	34	78	2
Political Blogs	1,222	16,717	2
Political Books	105	441	3

2) ARTIFICIAL BENCHMARK NETWORKS

SimCMR was designed specifically for large-scale community detection in very large and complex networks. In order to show its effectiveness in both quality and time of community detection, we also needed very large networks with known community structure. Thus, we used artificially-generated benchmark networks of different size and complexity using the model known as the LFR benchmark, introduced by Lancichinetti *et al.* [29]. The LFR benchmark has been a defacto standard for testing community detection algorithms because it produces networks with both community sizes and node degrees that follow a Power Law distribution, which is a common characteristics found in real-world networks. The LFR model has several parameters that determine the structural properties of the generated networks as follows. N is the number of nodes, τ_1 is the node degree distribution exponent, τ_2 is the community size distribution exponent, k is the average node degree, *maxk* is the maximum node degree, *minc* is the minimum community size, *maxc* is the maximum community size, and μ is the mixing parameter. Here, the most important parameter is the mixing parameter μ because it determines the fraction of a node's edges that go to other communities than the node's own community. That is, $1-\mu$ fraction of a node's edges stay in the same community as the node's own. According to this definition, as μ increases, the community structure becomes more difficult to detect. We illustrate this effect in Fig. 2.

We generated a total of 288 undirected LFR benchmark networks using the original LFR benchmark generator

TABLE 2. LFR benchmark network parameters.

Param.	Description	Value	Notes
N	Number of nodes	1K, 10K, 100K, 1M	10-fold increase in network size
k	Average node degree	25	Same for all networks
$maxk$	Maximum node degree	$N/10$	To scale with network size
μ	Mixing parameter	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	To make networks with communities of different difficulties
τ_1	Node degree distrib. exponent	-2	Default value
τ_2	Community size distrib. exponent	-1	Default value
$minc$	Minimum community size	50	Same for all networks
$maxc$	Maximum community size	$N/10$	To scale with network size

program provided by its authors. We outline the LFR parameters in Table 2. We frequently see in previous community detection studies that the generated LFR networks are usually very small (with 500 to 5,000 nodes), and their *maxk* and *maxc* parameters are also very close to *k* and *minc* parameters respectively, which make the communities in these networks easily detectable, thus creating no challenge for any community detection method at all. In a work that studied several community detection algorithms and quality metrics from a large-scale perspective, a more challenging set of LFR parameters were selected to generate LFR benchmark networks with $N = 1{,}000$, 10,000, 100,000, and 1,000,000 [30]. This work not only does differ from the previous studies in terms of the network sizes, but also differs by choosing larger values for *maxk* and *maxc* in order to make it much more difficult to detect communities as the number of nodes increases. That is, LFR with $N = 10$K is not simply 10-fold larger than the one with $N = 1$K, instead, its community sizes are distributed between 50 and 1,000 (it is between 50 and 100 for $N = 1$K). As N gets larger, this effect becomes more salient, and communities become more difficult to recover.

To generate the LFR benchmark networks for our research, we followed the same approach to show the performance of SimCMR at scale. Therefore, using the LFR parameters in Table 2, we generated 10 realizations of the LFR benchmarks for all values of the mixing parameter μ for sizes $N = 1$K, 10K, and 100K, thus creating 270 networks. Additionally, we generated only two realizations rather than 10 for each value of μ for $N = 1$M because the long computational time required to generate this large networks even on a good hardware prohibited us. Finally, we generated 288 LFR benchmark networks in total.

B. NETWORK DATASETS WITHOUT GROUND-TRUTH COMMUNITIES

Networks with known community structure provide a convenient way to test the effectiveness of any community detection algorithm. However, it is not always feasible to obtain very large networks with ground-truth communities for experimentation. In order to overcome this difficulty, community detection algorithms are usually tested on artificial benchmark networks like the ones we consider in Section IV-A2. There is an alternative to this approach such that we can test the algorithms on networks without ground-truth communities. In this alternative approach, effectiveness of a community detection algorithm can be assessed by Modularity metric alone that we describe in Section IV-D.

TABLE 3. General characteristics of the real-world networks without ground-truth communities. *N* denotes the number of nodes, *M* is the number of edges.

Network	N	M
DBLP	317,080	1,049,866
Amazon	334,863	925,872
YouTube	1,134,890	2,987,624
Pokec	1,632,803	22,301,964
WikiTalk	2,388,953	4,656,682

We considered the use of five very large real-world networks without ground-truth communities, which are commonly used as benchmarks in network community detection studies. DBLP network is a co-authorship network where two authors are linked if they publish at least one paper together [31]. This network is based on the DBLP computer science bibliography. Amazon network is a co-purchase network of items purchased together from Amazon website [31]. In YouTube network, two YouTube users are linked if they form a friendship with each other [32]. Pokec is a network from the online social network popular in Slovakia [33]. WikiTalk network consists of links between Wikipedia users *i* and *j* if user *i* at least once has edited a talk page of user *j* [34]. General characteristics of these networks are given in Table 3.

C. COMMUNITY DETECTION ALGORITHMS AND SOFTWARE

We compared the performance of SimCMR with the state-of-the-art community detection algorithms in terms of both detection quality and detection time, namely, Louvain, LP, Infomap, OSLOM, and CNM algorithms. In addition, we wanted to compare SimCMR with two very recent algorithms that were also developed to address the large-scale community detection problem, namely, Leiden and CoreExp. Furthermore, we included K-Means (KM) and Bisecting K-Means (BKM) algorithms in our experiments [35]. KM and BKM are very well-known and commonly-used partitioning clustering algorithms preferred in arbitrary data clustering tasks due to their speed and acceptable performance. KM divides the data points into k clusters by minimizing the intra-cluster distances between data points and maximizing the inter-cluster distances between clusters from an optimization perspective. BKM begins with the whole dataset as a single large cluster, and then repeatedly divides the remaining largest cluster into two clusters using KM until k clusters are obtained.

We implemented SimCMR in pure Python on top of NetworkX which is a very popular network analysis framework written purely in Python [36]. We also used the Python

implementations of LP and CNM algorithms found in NetworkX. For Louvain, we used the Python implementation provided by Aynaud [37], which is a very fast and efficient implementation that also uses NetworkX as the underlying graph library. For Infomap, OSLOM, and Leiden, we used the original C++ implementations by their respective authors. For CoreExp, on the other hand, we used the original Java implementation by their authors. It seems quite unfair to compare programs written in C++ and Python, however, we experimentally show that even Python implementations of some algorithms, including SimCMR, can perform as comparably fast as and sometimes faster than C++ implementations of the others. Since they require the number of communities to detect in advance, KM and BKM are not commonly used for network community detection even though there are a large number of studies that utilize them in network analysis domain. For comparison reason, we also implemented both KM and BKM in Python, and included them in our experiments to test their effectiveness when they use the same similarity index we defined between a node and a community.

Considering the reproducibility of our research, we made publicly available the code of SimCMR, the code used for running the experiments and collecting the results, and the experiment datasets in the public repository located at https://github.com/volkantunali/SimCMR. In addition, we included the executable of the LFR benchmark generator and the code to generate all LFR benchmark networks used in our research by using this executable in this repository. Moreover, the executable and the Python interface of OSLOM can be found there. Besides, we included the Java executable of CoreExp in the repository. Since we used C++ implementations of Infomap and Leiden that can be installed via the package installer of Python and that can be accessed via the installed Python package, we did not include any executable file for them in the repository.

D. COMMUNITY DETECTION VALIDITY METRICS

In network community detection research, there are two main validity metrics that are widely used. The first one is Normalized Mutual Information (NMI) metric which has roots from the Shannon entropy of information theory [38]. Assume that A and B are two different community assignments $\{a_i\}$ and $\{b_i\}$ for each node i in the network. We can think that A represents the ground-truth communities (labels) and B represents the community assignment by an algorithm. NMI computes the overlap between A and B in terms of the mutual information between them. Suppose N_b is the number of nodes in community b; N_a is the number of nodes with label a; $N_{b,a}$ is the number of nodes in community b with label a; and N is the total number of nodes in the network. Then, NMI is calculated as in (4). An NMI value of 1 between A and B denotes identical community assignments, whereas a value of 0 denotes totally independent assignments.

$$NMI = \frac{\sum_{b\in B}\sum_{a\in A} N_{b,a}\log\frac{N\cdot N_{b,a}}{N_b\cdot N_a}}{\sqrt{(\sum_{b\in B} N_b\log\frac{N_b}{N})(\sum_{a\in A} N_a\log\frac{N_a}{N})}} \tag{4}$$

The second metric that we used to compare algorithms is Modularity, which is also used by SimCMR as a guiding metric in stage 4. Calculation of modularity does not require the knowledge of ground-truth communities. Instead, it is calculated using only the community assignments of nodes, and it measures the density of edges inside communities as compared to the edges between communities [6], [25]. Modularity can take a value between 0 and 1, and the larger the modularity value, the stronger the community structure is. Consider again B is a community assignment of a network with M edges into k communities; A is the adjacency matrix of the network; v_i and v_j are two nodes with degrees d_i and d_j, respectively. Then, Modularity (denoted by Q) is calculated as in (5).

$$Q = \frac{1}{2M}\left(\sum_{x=1}^{k}\sum_{v_i,v_j\in B_x} A_{ij} - \frac{d_i d_j}{2M}\right) \tag{5}$$

E. HARDWARE AND SOFTWARE ENVIRONMENT

We conducted all experiments on the same HP Z240 Tower Workstation with Intel Core i7-7700 CPU at 3.6 GHz with 4 cores, 32 GB RAM, and MS Windows 10 Pro 64 bit OS. In order to run Python implementations of algorithms, we used Python 3.7 installed with an Anaconda distribution.

F. EXPERIMENTAL PROCEDURE

First of all, we prepared the benchmark network datasets for the experiments. We collected the real-world network datasets mentioned in Section IV-A1 from the network data collection of Newman [28]. Then, we extracted ground-truth community data from them and converted them to a more suitable edge list text format from their original GML format for unified experimentation. Furthermore, we generated the LFR benchmark networks as mentioned in Section IV-A2. In addition, we collected and prepared the large real-world network datasets without ground-truth communities mentioned in Section IV-B from the web sites of their respective authors.

Second, we ran each community detection algorithm 100 times on the real-world networks with ground-truth communities. For the LFR benchmarks, nevertheless, we ran each algorithm 10 times on each different realization of the LFR benchmark networks because the total number of networks were quite large and as the networks became larger and more complex (depending on N and μ) it happened to take extremely long to complete a single run for some algorithms. We did not run CNM, OSLOM, and CoreExp algorithms on the LFR benchmarks with $N = 100$K and $N = 1$M because they were the slowest algorithms and they would not complete in a reasonable amount of time on such large networks. As we did with the LFR benchmarks, we ran each algorithm 10 times on the real-world networks without ground-truth communities because they were very large and complex. Similarly, we did not run CNM, OSLOM, and CoreExp algorithms on them. What is more, we did not run KM and BKM on them because exact number of communities were unknown.

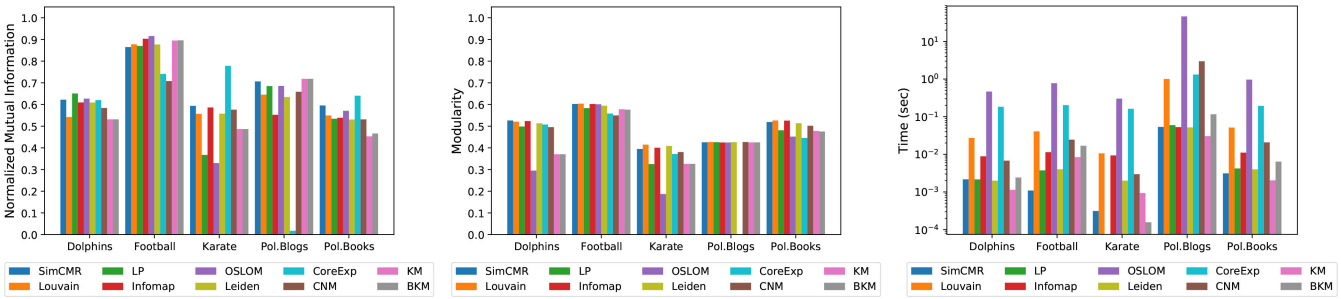

FIGURE 3. Experimental results on real-world networks with ground-truth communities.

In the end, it took several weeks for all experiments to complete. For the sake of the reproducibility of the experimental results, we supplied different random seeds to the algorithms at each run if the algorithm had a random seed parameter, which was the case for Louvain, Infomap, OSLOM, Leiden, KM, and BKM. In this way, we also eliminated the effect of chance on the algorithm performances. For each run of the algorithms, we calculated and recorded NMI (if applicable) and Modularity metric values, as well as the total time of the run in seconds.

V. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we present and discuss our experimental results on real-world networks and LFR benchmark networks. Presented results in the charts are the average of the calculated metrics obtained for each network and for each algorithm. It should be noted that the time axis on the right-hand side charts has logarithmic scale because some algorithms took very long whereas some took very short on the same network. Thus, we needed to use logarithmic scale to clearly present and compare their computation times.

A. ON REAL-WORLD NETWORKS WITH GROUND-TRUTH COMMUNITIES

Results of the experiments on real-world networks with ground truth communities are given in the charts in Fig. 3. According to the results, there is no clear winner. However, for almost all networks, SimCMR exhibits consistently very good performance in terms of NMI, Modularity, and computation time. Network visualizations in Fig. 4 show the communities detected by SimCMR.

First of all, SimCMR clearly outperforms Louvain and Leiden with respect to NMI, and in most cases it obtains Modularity close to or better than Louvain and Leiden, in much less time. In addition, SimCMR is consistently better than LP, and even faster than it in some cases. Moreover, in most cases, SimCMR recovers communities better than Infomap in less time even though Infomap is known to be one of the fastest algorithms and we used its C++ implementation in our experiments. Finally, SimCMR outperforms OSLOM, CNM, and CoreExp in most cases, again in much less time even we used C++ implementation of OSLOM. These results suggest that SimCMR is very effective at finding embedded communities in real-world networks with varying sizes in very short time.

B. ON ARTIFICIAL BENCHMARK NETWORKS

We present our experiment results on LFR benchmark networks in Fig. 5, 6, 7, and 8 for $N = 1$K, 10K, 100K, and 1M, respectively.

Within the LFR benchmarks, networks with $N = 1$K are the easiest ones to partition but this is not due to the size of the networks. Instead, community sizes in these networks are distributed within the range of 50 to 100, which means that there are approximately 15 communities of about the same size in these networks. Especially for the networks with small mixing parameter μ, there was no big challenge for the algorithms in detecting the communities. According to NMI and Modularity results, SimCMR evidently recovers the communities with almost 100% success even for $\mu = 0.6$. When μ is greater than 0.6, SimCMR is better than or close to Louvain and Leiden, and it is clearly better than the others since they begin to fail as μ becomes larger. In terms of computation time, SimCMR clearly outperforms even the fastest algorithms like LP, Infomap, and Leiden when μ is less than 0.5. Even for larger μ values, SimCMR is clearly among the fastest algorithms, keeping its computation time less than 1 second on the average. CoreExp, on the other hand, fails dramatically to recover the community structures for all values of mixing parameter μ. We observed that it almost always generated a single cluster that contained all nodes for the LFR benchmark networks. This behavior was persistent across all LFR benchmarks networks, including $N = 10$K.

Difficulty of finding communities in the LFR benchmark networks becomes more than 10-fold as the size of them grows 10-fold due to the community size distribution and node degree distribution. As a result, LFR benchmarks with $N = 10$K and more were a real challenge for the algorithms. This phenomenon manifests itself in the experimental result charts in Fig. 6, 7, and 8.

For the LFR benchmarks with $N = 10$K, OSLOM outperforms all algorithms even for higher values of μ. Among the other algorithms, Louvain, Leiden, and SimCMR follow OSLOM very closely and consistently according to NMI and Modularity values. Nevertheless, OSLOM requires too much

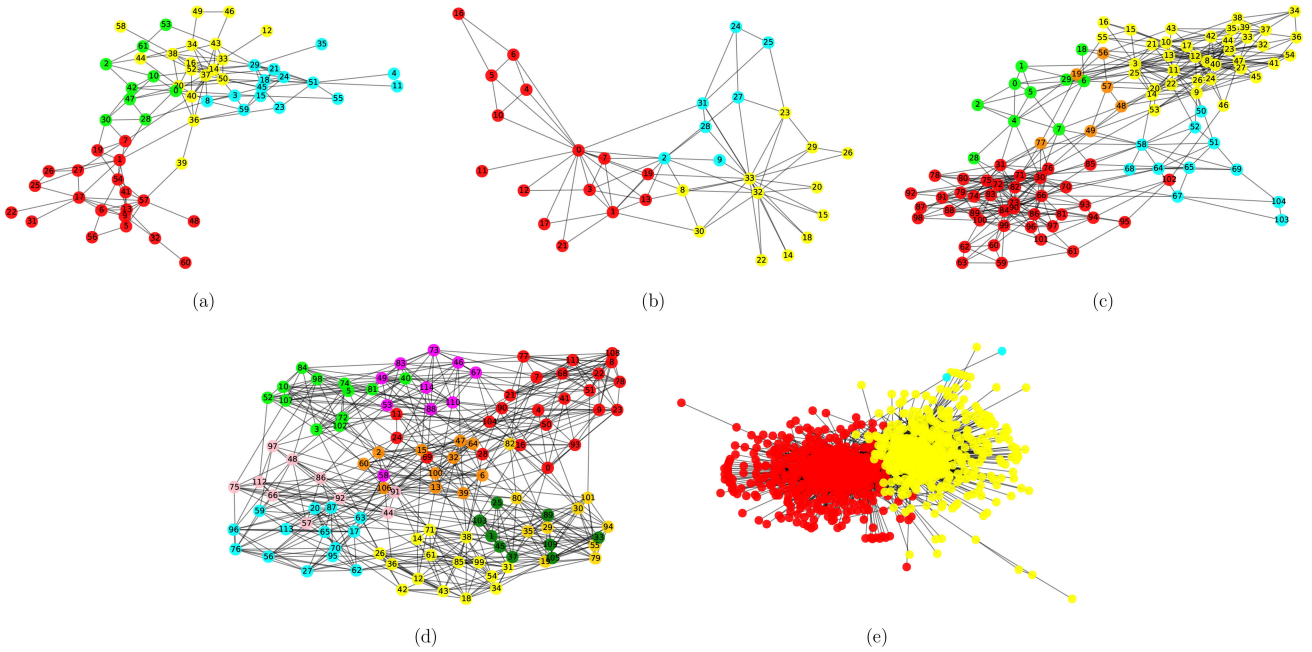

FIGURE 4. Communities of real-world networks with ground-truth communities detected by SimCMR (colors represent community membership of nodes). (a) Dolphins network: SimCMR identified four communities that overlap well with the ground-truth two communities (b) Karate network: SimCMR detected three communities that overlap very well with the ground-truth two communities (c) Political Books network: There are five communities detected by SimCMR with a very good overlap with the ground-truth three networks (d) Football network: SimCMR detected nine communities that overlap well with the ground-truth twelve communities as suggested by the NMI and Modularity scores (e) Political Blogs network: SimCMR detected two ground-truth communities almost exactly while identifying two peripheral nodes as a distinct community.

time to obtain these high scores, and thus, Louvain, Leiden, and SimCMR are the only algorithms that come forward with their good and consistent performances. After all, SimCMR is 5 to 10 times faster than Louvain in general, and it is close to LP, Infomap, and Leiden in terms of computation time, which makes SimCMR a very bright alternative. Infomap, on the other hand, fails dramatically when μ is 0.6 or more. Besides, LP becomes unsuitable even for small μ values like 0.4. CoreExp is again unable to recover any useful community structure even for small values of μ.

For the LFR benchmarks with $N = 100$K, Louvain and Leiden exhibit a very good performance and SimCMR displays close performance to them consistently in terms of NMI and Modularity when μ is less than 0.7. When μ becomes 0.7, SimCMR achieves better NMI but lower Modularity. In terms of computation time, however, SimCMR is again consistently 5 to 10 times faster than Louvain even for higher values of μ. Performance of Infomap falls sharply when μ is 0.5 or more. In addition, LP shows very poor performance even for smaller values of μ.

When we examine the result charts in Fig. 8, we observe a similar pattern as in Fig. 7. LFR benchmarks with $N = 1$M make the most difficult cases in our experiments. We clearly see that SimCMR, Louvain, and Leiden are the only algorithms that consistently attain about the highest NMI and Modularity scores. SimCMR is again about 5 to 10 times faster than Louvain. Infomap is unable to keep its high performance when μ becomes 0.5. Similarly, when μ becomes larger than 0.1, LP performs very poorly.

When we evaluate all experimental results on LFR benchmark networks together, we can clearly conclude that SimCMR, like Louvain and Leiden, consistently perform well even for higher values of μ, where communities are not well-defined. We also observe that communities recovered by SimCMR show high modularity very close to Louvain and Leiden even though SimCMR is partially guided by modularity, and in its essence, it does not try to globally optimize modularity like Louvain and Leiden. Accordingly, we understand that the similarity indices that we have defined are very good at guiding the community merge and refinement processes inside SimCMR. In addition, we see that performance of SimCMR is not affected by network size and it is relatively insensitive to mixing parameter μ. Furthermore, SimCMR achieves its high community detection performance without sacrificing too much computation time even with its Python implementation. Therefore, SimCMR scales very well with network size and complexity. All these properties make SimCMR highly suitable for large-scale community detection in complex networks. There is also room for further improvement. Current Python implementation of SimCMR uses NetworkX graph structure but it does not heavily depend on it. Instead, SimCMR works simply with the node neighborhood data, which makes it extremely easy to implement in any other programming language and with any other graph framework. We naturally expect that if SimCMR is coded in C or C++, for example, it can have around 10-100 times speedup with careful implementation.

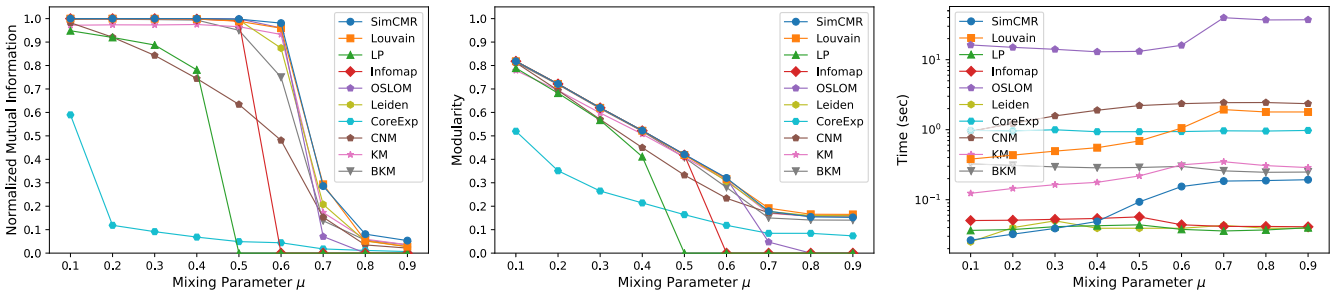

FIGURE 5. Experimental results on LFR benchmark networks with $N = 1K$.

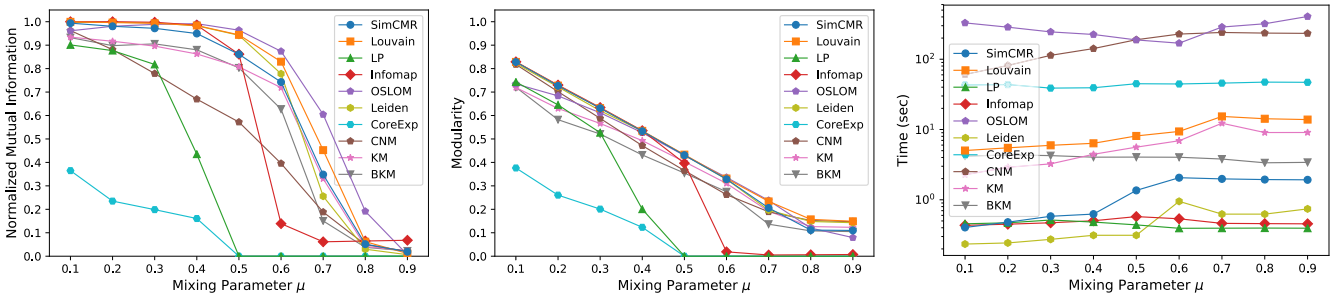

FIGURE 6. Experimental results on LFR benchmark networks with $N = 10K$.

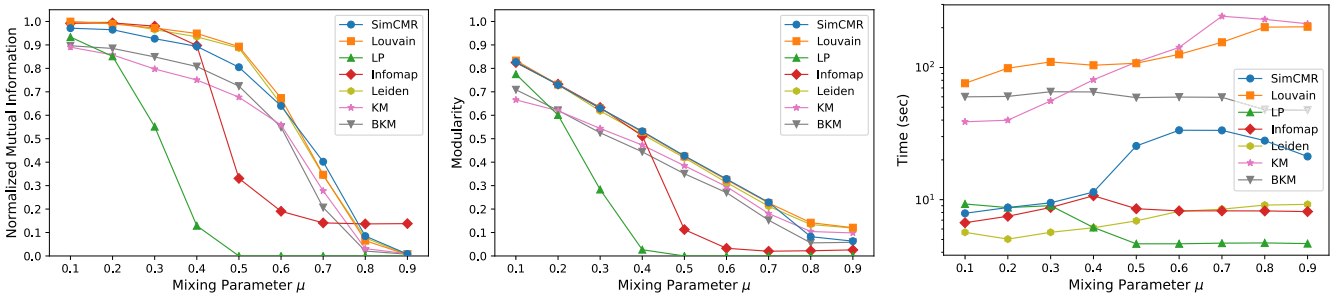

FIGURE 7. Experimental results on LFR benchmark networks with $N = 100K$.

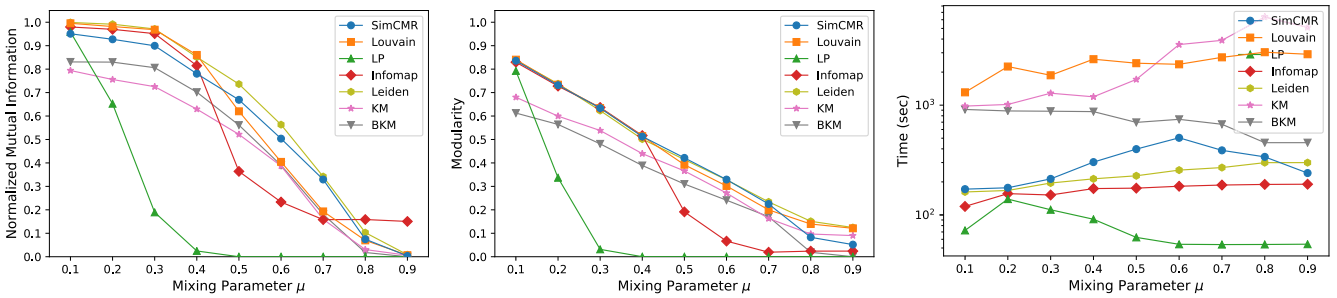

FIGURE 8. Experimental results on LFR benchmark networks with $N = 1M$.

We also need to discuss the performances of KM and BKM algorithms on LFR benchmark networks. KM and BKM are the most widely used algorithms in general clustering tasks as they usually perform well on several types of data including high-dimensional document data [35]. If we have a good and reliable similarity index, it is very straightforward to adapt

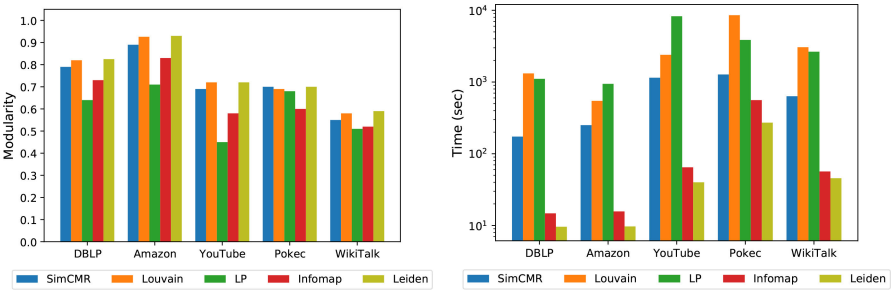

FIGURE 9. Experimental results on large-scale real-world networks without ground-truth communities.

KM and BKM to network data. In our research, we defined a similarity index between a node and a community, and it was our natural expectation like many previous studies that KM and BKM would perform well and fast in community detection. In order to test this, we included KM and BKM in our experiments. According to experimental results, KM and BKM achieves consistently lower NMI and Modularity for almost all LFR benchmarks. However, they scale well with network size and mixing parameter μ, like SimCMR and Louvain. Unfortunately, their computation times are very high, especially of KM. Besides, it makes KM and BKM less attractive to provide the number of communities to detect beforehand. As a result, having a well-defined similarity index only is insufficient to create a good community detection algorithm based on KM. We have one other view that one may find some other heuristics to calculate the similarity between a node and a community, which can be best suitable for KM-based approaches.

C. ON LARGE-SCALE REAL-WORLD NETWORKS WITHOUT GROUND-TRUTH COMMUNITIES

Results of the experiments on large-scale real-world networks without ground-truth communities are given in the charts in Fig. 9. According to the results, SimCMR performs very close to Louvain and Leiden in terms of Modularity, and even better than Louvain for Pokec network. When we compare the computation times of SimCMR and Louvain, on the other hand, SimCMR is consistently faster (about 2 to 10 times) than Louvain. LP mostly obtains lower Modularity in almost all cases despite its very high computation time. Infomap, nevertheless, performs better than LP but worse than both SimCMR, Louvain, and Leiden in terms of Modularity. It is, however, clearly the fastest of all algorithms on these networks after Leiden.

We have one more observation on the behavior of the algorithms with respect to network size. According to the timing results, all five algorithms need more time as the number of nodes increases. When we compare the computation times for particularly YouTube and Pokec networks, which can be regarded as networks of relatively the same size under our consideration with $N = 1.1$M and $N = 1.6$M, respectively, we see a dramatic increase in the computation times of all algorithms except for SimCMR from YouTube network to Pokec network. This is because computational complexity of these algorithm is proportional to the number of edges in the network as well as the number of nodes. Note that number of edges in these two networks are $M = 3$M and $M = 22.3$M, respectively. Computation time of SimCMR, however, mostly proportional to the number of nodes in the network as we mentioned in Section III-C.

VI. CONCLUSION AND FUTURE WORK

Owing to the appearance of very large and complex networks from variety of domains, community detection has been an active research area with the goal of developing methods that can be applicable to large-scale complex networks effectively and efficiently. In this paper, we propose a novel algorithm, SimCMR, that utilizes two novel similarity indices we propose as well. Using the guidance of these similarity indices, SimCMR performs a sequence of candidate community merges and refinements in five stages. Because the computational overhead of these similarity indices is very low and none of the stages considers node-by-node interactions for any underlying process, SimCMR detects communities even in very large and complex networks considerably fast.

We conducted extensive experiments on real-world and artificially-generated benchmark networks to demonstrate and verify the performance and effectiveness of SimCMR comparing it with the state-of-the-art methods. Experimental results reveal that SimCMR scales very well with growing size and complexity of networks. Moreover, SimCMR outperforms most state-of-the-art community detection methods both in detection performance and computation time, which makes SimCMR a very powerful alternative for practical community detection at scale.

There are some known limitations and shortcomings of SimCMR, which may limit its performance and usefulness. Firstly, in the fourth stage of SimCMR, modularity gain is tested to decide if two most similar communities should be merged or not. Instead of basing this decision on modularity, a new heuristic based on similarity can be devised, and therefore, that stage can be made faster eliminating the need for modularity calculation even though it is done very fast in SimCMR. Moreover, we experimentally observe that SimCMR tends to produce very large communities whereas

smaller communities might be more informative and helpful. We think that like many other community detection approaches that primarily utilize modularity notion, SimCMR suffers from the problem known as the resolution limit. In order to overcome this issue, we can seek for alternative strategies for merging small communities.

Overlapping community detection is still an open problem that needs attention and effective methods. In its current form, SimCMR is developed to detect non-overlapping communities. However, a novel method for overlapping community detection can be developed that borrows the main concept of SimCMR utilizing the proposed similarity indices.

REFERENCES

[1] H. Sayama, *Introduction to the Modeling and Analysis of Complex Systems*. Geneseo, NY, USA: Open SUNY Textbooks, 2015.
[2] V. Tunalı, *Sosyal Ağ Analizine Giriş*. Ankara, Turkey: Nobel Akademik Yayıncılık, 2016.
[3] J. Cheng, X. Su, H. Yang, L. Li, J. Zhang, S. Zhao, and X. Chen, "Neighbor similarity based agglomerative method for community detection in networks," *Complexity*, vol. 2019, pp. 1–16, May 2019.
[4] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PLoS ONE*, vol. 6, no. 4, pp. 1–18, 2011.
[5] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 70, no. 6, Dec. 2004, Art. no. 066111.
[6] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 2, Feb. 2004, Art. no. 026113.
[7] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech., Theory Exp.*, vol. 2008, no. 10, Oct. 2008, Art. no. P10008.
[8] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 76, no. 3, Sep. 2007, Art. no. 036106.
[9] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proc. Nat. Acad. Sci. USA*, vol. 105, no. 4, pp. 1118–1123, Jan. 2008.
[10] K. R. Žalik, "Maximal neighbor similarity reveals real communities in networks," *Sci. Rep.*, vol. 5, no. 1, pp. 1–10, Nov. 2016.
[11] Y. Chen, P. Zhao, P. Li, K. Zhang, and J. Zhang, "Finding communities by their centers," *Sci. Rep.*, vol. 6, no. 1, pp. 1–8, Apr. 2016.
[12] R. Shang, W. Zhang, L. Jiao, R. Stolkin, and Y. Xue, "A community integration strategy based on an improved modularity density increment for large-scale networks," *Phys. A, Stat. Mech. Appl.*, vol. 469, pp. 471–485, Mar. 2017.
[13] R. Shang, H. Liu, L. Jiao, and A. M. G. Esfahani, "Community mining using three closely joint techniques based on community mutual membership and refinement strategy," *Appl. Soft Comput.*, vol. 61, pp. 1060–1073, Dec. 2017.
[14] T. Meng, L. Cai, T. He, L. Chen, Z. Deng, W. Ding, and Z. Cao, "A modified distance dynamics model for improvement of community detection," *IEEE Access*, vol. 6, pp. 63934–63947, 2018.
[15] Y. Luo, L. Wang, S. Sun, and C. Xia, "Community detection based on local information and dynamic expansion," *IEEE Access*, vol. 7, pp. 142773–142786, 2019.
[16] J. Cao, Z. Bu, Y. Wang, H. Yang, J. Jiang, and H.-J. Li, "Detecting prosumer-community groups in smart grids from the multiagent perspective," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 8, pp. 1652–1664, Aug. 2019.
[17] Z. Bu, H.-J. Li, C. Zhang, J. Cao, A. Li, and Y. Shi, "Graph K-means based on leader identification, dynamic game, and opinion dynamics," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 7, pp. 1348–1361, Jul. 2020.
[18] F. Hu, J. Liu, L. Li, and J. Liang, "Community detection in complex networks using Node2vec with spectral clustering," *Phys. A, Stat. Mech. Appl.*, vol. 545, May 2020, Art. no. 123633.
[19] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: Guaranteeing well-connected communities," *Sci. Rep.*, vol. 9, no. 1, p. 5233, Dec. 2019.
[20] A. Choumane, A. Awada, and A. Harkous, "Core expansion: A new community detection algorithm based on neighborhood overlap," *Social Netw. Anal. Mining*, vol. 10, no. 1, p. 30, Dec. 2020.
[21] G. Xu, J. Guo, and P. Yang, "TNS-LPA: An improved label propagation algorithm for community detection based on two-level neighbourhood similarity," *IEEE Access*, vol. 9, pp. 23526–23536, 2021.
[22] K.-K. Shang, M. Small, Y. Wang, D. Yin, and S. Li, "A novel metric for community detection," *EPL (Europhys. Lett.)*, vol. 129, no. 6, p. 68002, Apr. 2020.
[23] E. Estrada, *The Structure of Complex Networks: Theory and Applications*. New York, NY, USA: Oxford Univ. Press, 2011.
[24] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behav. Ecol. Sociobiol.*, vol. 54, no. 4, pp. 396–405, Sep. 2003.
[25] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proc. Nat. Acad. Sci. USA*, vol. 99, no. 12, pp. 7821–7826, Apr. 2002.
[26] W. W. Zachary, "An information flow model for conflict and fission in small groups," *J. Anthropolog. Res.*, vol. 33, no. 4, pp. 452–473, Dec. 1977.
[27] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 U.S. Election: Divided they blog," in *Proc. 3rd Int. Workshop Link Discovery (LinkKDD)*, 2005, pp. 36–43.
[28] M. E. J. Newman. Nov. 2019. *Network Data Collection*. [Online]. Available: http://www-personal.umich.edu/~mejn/netdata/
[29] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 78, no. 4, Oct. 2008, Art. no. 046110.
[30] S. Emmons, S. Kobourov, M. Gallant, and K. Börner, "Analysis of network clustering algorithms and cluster quality metrics at scale," *PLoS ONE*, vol. 11, no. 7, Jul. 2016, Art. no. e0159161.
[31] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. ACM SIGKDD Workshop Mining Data Semantics (MDS)*, 2012, pp. 745–754.
[32] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2007, pp. 29–42.
[33] L. Takac and M. Zabovsky, "Data analysis in public social networks," in *Proc. Int. Sci. Conf. Int. Workshop, Present Day Trends Innovations*, Lomza, Poland, 2012, pp. 1–6.
[34] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proc. 19th Int. Conf. World Wide Web (WWW)*, 2010, pp. 641–650.
[35] V. Tunalı, T. T. Bilgin, and A. Y. Çamurcu, "An improved clustering algorithm for text mining: Multi-cluster spherical k-means," *Int. Arab. J. Inf. Technol.*, vol. 13, no. 1, pp. 12–19, Jan. 2016.
[36] D. Zinoviev, *Complex Network Analysis in Python: Recognize—Construct—Visualize—Analyze—Interpret*. Raleigh, NC, USA: Pragmatic Programmers, 2018.
[37] T. Aynaud. Nov. 2019. *Community Detection for NetworkX*. [Online]. Available: https://python-louvain.readthedocs.io/en/latest/index.html
[38] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *J. Stat. Mech., Theory Exp.*, vol. 2005, no. 9, Sep. 2005, Art. no. P09008.



VOLKAN TUNALI (Member, IEEE) was born in Bursa, Turkey, in 1978. He received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in computer and control education from Marmara University, Istanbul, in 2001, 2005, and 2012, respectively. From 2001 to 2012, he worked as a software engineer in a private software company, Istanbul. Since 2012, he has been an Assistant Professor with the Department of Software Engineering, Maltepe University, Istanbul. His research interests include network science, social network analysis, text mining, data science, and software engineering.