STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **8**, March 2020, pp 254–271.
Published online in International Academic Press (www.IAPress.org)

A Consensus Clustering Method for Clustering Social Networks

Masoumeh Kheirkhahzadeh, Morteza Analoui *

Department of Computer Engineering, Iran University of Science and Technology, Iran

Abstract Detecting Communities in networks is one of the appealing fields in computer science. A wide range of methods are proposed for this problem. These methods employ different strategies and optimization functions to detect communities (or clusters). Therefore, it seems a good idea to combine these strategies to take advantage of the strengths of the methods and overcome their problems. This is the idea behind consensus clustering technique which combines several clustering results into one. In this paper, we propose a very good-performing method based on consensus clustering to detect communities of a network. Our method, called *Azar*, employed several community detection methods as base methods. Then *Azar* generates a new compressed network based on the common views of the used base methods and, gives this new compressed network to the last community detection method to find the final partition. We evaluate our approach by employing real and artificial datasets. The implementation results compare the base methods with *Azar* according to accuracy measures such as modularity and Normalized Mutual Information (NMI). The results show the good-performing behavior of *Azar* even for the most difficult networks. The results show the brilliant power of *Azar* in comparison with all the other methods.

Keywords Community detection, Ensemble clustering, Consensus clustering, social networks.

DOI: 10.19139/soic-2310-5070-716

1. Introduction

Discovering clusters or communities in real-world graphs such as online social networks and web graphs is a problem of considerable practical interest that has received a great deal of attention [1, 2, 45, 4]. Community detection is used in particular to understand image segmentation, natural language processing, product-customer segmentation, web page sorting, sociological behavior, protein to protein interactions, gene expressions, recommendation systems, medical prognosis, DNA 3D folding and more [5, 6, 7].

Most recent approaches consider that a network community (also sometimes referred to as a module or cluster) is typically thought of as a group of nodes with more and/or better interactions amongst its members than between its members and the remainder of the network [1, 2, 45, 4, 8]. More formally, a partition $P = \{C_1 , C_2 ,... , C_k\}$ of the vertices of a graph $G = (V, E)$ $(\forall i \in \{1, 2, ..., k\}, C_i \subseteq V)$ (represents a good community structure if the proportion of edges inside the community C_i (internal edges) is high compared to the proportion of edges between them (see for example the definitions given in [9]). Identifying communities in a network can provide valuable information about the structural properties of the network, the interactions among nodes in the communities, and the role of the nodes in each community. Thorough reviews of different types of community detection algorithms can be found in [10, 11, 12, 13].

Several community detection algorithms give promising results when they are tested on networks. Nevertheless, since some methods show serious limitations and biases, and most algorithms are likely to fail in some limit, an ideal method to detect communities in graphs, does not exist [14]. So, it seems a good idea to take advantage of the

*Correspondence to: Morteza Analoui (Email: analoui@iust.ac.ir), Department of Computer Engineering, Iran University of Science and Technology, Iran.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

strengths of different existing methods to find a better solution and avoid the weaknesses of these methods. To reach this goal, some approaches are proposed. One of them is ensemble clustering (also known as consensus clustering), where multiple community detection algorithms run as an ensemble and then the identified communities are combined to improve the communities. Ensemble clustering is also a technique used in some applications such as machine learning and is useful in merging several clustering results into one [15]. In this work, we use ensemble clustering to make a fusion of the results of some famous community detection algorithms (called base algorithms), compress their results by a special projection method [16] and get a consensus on the opinions of these algorithms on a community detection problem. This fusion process decreases the generalization error, because the more predictors differ, the better the performance of the ensemble is [17]. We evaluate our approach on several artificial networks, called LFR networks [18]. The quality of detected communities is measured based on Modularity [19] and, the comparison of the reference communities and the detected ones is done according to NMI values [20].

The following of this paper is organized as follows. Section 2 reviews some related work on ensemble clustering. Our new approach, called *Azar*, and fast-projection method are explained in Section 3. Then, the description of employed base community detection methods and SHRINK are presented in Section 4. The used real and artificial datasets are introduced in Section 5. In Section 6 the evaluation criteria and implementation details are provided. Then the important results of the proposed method are reported in this Section and, finally we conclude the paper in Section 7.

2. Related work

Many community detection methods have been developed recently. These methods have also been improved to handle weighted, directed, and multi graphs. For a comprehensive review of the field as a whole, we refer the readers to reference [21]. Ensemble clustering (also called consensus clustering) is one of the approaches proposed to improve community detection. In Ensemble clustering, multiple community detection algorithms run as an ensemble and the identified communities are combined to improve the community qualities.

An ensemble of clustering methods makes it possible to consider different definitions of a community structure. In addition, more effective algorithms can be found by merging many runs of fast probabilistic algorithms as well as several runs of the same algorithm using different settings. Moreover, the latter method can be used to analyze the community structure of the network at many different scales, providing insight into the relations between community structures at different levels. The integration of consensus clustering with popular existing techniques leads to more accurate partitions than those delivered by the methods alone on LFR benchmark graphs [18]. Interestingly, this holds even for methods whose direct application gives poor results on the same graphs, like modularity optimization [21]. In other words, to identify communities one is often confronted with the problem that one has a large number of potential partitions and often no good way to select a single best partition. Consensus clustering attempts to mitigate the problem by identifying common features of an ensemble of partitions [22].

Constructing a consensus graph is one of the methods of ensemble clustering. A consensus graph is a combination of the partitions returned by base community detection algorithms [23]. The consensus graph G_{cons} is defined over the set of nodes of a network graph G. Two nodes $v_i, v_j \in V$ are linked in G_{cons} if there is at least one partition where both nodes are in the same cluster. Each link (v_i, v_j) is weighted by the frequency of instances that nodes v_i, v_j are placed in the same cluster. The obtained graph is not necessarily a connected one. If there is not a priori information about the relative importance of the original clusters, then a reasonable aim for the ensemble result is to search for a grouping that shares the most information with the original groupings [23]. Different approaches can be applied in order to compute the aggregated clustering out of the consensus graph. In the following paragraphs, some of these approaches are mentioned.

One of the approaches to fuse the different partitions is transforming the consensus graph into a complete one by adding missing links with a null weight [23]. In this work, nodes are finally partitioned into clusters using agglomerative hierarchical clustering with some linkage rule, or by using a classical graph partitioning method such as the Kernighan-Lin algorithm [24]. In another work [25], by maximizing the overlap of the weak input partitions, the authors search for a strong partition. They show that if the process of restarting begin from maximal overlaps of the initial partitions, the quality of the initial weak partitions is not so important and the final result will be good. The point is if the base algorithms do not correspond on the communities of the nodes, the method will not return any ensemble solution. The proposed ensemble clustering method in [26] works on the derived network partitions computed around a seed instead of growing communities around selected seeds. An ensemble ranking method to fuse different local modularity functions, computes the local communities. An iterative agglomerative algorithm expands the seeds. One of the drawbacks of this work is that the networks used for evaluating the method are all small networks. In [27] given a set of r base partitions, the authors compute an $r \times r$ pair-wise clustering similarity matrix M. A similarity graph GV is defined over the set of base partitions by using M. Then the given samples are clustered by a community detection algorithm applied on the similarity graph. A modularity-driven ensemble-based approach to multilayer community detection is proposed in [28] by aggregating the community structures separately generated for each network layer. Using a modified version of modularity with a null model based on the ensemble of partitions for the consensus clustering step is presented in [22]. In this work the authors propose a hierarchical consensus clustering procedure, based on this modified modularity. A genetic algorithm for detecting a community structure in attributed graphs is proposed in [29]. The method optimizes a fitness function that combines node similarity and structural connectivity. The communities obtained by the method are composed by nodes having both similar attributes and high link density. We recently introduced a new ensemble clustering algorithm for social networks (Mitra), which optimizes a special function according to NMI values and compared the algorithm results using several quality metrics [30].

The key distinguishing feature of our ensemble clustering procedure compared to these existing approaches is that we uses different community detection algorithms instead of multiple runs of one algorithm. On the other hand, our approach is not agglomerative or extending method. The next Section explains our ensemble approach with more details.

A notable point is since some of the recent proposed ensemble approaches employ basically different strategies (e.g. using only one community detection algorithm instead of several different algorithms) in comparison with *Azar*, or ensemble approaches work on the other types of networks (such as multilayer networks), or ensemble approaches work based on multiple specifications of network nodes (not only one specification like our work), or designed to reach a dissimilar target function (not optimizing the similarity of communities), comparison of *Azar* with recent similar approaches does not seem fair. Hence, we do not compare *Azar* with other similar ensemble algorithms in our experiments.

3. Our approach

In the following, we define a bipartite network and explain our fast-projection method which is applicable on bipartite networks. Then we present our algorithm.

3.1. Fast-projection method

A bipartite network is a graph whose nodes are divided into two disjoint sets such that no two graph nodes within the same set are adjacent. These node sets are called primary nodes and secondary nodes. For the convenience of directly showing the relation structure among a particular set of nodes, bipartite networks are usually compressed by one-mode projection. The projected network is a unipartite network. The unipartite network contains nodes of only either one of the two sets, and two nodes of this set are connected only if they have at least one common neighboring node in the bipartite network. In this work we consider the real nodes of the original network as primary nodes and communities as secondary nodes in our bipartite network graph.

In fast-projection technique [16], we use random walk to compress information of network and produce a unipartite network. A random walker starts walking on the bipartite network from a random real node and reaches to another real node in each step by passing two consecutive links. Therefore, the walker is forced to pass a community node in each step, which is a common community of the two real nodes of this step. Fast-projection approach is based on the sampling of links. Sampling of important links works well in practice, because most links in a weighted projection carry redundant information for community detection. Therefore only the important and

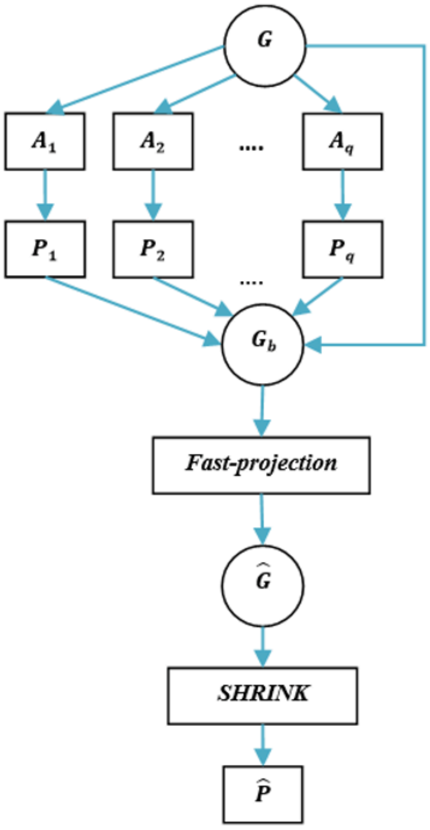

Figure 1. Schematic view of *Azar*.

non-redundant links must be sampled. We apply this method to identify similar nodes which are frequently visited in a sequence by a random walker on a bipartite network.

3.2. Our algorithm

In this Section we explain our approach, called *Azar*, which consists of the following four phases:

1. Create a set $\mathcal{P}$ of q partitions $\{P_1, P_2,\dots, P_q\}$ of graph G with the base community detection algorithms $\{A_1, A_2,\dots, A_q\}$
2. Construct a bipartite graph G_b
3. Apply fast-projection method to G_b and generate $\hat{G}$
4. Find the final partition $\hat{P}$ by employing SHRINK method on $\hat{G}$

Figure 1 depicts the above algorithm. In the first phase, a number of popular community detection algorithms (which are called base methods) are applied to the network graph G to extract node communities. In order to add more diversity to the consensus, and consider different definitions of a community, we selected the base methods such that they employ different strategies to find the communities. Each base method produces a partition P_i of the nodes independently. The process of running the base methods can be done in parallel to decrease the execution time. The base methods set includes Louvain, Infomap, CNM, Walktrap, LPM, MCL, Copra, Oslom and CONCLUDE. We briefly introduce these methods in Section 4.

In the second phase, a bipartite network G_b is generated from the original network G, which includes two types of nodes: the real nodes of G as primary nodes and, the secondary nodes associated with the communities found by the base community detection methods. In other words, for each base method we add a community node for each community found by this method and connect this community node to all the real nodes belong to that community. As a result, at the end of this step, G_b consists of all the real nodes of G and all the community nodes produced by the base methods. The number of community nodes is $\sum_{i=1}^{k} |P_i|$, where $|P_i|$ is the number of communities in P_i, $i \in \{1, 2, ..., k\}$. As a result, the number of all the nodes of G_b is $N_b = N + \sum_{i=1}^{k} |P_i|$ where N is the number of real nodes of G.

In the third phase, G_b is reduced to a unipartite network $\hat{G}$ by fast-projection technique [16]. In fact $\hat{G}$ is a consensus graph. In other words, we keep only the real nodes of G and remove the community nodes. The removal process of community nodes is done in such a manner that the effect of the removed nodes is kept by the weights of the links of $\hat{G}$. Therefore $\hat{G}$ consists of all the nodes of G, but the links of $\hat{G}$ are different from the links of G. In other words, the links of $\hat{G}$ show the compressed result produced by the partitions of all the base methods.

In the last phase the final partition of $\hat{P}$ is found by SHRINK algorithm which is a community detection method not included in base methods set. It is possible to construct $\hat{G}$ in four different ways: it may constructed as an undirected/directed and unweighted/weighted graph. We report the case of undirected and unweighted graph in our implementations. It is notable that some base methods are capable of finding overlapping communities, but we consider the versions which find disjoint communities.

To describe *Azar* more formally, let $G =< V, E >$ be an undirected simple graph where V is the set of nodes and E is the set of edges. We have a set $A = \{A_1, A_2, ..., A_q\}$ of base community detection algorithms which generates a set of different partitions $P = \{P_1, P_2, ..., P_q\}$ defined over the same set V by q algorithms respectively, i.e. P_i be a partition of the set V generated by the base algorithm A_i which is applied on graph G and SHRINK is the algorithm which is applied on the consensus graph $\hat{G}$. We have by definition $P_i = \{C_i^1, C_i^2, ..., C_i^l\}$ where $C_i^j \subseteq V$ and $\forall i, \bigcup_j C_i^j = V$ and $\forall j, q \in [1, l], C_i^j \bigcap C_i^k = \phi$. We consider σ^* as the modularity of consensus graph $\hat{G}$ and σ_s as the modularity produced by SHRINK on G. We explain modularity function formally in Subsection 6.2. The goal of our ensemble clustering function is to compute a consensus clustering $\hat{P}$ in which the modularity of the consensus graph $\hat{G}$ is bigger than the modularity of the original graph G. In a formal way we have (1). The next Section explains the base methods and SHRINK method briefly.

$$\sigma^* > \sigma_s \tag{1}$$

4. Methods

In the following, the base methods set and SHRINK method are introduced.

4.1. Base methods

We have tested some community detection methods as base methods. This set is comprehensive and exploit some of the most interesting ideas and techniques that have been developed for finding network clusters over the last years. The base methods set includes Louvain, Infomap, CNM, Walktrap, LPM, MCL, Copra, Oslom and CONCLUDE. The implementation code for all the algorithms is publicly available. We tried to select the base methods which use different strategies to find a good partition of nodes. These base algorithms use different techniques such as modularity, label propagation, and random walk. Some of these algorithms guide their clustering by employing objective functions and some does not. The following Subsections provide the list of the base methods. Moreover, we explain SHRINK method which is typically a community detection method but we use it in the final step of our approach and separate it from the base methods.

1. CNM (Clauset, Newman and Moore), is a method proposed by Clauset et al. [31]. This method achieves a fast maximization of the modularity [19]. It initially associates each vertex of the network to a community and then, repeatedly merges the communities, the union of which produces the highest elevation of modularity. Code from [32] is used here for the implementations.

2. Walktrap [33], performs random walks to maximize modularity by computing node similarities. Each step of Walktrap defines a partition P_k of the graph into communities, which generates a hierarchical structure of communities. After $n-1$ steps (n is the number of vertices in the input graph), the algorithm finishes and we obtain $P_n = \{V\}$. Code from [34] is used for the case of the best partition found according to modularity.

3. CONCLUDE (COmplex Network CLUster DEtection) [35] produces random, non-backtracking walks of finite length to compute edge centrality of edges. Edge centralities map nodes onto points of a Euclidean space and compute all-pairs distances between nodes; those distances are then employed to group the network nodes into communities. Code from [36] is used.

4. Copra is an extension of the label propagation technique proposed in [41]. Unlike its base technique, Copra is able to detect communities that overlap [37]. In Copra, vertices have labels that propagate between neighboring nodes so that nodes of a community reach a consensus on their community membership. Code from [38] is used.

5. Oslom [39], is based on the local optimization of a fitness function based on the statistical importance of communities with respect to random variations. Oslom consists of looking for significant communities until convergence, analyzing the resulting set of communities and detecting the hierarchical structure of the communities. Code from [40] is used with all default options for unweighted networks.

6. Label propagation method (LPM) [41], simulates the diffusion of labels of nodes. The method assigns a label to each node according to the frequent label of its neighbor nodes. In the first configuration each node has a different label and in the next configurations the process is iterated until all the nodes reach to a label agreement in each community. Code from [42] is used with $p = 5$ for running LPM and $c = 1$ for one time running of the LPM code with default parameters.

7. Markov Cluster Algorithm (MCL) [43] resembles a special flow propagation in a graph. In each iteration two phases called expansion and inflation are performed. Expansion and inflation causes flow to expand within communities and disappear between different communities. After some iterations, the process delivers a stable matrix. The graph described by the matrix is disconnected, and its connected components are the communities of the original graph. Code from [44] is used for the calculations.

8. Infomap [45] is the next base method. The idea behind this method is dividing the network in areas, like countries in a map, and recycling the identifiers of vertices among different areas. This algorithm is a flow-based and information theoretic clustering approach. In this method information flow is simulated by random walk to minimize a map equation over all the network communities to detect its community structure. Code from [46] is used in the implementations.

9. Louvain [5] try to optimize the modularity, by means of a hierarchical approach. The method groups the network into small clusters, to maximize modularity with respect to local moves of the nodes. Then this clusters turn into super-vertices and the network becomes much smaller. This process is iterated until modularity maximization occurs. Code from [47] is used.

4.2. SHRINK

SHRINK [48] is a hierarchical network clustering algorithm. The aim of SHRINK is maximization of modularity. Given a network with n nodes, first one initializes each node with a different community label. In the first phase, the algorithm finds local micro-communities. A micro-community is an isolated node or a sub-graph that consists of one or more connected dense pairs with certain density. According to the groups made by micro-communities, the original network can be reduced and convert to a super-graph. A super-graph is built by shrinking the micro-communities into super-nodes. In the second phase, a super-network is built that consists of super-nodes. The

Variable	Value			Description
N	1,000	5,000	1,000,000	number of nodes in the network
k_{avg}	20	25	50	average degree of nodes
k_{max}	50	100	150	maximum degree of nodes
c_{min}	10	20	20	minimum size of a community
c_{max}	50	100	100	maximum size of a community
μ	{0.1, 0.2, ..., 0.9}			mixing parameter
β	-1			exponent of community size distribution
γ	-2			exponent of degree distribution

Table 1. The parameters used for generating synthetic networks in the implementations.

above two phases are iterated until there is no micro-community with positive modularity gain. Then the hierarchy of communities naturally occurs. The authors kindly sent us the code.

5. Network datasets

To evaluate the performance of our method, two types of datasets are deployed, real and artificial datasets. Zachary [49], Polbooks [50] and Football [51] datasets are real datasets. The datasets comprise 34, 105 and 115 nodes respectively. These datasets are famous and the most of researches in community detection field are tested on these networks.

We also use the state-of-the-art artificial benchmark graphs with built-in community structures, i.e., the LFR benchmark [52]. The main reason for using benchmark graphs for evaluating community detection algorithms is the lack of ground truth information for the communities in the real-world networks. LFR graphs are characterized by power law distributions of vertex degree and community size, features that are frequently observed in real world networks. An LFR graph has a clear community structure. Thus, it serves as a baseline reference for a network with known and detectable structure. LFR networks were created with standard LFR code (see [53]).

One of the parameters of an LFR network is mixing parameter, which is a measure of the degree of fuzziness of the clusters. Mixing parameter μ is the ratio of the external degree of each node (with respect to the node cluster) divided by the total degree of the node, so it varies from 0 to 1. The values of μ which are close to zero correspond to well-separated clusters, whereas the values near 1 indicate a system with very mixed communities (hence hard to identify).

6. Experimental evaluation

6.1. Implementations

First, we generate a total of 270 undirected LFR benchmark graphs using the parameters outlined in Table 1 as artificial dataset. We choose these parameters based on the literature [53]. We generate 10 realizations of the LFR benchmark for each value of the mixing parameter $\mu \in \{0.1, 0.2, ..., 0.9\}$ and for each network size, creating 270 graphs (i.e. 10 samples of each network generated for each pair of N and μ values). All the generated networks are undirected and unweighted. All the reported data points in the diagrams and tables are averaged over these 10 realizations.

Low values of μ correspond to well-separated clusters, which are fairly easy to detect; by increasing the mixing parameter, the communities get more mixed and it is more difficult for clustering algorithms to distinguish them from each other. As a result, we change the value of the mixing parameter μ to study performance as the communities embedded within the graphs become more difficult to recover. We also increase the sizes of the benchmark graphs in order to study the performance of clustering algorithms as networks scale. We test our method against the benchmark graphs which consist of 1000, 5000 and 1,000,000 vertices. Table 1 shows the parameters used for the generated networks. We exclude CONCLUDE and Walktrap from the base methods in the case of the networks with 1,000,000 nodes. As CONCLUDE run time is about some days for this size of networks. Walktrap code also halts in this case. Moreover, Oslom shows unstable results in the implementations in the case of $N = 1000$ nodes for higher μ values. Therefore, Oslom is excluded from the base methods for networks with 1000 nodes.

6.2. *Evaluation criteria*

In the following, we consider two heuristic approaches and compare different community detection algorithms with our approach according to these aspects. First, we are interested in the quality of the communities identified by the algorithms to clarify the behavior of the methods according to modularity [19]. Second, we are interested in the structures of the clusters that various methods are able to find. Basically, we would like to understand how well algorithms perform in terms of optimizing the notion of NMI [20]. These two aspects are explained in the following Subsections.

6.2.1. Modularity. Modularity [19], is one of the most widely used methods to evaluate the quality of modules or communities found in a network. For a given partition of a network into communities, modularity measures the number of internal community edges, relative to a null model of a random graph with the same degree distribution. Although modularity has been shown to have a resolution limit [2], some of the most popular clustering algorithms use it as an objective function [5, 54]. Modularity is given by Equation (2)

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

where the sum runs over all pairs of vertices, A is the adjacency matrix, m is the total number of edges of the graph, and k_i represents the degree of the node i. Modularity yields one if vertices i and j are in the same community $(C_i = C_j)$, zero otherwise. Since the only contributions to the sum come from vertex pairs belonging to the same cluster, we can group these contributions together and rewrite the sum over the vertex pairs as a sum over the clusters [10]:

$$Q = \sum_{S=1}^{k}[\frac{m_S}{m} - (\frac{d_S}{2m})^2] \tag{3}$$

Here, k is the number of clusters, m_S the total number of edges joining vertices of module S and d_S the sum of the degrees of the vertices of S. In Equation (3), the first term of each summand is the fraction of edges of the graph inside the module, whereas the second term represents the expected fraction of edges that would be there if the graph were a random graph with the same expected degree for each vertex. The reader is referred to [55] for a good example of modularity computation.

A higher modularity is often taken as an indication of a better community structure as it is different from the random null model. As such, the modularity can only be used as a comparative measure [25]. It can be shown that modularity optimization, i.e. finding the optimal community structure, is an NP-complete problem [56]. It is also possible to show that modularity has many local maxima [57] making the identification of a global maximum very difficult.

6.2.2. NMI. Testing an algorithm on any graph with built-in community structure implies defining a quantitative criterion to estimate the goodness of the answer given by the algorithm as compared to the real answer that is expected. This can be done by using suitable similarity measures. For reviews of similarity measures see References [**?**, 59].

Here $Normalized\ Mutual\ Information(NMI)$ is selected, borrowed from information theory [20] which have proved to be a reliable measure to assess our approach. To evaluate the Shannon information content [60] of a partition, one starts by considering the community assignments x_i and y_i, where x_i and y_i indicate the cluster labels of vertex i in partition $\mathcal{X}$ and $mathcal(Y)$, respectively. One assumes that the labels x and y are values of two random variables X and Y, with joint distribution $P(x, y) = P(X = x, Y = y) = n_(xy)/n$, which implies that $P(x) = P(X = x) = n_x^X/n$ and $P(y) = P(Y = y) = n_y^Y/n$, where n_x^X, n_y^Y and n_{xy} are the sizes of the clusters labeled by x, y and of their overlap, respectively. The mutual information $I(X, Y)$ of two random variables is defined as

$$I(X,Y) = \sum_x \sum_y P(x,y)\ log \frac{P(x,y)}{P(x)P(y)} \tag{4}$$

The measure $I(X, Y)$ tells how much we learn about X if we know Y, and vice versa. Actually $I(X, Y) = H(X) - H(X|Y)$, where $H(X) = -\sum_x P(x) log P(x)$ is the Shannon entropy of X and $H(X|Y) = -\sum x, y P(x, y) log P(x|y)$ is the conditional entropy of X given Y. The mutual information is not ideal as a similarity measure: in fact, given a partition $\mathcal{X}$, all partitions derived from $\mathcal{X}$ by further partitioning (some of) its clusters would all have the same mutual information with $\mathcal{X}$, even though they could be very different from each other. In this case the mutual information would simply equal the entropy $H(X)$, because the conditional entropy would be systematically zero. To avoid that, Danon et al. adopted the normalized mutual information [20]

$$I_{norm}(\mathcal{X},\mathcal{Y}) = \frac{2I(X,Y)}{H(X)+H(Y)} \tag{5}$$

which equals 1 if the partitions are identical. It has an expected value of 0 if the partitions are independent. The normalized mutual information is currently very often used in tests of community detection algorithms. In our experiments we use max$(H(X), H(Y))$ instead of $(H(X) + H(Y))/2$ in (5), which is a strict version of NMI. Therefore, the following version of NMI is implemented in this work.

$$I_{norm}(\mathcal{X},\mathcal{Y}) = \frac{I(X,Y)}{max(H(X),H(Y))} \tag{6}$$

6.3. Implementation results

6.3.1. Modularity results. Experimental comparisons of all community detection algorithms have been conducted on LFR benchmark graphs. The most important strength point of *Azar* reveals here which is explained in the next paragraph. We compare different algorithms including Louvain, Infomap, CNM, Walktrap, LPM, MCL, Copra, Oslom, CONCLUDE, SHRINK and our *Azar* algorithm and discuss the plots of modularity for these algorithms. “Ground” refers to the reference number of ground-truth communities in the diagrams.

The modularity of the all methods produced on real networks is reported in Table 2. The table shows all the methods generate similar results for the three datasets, especially for Polbooks and Football datasets. Modularity values of all the methods on Zachary dataset, is less than the two other real datasets except for *Azar* method. *Azar* works better for Zachary according to modularity. In the following, we concentrate on the modularity results of all the methods in Figure 2, Figure 3 and Figure 4. These figures consider the modularity measure for N= 1000, 5000 and 1,000,000 nodes respectively. In these figures, greater values represent better community-like groups. We see that modularity rewards the sets of nodes that have many internal edges and a few external, pointing to other clusters. A general observation is that modularity tends to decrease monotonically for all the methods. This is not surprising, since the networks are becoming more complicated gradually and the algorithms are not able to find the true communities for the networks with higher mixing parameters. Moreover, the figures illustrate that for each network size, all the methods have a similar behavior. However, for the more complicated networks, all the methods except for *Azar*, produce diverse results and exhibit dispersion, although all the methods show decreasing trend. In these figures *Azar* produces surprisingly good results. The most important strength point of *Azar* is presented here. *Azar* performance is promisingly better than Shrink and all the base methods according to modularity. The modularity of *Azar* is very close to 1 and never goes under 0.8 in all the three figures. It is observable that although SHRINK produces good modularity for the original network G, its modularity value falls down for the complicated networks. At the same time, when we construct our consensus graph and apply SHRINK after fast-projection on modularity becomes extremely well. The key reason for the superiority of *Azar* over SHRINK is efficient structure

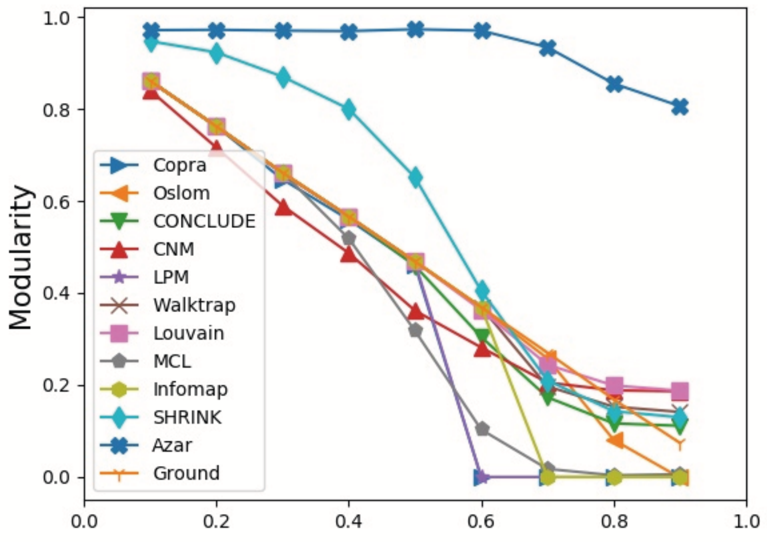

Figure 2. Modularity for all the methods for N = 1000.

of our consensus graph. The consensus graph structure is the abstract compressed result of the original graph which has the most important information of some hidden relations between nodes. SHRINK works interestingly much better on the final network produced by our approach (i.e. $\hat{G}$) as compared to the initial network (i.e. G). This is due to the improved structure of $\hat{G}$, which is the result of our approach and applying our fast-projection method on G_b. As another main point, the difference between modularity of *Azar* and the other methods is increasing as μ increases. This behavior is repeated for all the networks even for networks with 1,000,000 nodes in the case of μ= 0.9 which is the largest and the most complicated network type. The above observations demonstrate the greatly power of *Azar* method in the community detection field. This conclusion confirms our first assumption that the weak points of one base method are covered by the strengths of another. So *Azar* helps in mitigating the limitations of the base and SHRINK methods.

Now we analyze the behavior of the base methods and SHRINK. It is observable that SHRINK is better than all the base algorithms according to modularity. It was predictable because the aim of SHRINK is detecting communities by maximizing modularity with efficient strategies. In the figures we see that the modularity of some methods (like MCL, Copra and LPM) becomes zero earlier in smaller networks. For example, the modularity of LPM becomes zero for μ = 0.6 and μ = 0.7 and μ = 0.9 for networks with N = 1000, 5000 and 1,000,000 respectively. This means that for larger networks, most methods perform better and the modularity values become zero later. As another example, the modularity of Infomap falls down to zero later than MCL, Copra and LPM. The modularity of Infomap never becomes zero for the largest and most complicated networks (see Figure 4).

6.3.2. NMI results. For more investigation of the properties of the methods, the overview of all the NMI values between the ground truth partition and the detected partition of each algorithm is presented. First of all, we analyze the NMI results on real datasets and report them in Table 3. The best method according to average performance on three datasets is LPM and Oslom is the worst method.

Now we analyze the results of all the methods on artificial datasets which are reported in Figure 5, Figure 6 and Figure 7. These figures are associated with 1000, 5000 and 1,000,000 nodes respectively. As pointed before, CONCLUDE and Walktrap are excluded from the consensus in Figure 7. Most of the observations are similar in Figure 5 and Figure 6 and they only differ in some details, but Figure 7 is somehow different. *Azar* is one of the

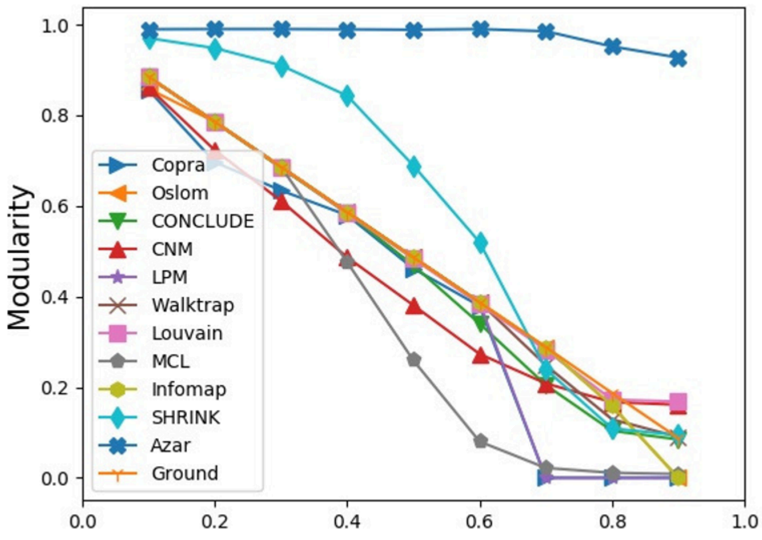

Figure 3. Modularity for all the methods for N = 5000.

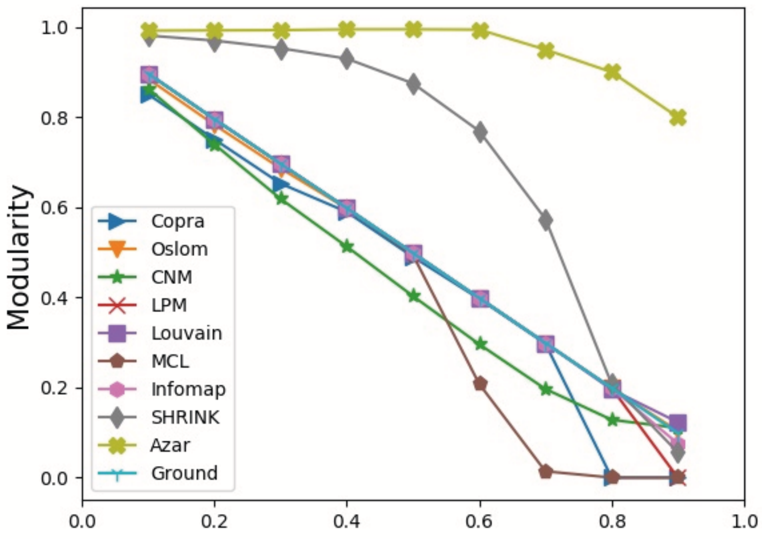

Figure 4. Modularity for all the methods for N = 1,000,000.

Modularity	**Copra**	**Oslom**	**CONCLUDE**	**CNM**	**LPM**	**Walktrap**	**Louvain**	**MCL**	**Infomap**	**SHRINK**	***Azar***
Zachary	0.21	0.36	0.37	0.38	0.4	0.42	0.38	0.36	0.4	0.46	0.52
Polbooks	0.48	0.49	0.43	0.5	0.49	0.52	0.5	0.51	0.53	0.54	0.78
Football	0.60	0.60	0.56	0.58	0.58	0.60	0.58	0.60	0.60	0.78	0.84

Table 2. Modularity of real datasets.

NMI	Copra	Oslom	CONCLUDE	CNM	LPM	Walktrap	Louvain	MCL	Infomap	SHRINK	*Azar*
Zachary	0.32	0.83	0.41	0.58	0.59	0.52	0.45	0.83	0.59	0.57	0.74
Polbooks	0.59	0.56	0.30	0.51	0.57	0.50	0.32	0.46	0.48	0.49	0.46
Football	0.83	0.90	0.87	0.67	0.90	0.86	0.90	0.92	0.92	0.90	0.86

Table 3. NMI of real datasets.

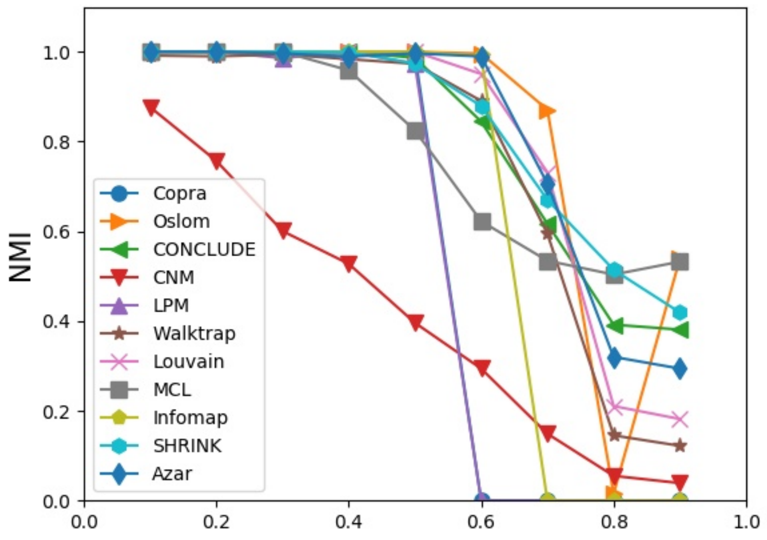

Figure 5. NMI for the base methods and Azar, versus mixing parameter for N = 1000.

best methods in Figure 7 and produces a very good performance for these large networks (N = 1,000,000), even for the most complicated ones.

A general overview of Figure 5 - Figure 7 reveals that all the methods start very well and their NMI values are close to 1 for the lower values of μ, but all the methods show a decreasing trend and approaching zero for the upper values of μ. In Figure 5 this event occurs where μ is 0.6. This decreasing trend was predictable because for the lower values of μ the networks are simpler and almost all the methods can find the correct clusters. By increasing μ, the networks become more complicated and the chance of finding the correct clustering decreases. As illustrated in Figure 5, Figure 6 and Figure 7, most algorithms perform well except for CNM. We can see in Figure 5 and Figure 6, CNM generates an approximately linear curve that is almost the worst case in comparison with the other base methods. Moreover, the performances of Copra and Infomap become zero before all the other base algorithms. On the other hand, some methods are able to find communities even for the most complicated networks and NMI values of these methods never become zero, e.g. CONCLUDE and SHRINK. Figure 5, Figure 6 and Figure 7 also show the performance of MCL and Copra decrease faster than the others. Nevertheless, MCL is performing better than other methods when it comes to greater values of μ (e.g. when μ = 0.8 and 0.9) and NMI of MCL stays above zero for all the networks. However, MCL performance falls down rapidly in Figure 7 and becomes zero in μ = 0.8. SHRINK is one of the best performing methods according to NMI and, works well based on modularity. Also it is interesting that CONCLUDE and SHRINK have a close value of NMI in all the networks of Figure 5 and Figure 6 and close community numbers. In the diagrams we can also see NMI of some methods becomes zero for complicated networks. In these cases, the method is not able to find meaningful clusters and assigns all the nodes to one cluster (e.g. Copra).

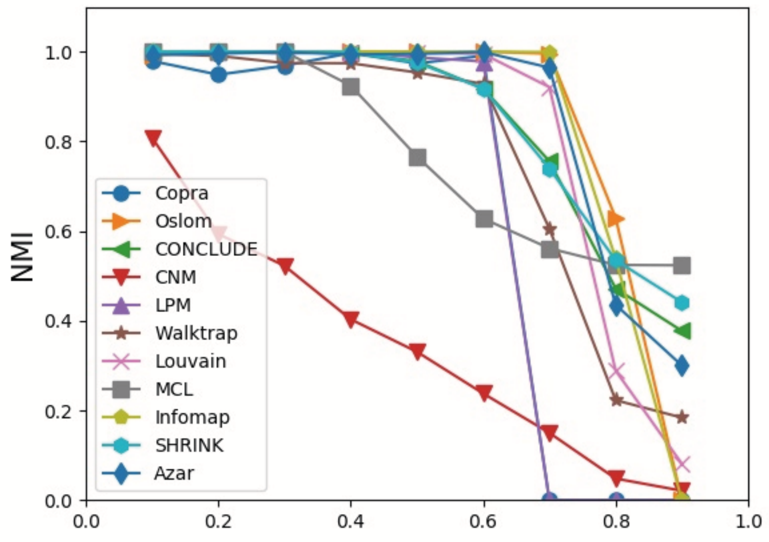

Figure 6. NMI for the base methods and Azar, versus mixing parameter for N = 5000.

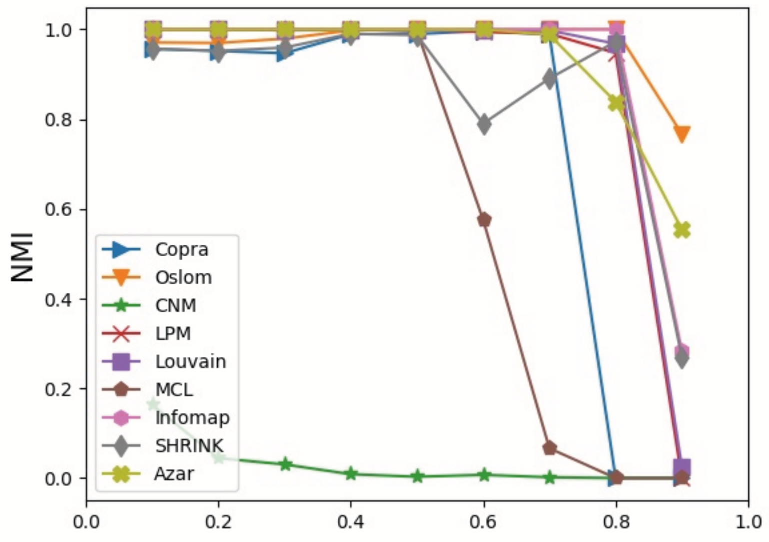

Figure 7. NMI for the base methods and Azar, versus mixing parameter for N = 1,000,000.

The comparison of NMI average values and NMI of *Azar* is provided in Figure 8, Figure 9 and Figure 10. The average is calculated over the NMI values of all the base methods. It is observable in these figures that the average curve is always under *Azar* curve. These observations demonstrate that *Azar* benefits from creating bipartite network and fast-projection and produce more information than a simple averaging method especially when μ is between 0.5 and 0.8 which is observable in Figure 8 and Figure 9. In these figures, the distance between curves is

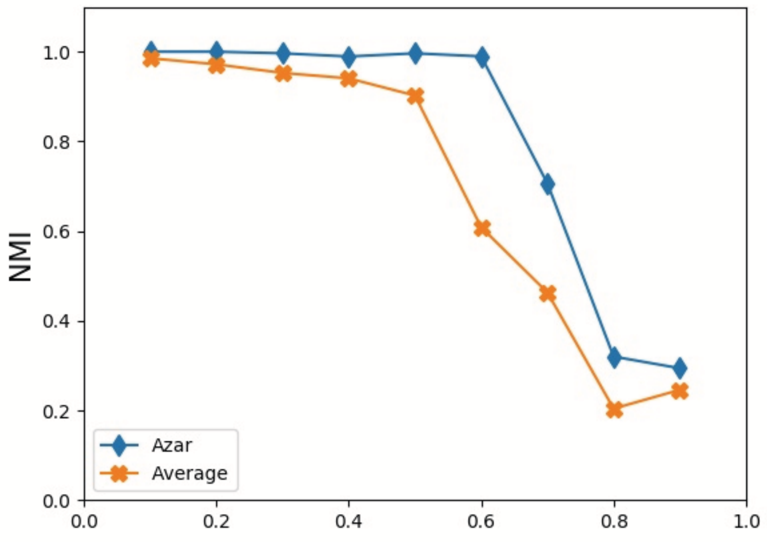

Figure 8. NMI Comparison with average for N = 1000.

0.1 on average. Greater distances are seen for larger networks of Figure 10 in which the average distance becomes 0.27 between two curves. That is the most brilliant result of using *Azar*. In other words, *Azar* is able to find more accurate communities in larger networks and is a scalable method even in the most complicated networks. The results show the superiority of *Azar* especially when μ goes from 0.5 to 0.8. This is very promising. Same or greater distance is seen for larger networks.

In the following, we concentrate on the number of communities which found by all the methods. Some points are distinguished about the number of communities found by different methods. All the methods are able to find the correct number of clusters for simpler networks. It is not surprising and agrees with the reported results of NMI. Infomap and Oslom can find the most accurate number of communities in the three network sizes. Moreover, Infomap produces more accurate results for larger networks. For MCL and SHRINK, the number of communities is generally increasing when the networks are being more complicated. On the other hand, for some other base methods such as LPM and CNM, the number of communities is generally decreasing by increasing μ value in all the networks. The speed of decreasing the number of communities is smoothing for Walktrap and Louvain but this speed is fast for the other methods in the decreasing group. LPM, Infomap and Copra assign all the nodes in one community for complicated networks, which is not informative, however when networks grow in size, this behavior is improved especially for Infomap. CNM has a reverse behavior and approximately put each node in a single community for upper values of mixing parameter such as μ = 0.8 and 0.9. We decided to not include the entire number of communities to summarize the implementation results.

6.3.3. Execution times. The execution times for all the methods are investigated here. The obtained values are reported in Table 4 for real networks and, in Table 5, Table 6 and Table 7 for 1000, 5000 and 1,000,000 network sizes respectively. The time is computed in terms of second. *Azar* run time comprises of three parts, the maximum of the base method run times, the time of building a bipartite network and a consensus graph ,and SHRINK run time. We use only the maximum run time of the base methods in the first part, since it is possible to execute base methods in parallel. LPM is the fastest and, Oslom is the slowest method in base methods set as showed in Table 4.

It is observable in Table 4-Table 7 that *Azar* is the most time consuming method since base methods executions are a part of *Azar* execution. In the case of 1000 and 1,000,000 nodes, CNM is the most time consuming method

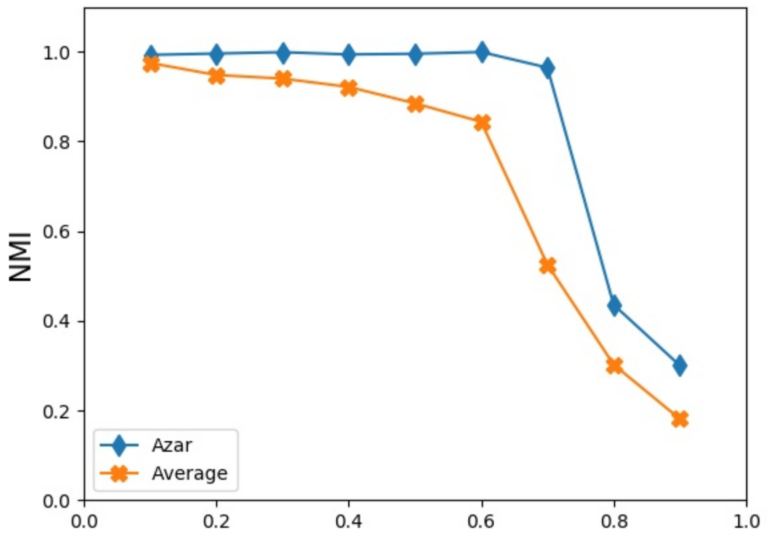

Figure 9. NMI Comparison with average for N = 5000.

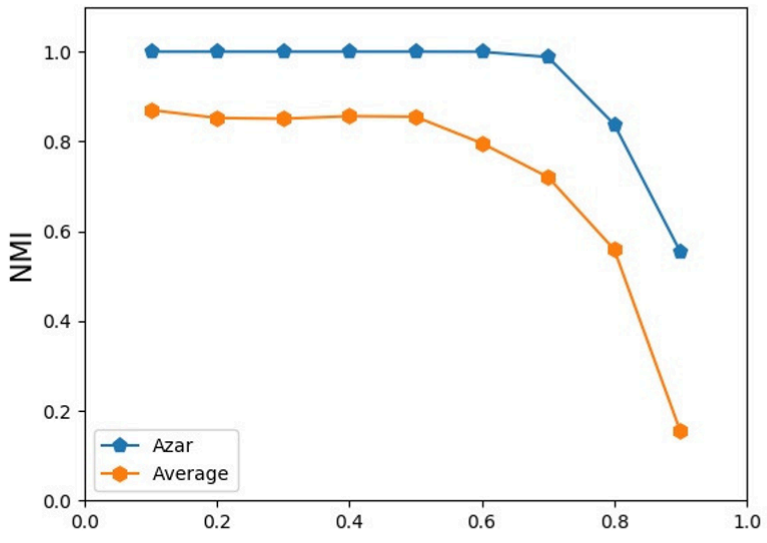

Figure 10. NMI Comparison with average for N = 1,000,000.

between the base methods whereas, for 5000 nodes CONCLUDE consumes the most execution time in the base methods. The tables also show LPM and Walktrap are the fastest methods in all the tables.

Finally, we can conclude about the overall behavior of *Azar*. The method works extremely good in terms of modularity. The modularity of *Azar* is always higher than 0.8, even for the largest and most complicated networks. In the other words, *Azar* modularity is considerably high and is not comparable to the other methods in this work. The results show the brilliant power of *Azar* in comparison with all the other methods. As another point, *Azar* performance is good, in the sense that its performance never becomes zero in terms of NMI and is far better than the

Time	Copra	Oslom	CONCLUDE	CNM	LPM	walktrap	Louvain	MCL	Infomap	SHRINK	Azar
Zachary	0.1	2.15	0.46	0.01	0	0.01	0.01	0.01	0.01	1	8.02
Polbooks	0.16	2.32	0.67	0.01	0	0.01	0.01	0.02	0.01	1.10	8.04
Football	0.19	2.2	0.76	0.01	0	0.01	0.01	0.02	0.01	1.12	8.04

Table 4. Execution times of all the methods for real datasets.

Time	Copra	Oslom	CONCLUDE	CNM	LPM	Walktrap	Louvain	MCL	Infomap	shrink	Azar
0.1	0.72	5.24	15.16	0.11	0.05	0.42	0.07	0.21	0.1	1.9	105.72
0.2	0.66	5.56	15.3	0.12	0.04	0.37	0.04	0.24	0.04	2.92	98.28
0.3	0.7	5.44	17.66	0.16	0.04	0.46	0.05	0.51	0.05	5.24	86.18
0.4	0.67	5.23	19.26	0.19	0.04	0.53	0.05	0.54	0.05	7.4	106.37
0.5	0.69	5.84	17.11	0.21	0.04	0.41	0.05	1.02	0.06	9.01	133.35
0.6	0.74	6.95	17.07	0.24	0.04	0.39	0.05	1.25	0.07	10.64	120.35
0.7	0.73	18.34	15.23	0.25	0.04	0.53	0.07	1.4	0.05	11.73	118.47
0.8	0.69	27.28	14.25	0.24	0.04	0.39	0.07	1.43	0.05	12.37	124.25
0.9	0.7	23.1	14.27	0.25	0.04	0.32	0.06	1.35	0.06	12.5	140.72

Table 5. Execution times of all the methods for N= 1000.

Time	Copra	Oslom	CONCLUDE	CNM	LPM	Walktrap	Louvain	MCL	Infomap	shrink	Azar
0.1	1.53	30.56	377.23	0.97	0.29	7.43	0.3	1.74	0.22	1.9	449.68
0.2	1.35	38.26	402.73	1.92	0.28	7.62	0.32	3.05	0.26	2.92	554.87
0.3	1.47	35.75	459.23	4.24	0.36	3.93	0.34	4.83	0.3	5.24	556.83
0.4	1.85	34.22	437.71	6.2	0.35	5.13	0.41	5.71	0.33	7.4	513.29
0.5	1.5	31.44	448.31	8.01	0.3	2.39	0.34	12.49	0.32	9.01	527.61
0.6	1.8	36	425.08	9.84	0.32	5.28	0.37	12.09	0.39	10.64	581.04
0.7	1.68	55.09	397.82	10.73	0.36	5.55	0.43	16.85	0.68	11.73	490.36
0.8	1.52	129.01	370.83	11.37	0.33	6.71	0.42	17.4	0.51	12.37	551.21
0.9	1.46	148.67	379.69	11.5	0.33	2.55	0.48	13.13	2.85	12.5	590.99

Table 6. Execution times of all the methods for N= 5000.

Time	Copra	Oslom	CNM	LPM	Louvain	MCL	Infomap	SHRINK	Azar
0.1	51.64	3075.68	941.71	15.27	18.58	217.96	12.11	228.96	5726.09
0.2	58.15	9322.97	2992.31	15.75	19.95	364.57	13.86	384.57	10954.29
0.3	67.82	6708.34	5014.77	16.13	21.79	516.71	15.88	548.71	9740.02
0.4	77.48	4093.7	7037.24	16.5	23.63	668.85	17.91	678.85	8525.75
0.5	81.16	5142.61	7796.69	16.93	25.04	906.53	20.51	928.53	9520.05
0.6	59.38	4919.72	9239.33	17.26	27.09	1070.09	27.22	1095.09	15922.8
0.7	109.08	6360.06	9518.57	17.46	30.15	1012.48	33.02	1023.48	12235.11
0.8	114.49	7549.23	9803.84	24.77	35.84	1031.85	42.8	1065.85	11707.42
0.9	58.56	9102.89	9519.93	24.79	43.29	993.36	186.84	1028.36	13802.12

Table 7. Execution times of all the methods for N= 1,000,000.

average performance of the other employed base methods. *Azar* also produces acceptable number of communities for all the networks and these numbers are very close to the reference number of communities. The overall behavior of the base algorithms according to modularity and NMI, shows almost all the algorithms generate similar results for smaller values of μ, although they use different strategies to find clusters. Moreover, all the algorithms show better performances when apply on larger networks according to NMI and modularity especially for *Azar*.

7. Conclusion

In this work, we proposed a new ensemble clustering approach called *Azar* to enhance community detection. We selected a set of diverse clustering algorithms as base methods. The base algorithms use different definitions of communities and incorporate different ideas and techniques to find good clusters. We fused the base method output partitions by generating a bipartite network. The bipartite network was compressed to a unipartite network by fast-projection method. In other words, the output partitions of the base methods are combined to show a more accurate representation of the community structure of the network. As the final step, we used SHRINK method for clustering the unipartite network to detect the final partition.

Investigating the overall results of *Azar* shows the method works extremely good in terms of modularity. The modularity of *Azar* is always higher than 0.8, even for the largest and most complicated networks. In the other words, *Azar* modularity is considerably high and is not comparable to the other methods in this work. The results show the brilliant power of *Azar* in comparison with all the other methods.

As another conclusion, *Azar* performance is good, in the sense that its performance never becomes zero in terms of NMI and is far better than the average performance of the other employed base methods. *Azar* also produces acceptable number of communities for all the networks and these numbers are very close to the real number of communities for the networks.

We are going to investigate the effect of changing the priority of base methods in the quality of ensemble in future works. Moreover, employing more evaluation criteria for quality estimation in community detection methods is our future plan for continuing this work.

References

1. M. E. Newman, Fast algorithm for detecting community structure in networks, Physical review E 69 (6) (2004) 066133.
2. S. Fortunato, M. Barthelemy, Resolution limit in community detection, Proceedings of the National Academy of Sciences 104 (1) (2007) 36–41.
3. M. Rosvall, C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proceedings of the National Academy of Sciences 105 (4) (2008) 1118–1123.
4. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of statistical mechanics: theory and experiment 2008 (10) (2008) P10008.
5. E. Abbe, Community detection and stochastic block models: recent developments, The Journal of Machine Learning Research 18 (1) (2017) 6446–6531.
6. E. Macedo, Two-Step Semidefinite Programming approach to clustering and dimensionality reduction, Statistics, Optimization & Information Computing 3 (3) (2015) 294–311.
7. R. Alguliyev, R. Aliguliyev, L Sukhostat, Anomaly detection in Big data based on clustering, Statistics, Optimization & Information Computing 5 (4) (2017) 325–340.
8. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, Proceedings of the National Academy of Sciences of the United States of America 101 (9) (2004) 2658–2663.
9. S. Fortunato, V. Latora, M. Marchiori, Method to find community structures based on information centrality, Physical review E 70 (5) (2004) 056104.
10. S. Fortunato, Community detection in graphs, Physics reports 486 (3) (2010) 75–174.
11. J. Xie, S. Kelley, B. K. Szymanski, Overlapping community detection in networks: The state-of-the-art and comparative study, ACM Computing Surveys (csur) 45 (4) (2013) 43.
12. S. Harenberg, G. Bello, L. Gjeltema, S. Ranshous, J. Harlalka, R. Seay, K. Padmanabhan, N. Samatova, Community detection in large-scale networks: a survey and empirical evaluation, Wiley Interdisciplinary Reviews: Computational Statistics 6 (6) (2014) 426–439.
13. M. Khatoon, W. A. Banu, A survey on community detection methods in social networks, International Journal of Education and Management Engineering (IJEME) 5 (1) (2015) 8.
14. S. Fortunato, C. Castellano, Community structure in graphs, in: Computational Complexity, Springer, 2012, pp. 490–512.
15. S. Dudoit, J. Fridlyand, Bagging to improve the accuracy of a clustering procedure, Bioinformatics 19 (9) (2003) 1090–1099.
16. M. Kheirkhahzadeh, A. Lancichinetti, M. Rosvall, Efficient community detection of network flows for varying markov times and bipartite networks, Physical Review E 93 (3) (2016) 032309.
17. D. S. Touretzky, Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference, Vol. 8, Mit Press, 1996.
18. A. Lancichinetti, S. Fortunato, Consensus clustering in complex networks, Scientific reports 2.
19. M. E. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical review E 69 (2) (2004) 026113.
20. L. Danon, A. Diaz-Guilera, J. Duch, A. Arenas, Comparing community structure identification, Journal of Statistical Mechanics: Theory and Experiment 2005 (09) (2005) P09008.
21. S. Fortunato, D. Hric, Community detection in networks: A user guide, Physics Reports 659 (2016) 1–44.
22. L. G. Jeub, O. Sporns, S. Fortunato, Multiresolution consensus clustering in networks, Scientific reports 8 (1) (2018) 3259.

23. A. Strehl, J. Ghosh, Cluster ensembles—a knowledge reuse framework for combining multiple partitions, Journal of machine learning research 3 (Dec) (2002) 583–617.
24. M. Newman, Networks: an introduction, Oxford university press, 2010.
25. M. Ovelgönne, A. Geyer-Schulz, An ensemble learning strategy for graph clustering., Graph Partitioning and Graph Clustering 588 (2012) 187.
26. R. Kanawati, Community detection in social networks: the power of ensemble methods, in: Data Science and Advanced Analytics (DSAA), 2014 International Conference on, IEEE, 2014, pp. 46–52.
27. R. Kanawati, Ensemble selection for community detection in complex networks, in: International Conference on Social Computing and Social Media, Springer, 2015, pp. 138–147.
28. A. Tagarelli, A. Amelio, F. Gullo, Ensemble-based community detection in multilayer networks, Data Mining and Knowledge Discovery 31 (5) (2017) 1506–1543.
29. C. Pizzuti, A. Socievole, A genetic algorithm for community detection in attributed graphs, in: International Conference on the Applications of Evolutionary Computation, Springer, 2018, pp. 159–170.
30. M. Kheirkhahzadeh, M. Analoui, Community detection in social networks using consensus clustering, Statistics, Optimization & Information Computing 7 (4) (2019) 864–884.
31. A. Clauset, M. E. Newman, C. Moore, Finding community structure in very large networks, Physical review E 70 (6) (2004) 066111.
32. https://cs.unm.edu/ aaron/research/fastmodularity.htm.
33. P. Pons, M. Latapy, Computing communities in large networks using random walks., J. Graph Algorithms Appl. 10 (2) (2006) 191–218.
34. https://www-complexnetworks.lip6.fr/ latapy/pp/walktrap.html.
35. P. De Meo, E. Ferrara, G. Fiumara, A. Provetti, Mixing local and global information for community detection in large networks, Journal of Computer and System Sciences 80 (1) (2014) 72–87.
36. http://www.emilio.ferrara.name/conclude/.
37. S. Gregory, Finding overlapping communities in networks by label propagation, New Journal of Physics 12 (10) (2010) 103018.
38. https://www.cs.bris.ac.uk/ steve/networks/software/copra.html.
39. A. Lancichinetti, F. Radicchi, J. J. Ramasco, S. Fortunato, Finding statistically significant communities in networks, PloS one 6 (4) (2011) e18961.
40. http://www.oslom.org/software.htm.
41. U. N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Physical review E 76 (3) (2007) 036106.
42. https://sites.google.com/site/andrealancichinetti/software.
43. S. M. Dongen, Graph clustering by flow simulation, 2000.
44. http://micans.org/mcl/.
45. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure, Proceedings of the National Academy of Sciences 105 (4) (2008) 1118–1123
46. http://www.mapequation.org/code.html
47. https://perso.uclouvain.be/vincent.blondel/research/louvain.html.
48. J. Huang, H. Sun, J. Han, H. Deng, Y. Sun,Y. Liu, Shrink: a structural clustering algorithm for detecting hierarchical communities in networks. In: Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 219–228. ACM (2010)
49. W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of anthropological research 33 (4) (1977) 452–473.
50. http://networkrepository.com/polbooks.php.
51. M. Girvan, M. E. Newman, Community structure in social and biological networks, Proceedings of the national academy of sciences 99 (12) (2002) 7821–7826.
52. A. Lancichinetti, S. Fortunato, F. Radicchi, Benchmark graphs for testing community detection algorithms, Physical review E 78 (4) (2008) 046110.
53. https://sites.google.com/site/andrealancichinetti/files.
54. L. Waltman, N. J. van Eck, A smart local moving algorithm for large-scale modularity-based community detection, The European Physical Journal B 86 (11) (2013) 1–14.
55. Emmons, S., Kobourov, S., Gallant, M., Börner, K.: Analysis of network clustering algorithms and cluster quality metrics at scale, PloS one 11 (7) (2016).
56. U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, D. Wagner, Maximizing modularity is hard, arXiv preprint physics/0608255.
57. B. H. Good, Y.-A. de Montjoye, A. Clauset, Performance of modularity maximization in practical contexts, Physical Review E 81 (4) (2010) 046106.
58. Chakraborty, T., Dalmia, A., Mukherjee, A., Ganguly, N.: Metrics for community analysis: A survey. ACM Computing Surveys (CSUR) 50 (4) (2017) 1–37.
59. M. Meilă, Comparing clusteringsłan information based distance, Journal of multivariate analysis 98 (5) (2007) 873–895.
60. D. J. MacKay, Information theory, inference and learning algorithms, Cambridge university press, 2003.