Physica A 506 (2018) 975–986

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Self-adaptive Louvain algorithm: Fast and stable community detection algorithm based on the principle of small probability event

Ziqiao Zhang [a], Peng Pu [b,*], Dingding Han [a,*], Ming Tang [a]

[a] School of Information Science and Technology, East China Normal University, Shanghai, 200241, PR China
[b] School of Computer Science and Software Engineering, East China Normal University, Shanghai, 200241, PR China

HIGHLIGHTS

- We propose a algorithm to improve the Louvain algorithm by using the principle of small probability event.
- Our algorithm is self-adaptive and requires no additional parameters.
- Our algorithm can accelerate the Louvain algorithm without modularity decreasing.
- The performance of our algorithm on networks without distinct community structure is significant.
- We propose equivalent computing time to show the expectation runtime of the algorithm to obtain a relatively good partition, and our algorithm performs the best on this measure.

ARTICLE INFO

Article history:
Received 19 December 2017
Received in revised form 2 March 2018
Available online 3 May 2018

Keywords:
Community structure
Modularity
Louvain algorithm
Small probability event

ABSTRACT

Community structure is an important structure feature of complex networks. Due to its speed, effectiveness and simplicity, the Louvain algorithm is widely used to detect community structure planted in the network topology. Speeding up the Louvain algorithm, enabling the analysis of larger graphs in a shorter time and maintaining the accuracy of result, can benefit the research of networks in many fields. We here propose the Random Self-adaptive Neighbors Louvain algorithm as a new improved Louvain algorithm. The principle of small probability event is used to infer the number of neighbors to pick up randomly. The accuracy, speed and fluctuation of our method are compared with those of the original Louvain algorithm and the Random Neighbor Louvain algorithm. The results show that the RSNL can obtain as good partition as that of the original Louvain in a faster speed. On the networks without distinct community structures, the RSNL is faster and more accurate than the RNL. A new measure, equivalent computing time, is proposed to show the expectation runtime of the algorithm to obtain a relatively good partition. The comparison of this measure shows that the RSNL algorithm can make the best performance among the three algorithms in most cases.

* Corresponding authors.
E-mail addresses: 51151214019@stu.ecnu.edu.cn (Z. Zhang), ppu@cc.ecnu.edu.cn (P. Pu), ddhan@ee.ecnu.edu.cn (D. Han), mtang@ce.ecnu.edu.cn (M. Tang).

https://doi.org/10.1016/j.physa.2018.04.036
0378-4371/

1. Introduction

Complex networks have gained increasing attention in the past decade [1], as this description mode can compactly represent internal structures of a wide range of complex systems [2–4] from biology [5] to social media [6] or web analysis [7,8]. Individuals in those complex systems and their interactions can be abstracted as nodes and links of networks. By analyzing the complex networks, a structural property feature called community structure was found to be ubiquitous in the real world. The most acknowledged description of a community structure is that the communities are groups of nodes within which the network connections are dense, but between which they are sparser [9,10].

The community structures planted in complex networks are strongly related to the dynamical and functional properties of the networks [11,12]. In protein–protein interaction networks, communities are groups of proteins having the same specific function within the cell [13–15]. In the network of World Wide Web, communities represent groups of pages sharing the same or related topics [16,17]. In social networks, they may correspond to groups of individuals who have the same hobbies or consanguinity [18–21]. Thus, analyzing community structure is a vital method to understand structures and functions of complex networks.

Community detection is an essential step in analyzing community structure. Its aim is to identify the clusters by only using the information encoded in the network topology [9]. This field has gained attention for many years and various methods have been proposed to identify the embed modules in complex networks. The Kernighan–Lin algorithm is one of the earliest and still frequently used methods to partition a graph into two clusters of the same size [22]. Fiedler used the eigenvector of the second smallest eigenvalue of the Laplacian matrix to obtain a bipartition of the network [23]. Arenas, et al. used the synchronization theory to find communities in graphs [24]. Raghavan, et al. proposed an algorithm called Label Propagation Algorithm to detect the community structure in near linear time [25].

Among the varied algorithms of community detection, the modularity optimization algorithms are widely used. These algorithms use the best known quality function, Newman–Girvan modularity Q, which is originally introduced to define a stopping criterion for the algorithm of Girvan and Newman [26]. According to the hypothesis, high values of modularity indicate good partitions [9]. The modularity optimization algorithms therefore change the problem from finding the ground truth of the planted community structure to optimizing the quality function. A mass of optimization algorithms have been developed for optimizing the modularity Q. For example, Newman introduced a greedy algorithm [27], J.M. Pujol, et al. and H. Du, et al. improved the greedy optimization algorithm by starting from reasonable intermediate configurations [28,29]; Guimerà, et al. proposed the simulated annealing algorithm [30]. One of the fastest and most effective algorithms is the Louvain algorithm [31,32]. Its time complexity is believed to be $\mathcal{O}(m)$, which means that it is linear with the number of edges.

The Louvain algorithm is fast and effective, but it still has some shortcomings [9,33]. Santo Fortunato [9] pointed out that the results of the Louvain algorithm dramatically depend on the order of the sequential sweep over the vertices, which is generated randomly. It means that the algorithm may obtain different results when changing the sequential sweeping order of the nodes. Ozaki et al. [33] showed that the original algorithm spent a lot of computational time in estimating the nodes that would not be moved. Some improved algorithms for the Louvain algorithm have been proposed either to speed up the algorithm [32–34] or to diminish the influence of the randomness within the algorithm [35]. V.A. Traag [32] proposed the Random Neighbor Louvain algorithm (RNL for short) to accelerate the Louvain algorithm. This algorithm randomly picks up only one neighbor of each node, instead of all of the neighbors, to calculate the increment of modularity. The results in Ref. [32] showed that the RNL can speed up the Louvain algorithm roughly 2–3 times with only slightly decline of modularity. However, the neighbor of a node i which is chosen randomly by the RNL to calculate the increment of modularity is not determinate, so that the algorithm cannot make sure to move the node i into its most appropriate community. Thus, the modularity of the result may decrease a lot in some cases and the fluctuation of the result caused by the influence of the randomness may be increased. We here propose to randomly pick up several neighbors of node i to calculate ΔQ, neither only one neighbor nor all of the neighbors, to make sure that the neighbor which indicates the most appropriate community of node i can be chosen. The results of the experiments conducted on synthetic benchmarks and empirical networks show that our Random Self-adaptive Neighbors Louvain algorithm has the ability to maintain the modularity of the result with decreased runtime, and the fluctuation of the results obtained by the RSNL algorithm is lower than that of the RNL. Especially in the networks without distinct community structures embedded, the new algorithm is even faster than the RNL.

In this paper, in Section 2, we first illustrate the original Louvain algorithm and the shortcoming of the Random Neighbor Louvain algorithm. In Section 3, our new algorithm, the Random Self-adaptive Neighbors Louvain algorithm, is proposed for accelerating the Louvain algorithm and parameters of the new algorithm are inferred and estimated. Then, in Section 4, the RSNL is tested on artificial benchmarks and empirical networks. Comparisons among the original Louvain algorithm, the RNL and our RSNL are also made.

2. Related work

The Louvain algorithm can be described as following:

(a) Each node is located in an isolated community initially.
(b) At the start of each iteration, a node sweeping sequence is generated randomly and check each node according to the sequence.

(c) Given a node i, for each neighbor j of i, calculate the gain of Q, i.e., the increment of modularity coming from putting node i into the community of its neighbor j.
(d) Find the neighbor j' that yields the largest modularity increment. If $\Delta Q_{max} > 0, c_i = c_{j'}$, move the node i to the community of node j'. Else, the node i will not be moved.
(e) When reaching the end of the node sequence, if there is at least one node being moved, the algorithm will jump back to step (b) for another iteration.
(f) Generate a new network, where the nodes in the new network, called supervertices [9], represent the communities in the original network. And two supervertices are connected if there is at least an edge between nodes of the corresponding communities. The weight of the edge between the supervertices is the sum of the weights of the edges between the represented communities in the original network.
(g) The algorithm jump back to step (a), reiterate the procedure until there is no more nodes to be moved.

The Louvain algorithm consists of two main phases: modularity optimization [step (a) to step (e)] and community aggregation [step (f)]. The two phases constitute one pass. In the modularity optimization phase, the algorithm is trying to find the optimum of the modularity in the current scale by moving nodes into the communities of their neighbors that yield the largest modularity increment. If all of the nodes cannot be moved, which means that the algorithm has reached the optimal solution in the current scale, the modularity optimization phase will end. In the community aggregation phase a new network, which consists of supervertices, is formed and the algorithm will then jump back to the modularity optimization phase to find the optimal community structure in this higher level. The two phases will be repeated until there is not any move of node.

Due to the existence of the community aggregation phase, which will dramatically decrease the number of nodes of the network in operation, the most time-costing part of the algorithm is the first pass, where the size of the network in operation is the largest. And among the steps of the modularity optimization phase, the most time-costing procedure is the calculation of ΔQ. Blondel et al. proposed a ΔQ calculating method based on the local information to diminish the calculating time [31]. It makes it possible to calculate the variation of modularity without calculating on the whole network. However, V.A. Traag found that it spends roughly 95% of the time calculating the ΔQ in their implementation [32]. Fig. 1 shows how the runtime increases with the ΔQ calculating times. It is almost linear. So, to decrease the computational time of the algorithm, reducing the times of ΔQ calculating is a definitive factor.

To decrease the ΔQ calculating times, picking up less neighbors is the most direct way. V.A. Traag [32] proposed the Random Neighbor Louvain Algorithm to decrease the ΔQ calculating times. In this algorithm, not all of the neighbors of node i is selected to calculate ΔQ. Only one neighbor node is chosen randomly to calculate ΔQ and to estimate whether node i can be moved to. The result in the paper shows that the improved algorithm can accelerate the speed dramatically with slightly decline of modularity.

However, when the algorithm randomly picks up only one neighbor of node i, the chosen node is not determinate. The algorithm cannot make sure that the community of the chosen neighbor is the most appropriate community in which the node i is ought to be. This may lead to the instability of the performance. The fluctuation of the results may be increased due to the randomness of choosing one neighbor. In addition, the RNL requires an assumption that a random neighbor is likely to be in a good community [34]. This assumption, which is the foundation of the idea of the RNL algorithm, may not be true in some networks, so that the RNL will not be applicable in that cases.

Picking up less neighbors is a useful method to decrease the ΔQ calculating times to accelerate the Louvain algorithm, but picking up only one neighbor may be unreasonable. We here propose a new algorithm: the Random Self-adaptive Neighbors Louvain algorithm. In this algorithm, C_i neighbors of node i will be chosen randomly and these C_i chosen neighbors should contain at least one node which represents the most appropriate community of node i, so that the algorithm can put the node i into its most appropriate community by calculating the increment of modularity of these C_i neighbors. To achieve this goal, the number C_i is derived by using the principle of small probability event.

3. Methods

3.1. Correct nodes

Before we start to discuss our new algorithm, we first define the concept of correct nodes. As we know, the ground-truth is the partition that is planted in the network. It is considered to be the truth which each community detection algorithm is trying to discover. For the modularity based algorithm, there is an assumption that a partition with high modularity indicates a good partition, and the ground-truth indicates the partition with the highest modularity. This is the basic thought and assumption of the modularity optimization algorithms. Although it is not always true in some cases (we will discuss it later), our new method is still one of the modularity optimization algorithms. So that we continue using this assumption.

In the ground truth, each node is located in a community. Among the neighbors of node i, there ought to be some nodes located in the same community with node i. We define these neighboring nodes as correct nodes. The correct nodes represent the community of the node i in the ground-truth, and they also represent the target community of movement of node i to reach the maximal of the modularity in the algorithm.

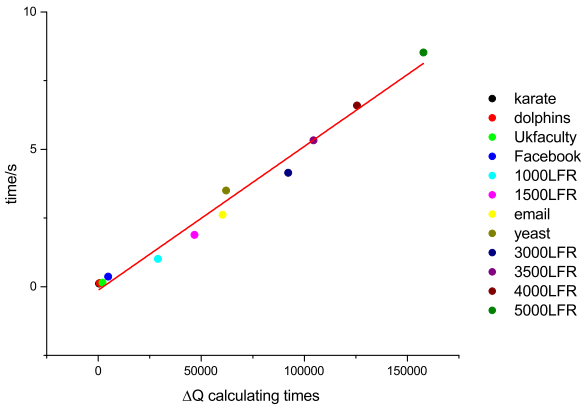

Fig. 1. The relationship between the computing time and the ΔQ calculating times in the original Louvain algorithm.

3.2. Derivation of C_i

C_i, the number of the chosen neighbors of node i, is the key factor of the idea of our algorithm. The correct nodes represent the most appropriate community that node i should be moved in. Therefore the C_i randomly chosen neighbors of node i must contain at least one correct node, so that the algorithm can move the node i into its correct community by calculating the ΔQ of the C_i neighbors and finding out the node with maximal ΔQ. The runtime of the algorithm depends on the ΔQ calculating times. To decrease the runtime, C_i should be as small as possible. Thus C_i is supposed to be the minimum number which can ensure that at least one of the correct nodes can be contained in the chosen neighbors. It is not reasonable to use an arbitrary fixed number as the number of the chosen neighbors, because we cannot make sure that the correct nodes are contained in these chosen nodes. For example, for the nodes with large degree, the uncertainty of the chosen neighbors will not be different when randomly choosing two or three neighbors instead of only one neighbor. So, the degree of a node i should be considered in the derivation of C_i.

As we have defined the correct node before, we can assume that the proportion of the correct nodes in the neighboring set of node i is $p_i = \frac{n^i_{correct}}{n_i}$, where $n^i_{correct}$ is the number of the correct nodes of node i and n_i is the number of neighbors of node i. Here we use n_i instead of node degree k_i due to self edges in a network, which is common after the community aggregation procedure in Louvain algorithm, the k_i indicates both the edges that link to the neighbors and the edges that link back to node i itself. When finding the target community that node i could be moved to, the self edges are obviously redundant. So our algorithm uses n_i to preclude the self edges. The number of the wrong nodes of node i, i.e. the neighbors that are not located in the same community with node i in the ground-truth, is $n^i_{wrong} = (1 - p_i) * n_i$. Obviously, as long as we pick up all of the wrong nodes and another one, a correct node is certain to be chosen. The maximum of C_i is:

$$C^i_{max} = (1 - p_i) * n_i + 1. \tag{1}$$

When randomly choosing C_i nodes from n_i neighbors without replacement, the probability of the event A that all of the C_i nodes are wrong nodes is:

$$P(A) = \frac{\binom{(1-p_i)*n_i}{C_i}}{\binom{n_i}{C_i}}. \tag{2}$$

In statistics theory, if the probability of an event is very small, it can be regarded as a small probability event [36]. And a small probability event will hardly occur in a simple experiment. So, if we only consider the experiment of sampling nodes from the neighbors of a node i, the event that the correct node is not included in the C_i chosen nodes will not occur if Eq. (2) is less than a threshold. Then, we can consider that the correct node must be included in the C_i chosen nodes. In the field of small probability event, there are two widely accepted probability threshold, 0.01 and 0.05. To determine which value should be used and to verify the reliability in community detection algorithm, we have tested our algorithm with different thresholds (ranging from 0.01 to 0.2) on networks. The result shows that a larger threshold will make the algorithm faster, but the modularity of the partition will be smaller. Thus, to make a tradeoff, we find that 0.05 is a proper value of the probability threshold here.

When n_i is small, it may happen that even if the algorithm choose all of the wrong nodes [i.e., $C_i = n^i_{wrong}$], the probability $P(A)$ still cannot be less than 0.05. In this case, the C^i_{max} could be the best solution of this problem. And when n_i is large, the sampling without replacement can be approximated by sampling with replacement. Thus, the number of neighbors to choose is given by

$$C^i_s = \log_{(1-p_i)} 0.05. \tag{3}$$

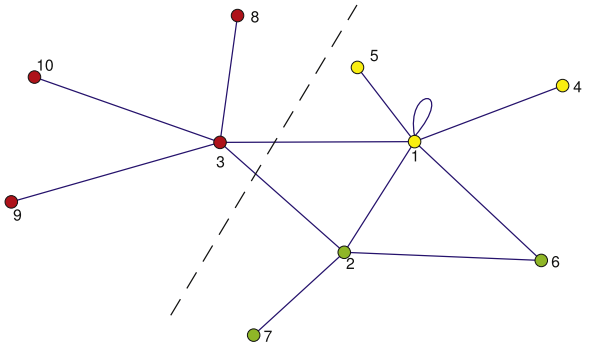

Fig. 2. The demonstration of the measurements in our Random Self-adaptive Neighbors Louvain Algorithm. The clusters separated by the dash line are the planted partition of the network, i.e. the ground-truth, and the color of the nodes shows the current partition exposed by the algorithm. In this case, for node $i = 1$, it has 5 neighboring nodes, the $n_1 = 5$. In the ground-truth, there are 4 neighbors located in the same community with node 1 (node 2, 4, 5 and 6), thus the $p_1 = 0.8$. And in the current partition, there are only 2 neighbors located in the same community with node 1 (node 4 and 5), so the $p'_1 = 0.4$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

These two numbers, C^i_{max} and C^i_s, can both ensure that at least one of the correct nodes is contained in the chosen neighbors. As we want C_i to be as small as possible, the solution of this problem is

$$C_i = \min(C^i_s, C^i_{max}). \tag{4}$$

With the proceeding of the algorithm, a node i will be located in a community with several neighbors located in the same one. These neighbors located in the same community with node i in the current partition is denoted by p'_i. In this case, the algorithm does not need to choose nodes from all of the neighbors of node i. When the Louvain algorithm moves a node i, it takes out i from its current community and places it into the other one. So, when the algorithm is manipulating a node i to check whether it can be moved, there is an assumption that the node i should not be located in its current community. It means that all of the neighbors which are located in the same community with node i in the current partition are not correct nodes, and all of the correct nodes are among the other neighbors. What the algorithm will do is to randomly choose C_i nodes from the other neighbors, expressed as $(1 - p'_i) * n_i$, and $p_i * n_i$ of them are correct nodes. So, the expression of C^i_{max} and C^i_s have to be modified as:

$$C^i_{max} = (1 - p'_i) * n_i - p_i * n_i + 1 \tag{5}$$

$$C^i_s = \log_{(1-\frac{p_i}{(1-pi')})} 0.05. \tag{6}$$

Fig. 2 is the diagram of n_i, p_i *and* p'_i. In the inference of C_i, p'_i is easy to obtain, but p_i, the proportion of the correct nodes among the neighbors of node i, is hard to know because when detecting the community structure of a given network, the ground truth of the structure is not a priori. So, we have to estimate the p_i roughly.

3.3. Estimation of p_i

The p_i of each node are not necessarily equal. According to the inference of C_i, for each node i, its p_i should be estimated to calculate the number of the chosen neighbors. This will be time costing. In our algorithm, to simplify the calculation, we use a uniform value to calculate C_i for each node, instead of distinct value of p_i. This value should be $\min p_i$, because the C_i calculated by using $\min p_i$ is just slightly larger than the C_i calculated by using the exact value of p_i. Thus, the algorithm can make sure that at least one of the correct nodes is contained in the C_i chosen neighbors as well as just performs slightly more times of ΔQ calculating procedure. In fact, it is still hard to calculate $\min p_i$, so we have to estimate its value. The estimation is denoted by p_{est}. Whenever we manipulate one node i, we can calculate the proportion of its neighbors located in the same community with node i in the current partition, i.e. p'_i. If node i is moved, it means that i is considered as not being located in its correct community. And according to the tendentiousness of node movement (see SI. for detail), there is $p_i > p'_i$. And on the contrary, $p_i \leq p'_i$ when node i is not moved. Thus, we have

$$\begin{cases} p'_i < p_i, & i \textit{ is moved} \\ p'_i \geq p_i, & i \textit{ is not moved}. \end{cases} \tag{7}$$

Here we propose to use the $\min(p'_i \mid i \textit{ is not moved})$ to be the estimation of $\min p_i$. The $\min(p'_i \mid i \textit{ is not moved})$ is very easy to be obtained so that it will not involve more time costing procedure. And the experiments in Section 4 will show that the error is not prominent. Thus, we have

$$p_{est} = \min(p'_i \mid i \textit{ is not moved}). \tag{8}$$

The information of the network that we use to obtain p_{est} is gathered gradually from the intermediate result that whether a node can be moved. With the partition process running, the algorithm can gather more information about the structure of the network, and thus the estimation will be more accurate. So, the estimation p_{est} should be updated during the processing of the algorithm. At the end of each iteration, i.e. when the algorithm reaching the end of a node sweeping sequence, the p_{est} will be updated by calculating $\min(p'_i$ of the nodes which are not moved in the latest iteration. And in the next iteration, C_i should be derived by the up to date p_{est}.

As we mentioned before, when the algorithm is checking whether a node i can be moved, there is an assumption that the neighbors located in the same community with node i in the current partition are not correct nodes. Thus, in the case that for a node i, $1 - p'_i < p_{est}$, even though all of the neighbors which are located in the different communities of node i in the current partition are correct nodes, the proportion of the correct nodes in that case still cannot reach the p_{est}, which is considered as the minimum of the proportion of the correct nodes. So, it is reasonable to consider this node i as the node that have already been located in the correct community, and the algorithm does not need to check it. In addition, we know that the node will move to the community with p'_i increasing. Just like the discussion above, if for a node i, $p'_i > 0.5$, it can be considered to be in the correct community because the p'_i will not increase if the algorithm extract the node from its current community and put it into another one.

Our improved algorithm is summarized as follows:

(a) Each node of the network is initialized to be located in an isolated cluster.
(b) Generate a random node sequence, then check each node according to the sequence.
(c) For node i, calculate p'_i. If $1 - p'_i < p_{est}$ or $p'_i > 0.5$, skip this node. Otherwise, calculate C_i by using Eqs. (1), (3) and (4) for $p'_i = 0$ or using Eqs. (4) to (6) for $p'_i > 0$.
(d) Randomly choose C_i nodes from the neighbors which are located in different community with node i in the current partition, and the set of these nodes are denoted as N_i^{chosen}
(e) For each neighbor j in the chosen neighboring set N_i^{chosen}, calculate the increment of modularity.
(f) Find the neighbor j' that yields the largest modularity increment. If $\Delta Q_{max} > 0$, $c_i = c_{j'}$, move the node i into the community of node j'. Else, update the value of $\min(p'_i \mid i \textit{ is not moved})$.
(g) When reaching the end of the node sequence, $p_{est} = \min(p'_i \mid i \textit{ is not moved})$, and the algorithm jump back to step (b), reiterate the procedure until there is no more nodes to be moved.
(h) Aggregate the network, like the step (f) in the original Louvain algorithm, and jump back to step (a), reiterate the procedure until there is no more nodes to be moved.

4. Experiment results

To verify the performance of our new algorithm, we do experiments on both artificial benchmark networks and empirical networks. The original Louvain algorithm, the Random Neighbor Louvain algorithm [32] and our Random Self-adaptive Neighbors Louvain algorithm are used for comparison in our experiments. We use the modularity ratio to show the accuracy, the speedup ratio to show the speed, and the coefficient of variation to show the influence of randomness. The modularity ratio is calculated by $R_{QRNL} = \frac{Q_{RNL}}{Q_{original}}$ and $R_{QRSNL} = \frac{Q_{RSNL}}{Q_{original}}$, where $Q_{original}$ refers to the modularity of the partition uncovered using the original Louvain algorithm, and Q_{RNL} and Q_{RSNL} refer to the modularity using the Random Neighbor Louvain algorithm and the Random Self-adaptive Neighbors Louvain algorithm, respectively. The speedup ratio is calculated by $R_{speedRNL} = \frac{t_{original}}{t_{RNL}}$ and $R_{speedRSNL} = \frac{t_{original}}{t_{RSNL}}$, where $t_{original}$ is the runtime of the original Louvain algorithm, t_{RNL} is the runtime of the Random Neighbor Louvain algorithm and t_{RSNL} is the runtime of the Random Self-adaptive Neighbors Louvain algorithm. The coefficient of variation is defined as $V_x = \frac{\sigma_x}{E_x}$, where σ_x is the standard deviation and the E_x is the mathematical expectation. The coefficient of variation is a standardized measure of dispersion of a set of data, which is used to depict the fluctuation of the modularity of results to show the influence of randomness of each algorithm.

4.1. Artificial benchmark

LFR benchmark is a widely used artificial benchmark for structure analysis [37]. These benchmark networks contain planted partitions which we use all of the three algorithms to uncover. We use the LFR algorithm to build network datasets with different sizes and different mixing parameters μ. We generate 10 networks for each group and the algorithms run 50 times on each network.

We firstly test the impact of the network size. We set the mixing parameter μ of the LFR benchmarks to 0.2 and change the size of the network from 1000 to 20,000. The results depicted in Fig. 3(a) shows that our RSNL can gain as good performance as the Louvain algorithm does, and it is more accurate than the RNL. Fig. 3(b) shows the speedup ratio of the algorithms. Our improved algorithm is somewhat slower than the Random Neighbor Louvain algorithm, because it just picks up only one neighbor to calculate in the RNL but we pick up C_i neighbors to calculate ΔQ, which is considered as the most time-costing procedure in the algorithm. However, there is still an improvement of the speed of our algorithm compared with the original Louvain algorithm. In addition, with the size of the network increasing, the speedup ratio of the RNL drops, but that of the RSNL is still firm, for about 1.5 times.

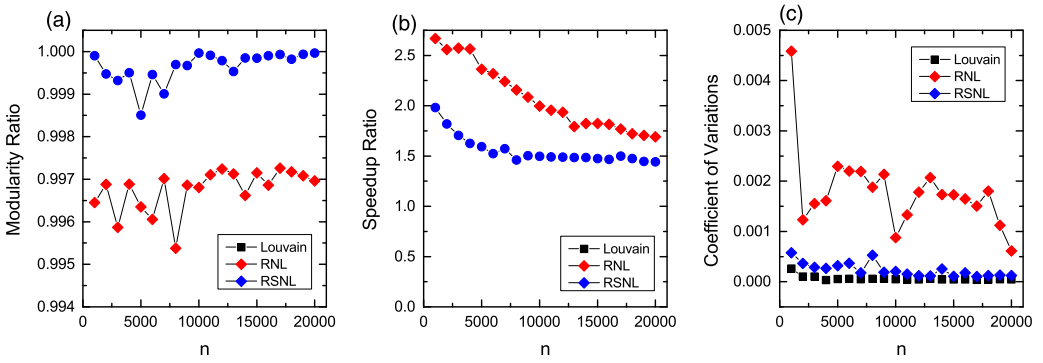

Fig. 3. The result of the algorithms on different LFR benchmarks with different size. (a) shows the modularity ratio of the results obtained by the algorithms, (b) shows the speedup ratio of the two improved algorithms and (c) shows the coefficient of variations of the three algorithms.

From Fig. 3(c), it can be seen that the coefficient of variations of RNL is much larger than the original Louvain algorithm, which indicates that it enlarges the influence of the randomness of the algorithm because of the uncertainty of its randomly choosing one neighbor method. The consequence led by this influence is that one researcher who uses the RNL to detect communities have to conduct more iterations of experiments to eliminate the influence of randomness in order to obtain as good result as the original Louvain algorithm does. The partition obtained by only one experiment will be lack of stringency. So, the total runtime of the experiment for one research may be actually increased.

On the contrary, the coefficient of variations of our RSNL is almost equal to that of the original Louvain algorithm, which means that the RSNL is more stable than the RNL. Although the RSNL also randomly chooses neighbors, the principle of the small probability events makes it possible to move the node into correct community. So, it does not involve more randomness and instability caused by the randomly choosing procedure.

The impact of the mixing parameter of LFR benchmarks is also tested. The mixing parameter μ is the parameter to set the partition of the neighbors of node i which is not located in the same community. The size of the networks is $n = 3000$ and we change the mixing parameters from 0.2 to 0.8. Fig. 4 shows the result of this test. When $\mu < 0.5$, the conclusion that we can derive from the figure is consistent with that of the test of different network sizes. With about 20%–40% slower than the RNL, the RSNL can obtain a result of higher modularity. The speed of the RSNL is still faster than that of the original Louvain, for about 1.7–1.8 times. The RSNL is also more stable than the RNL.

A special case is worth discussing when $\mu > 0.5$. When $\mu > 0.5$, it means that each node has more neighbors located in different communities than the neighbors located in the same community, which shows a disassortative community structure [38].

In a network with disassortative community structure, pairs of nodes are more likely to be connected if they are in different communities. So, the links inside the community are sparse and the links between communities are dense. This kind of community structure is opposite to the community structure depicted by the modularity. In this case, high values of modularity no longer indicate good partitions, and the partition with the highest modularity no longer indicates the ground-truth. The assumption we mentioned in Section 3 will not be true in this case.

To detect a disassortative community structure needs more information and it is beyond the ability of the Louvain algorithm. However, we can still use the benchmark networks with $\mu > 0.5$ to test the performance of the three modularity optimization algorithms, by considering the benchmark as a network planted with assortative community structure which is not distinct. In this way, we can continue using the assumption that the partition with the highest modularity indicates the ground-truth in the graph.

Fig. 4 shows that when dealing with a graph without distinct community structure in the topology, i.e. $\mu > 0.5$, our method can still get a result with the same modularity as the original one and it is faster, for over 1.4 times. But the modularity ratio of the RNL dramatically decreases when the mixing parameter increases, and when $\mu = 0.8$, the speedup ratio of the RNL is only about 0.95, which means that the RNL is even more slowly than the original Louvain. The coefficient of variations of the RSNL is almost equal to that of the original Louvain, but that of the RNL is much larger and increases fast when $\mu > 0.5$.

When $\mu > 0.5$, the Random Neighbor Louvain algorithm obtains result with much lower modularity, its time costing is the same or even larger than the original Louvain, and the influence of randomness is also greater. This is because the RNL randomly picks up only one neighbor to calculate ΔQ. The feasibility of this strategy is principally based on a priori that the probability, which the randomly chosen neighbor is located in the same community with node i in the planted community structure of the network, is large enough. This might be true when the mixing parameter is lower than 0.5, which indicates that most of the neighbors are located in the same community with node i and it is more probable to pick up a correct node than to pick up a wrong node randomly. However, if there is not a distinct assortative community structure planted in the graph, the proportion of the neighbors located in the same community with node i in the ground-truth is not so large. It is

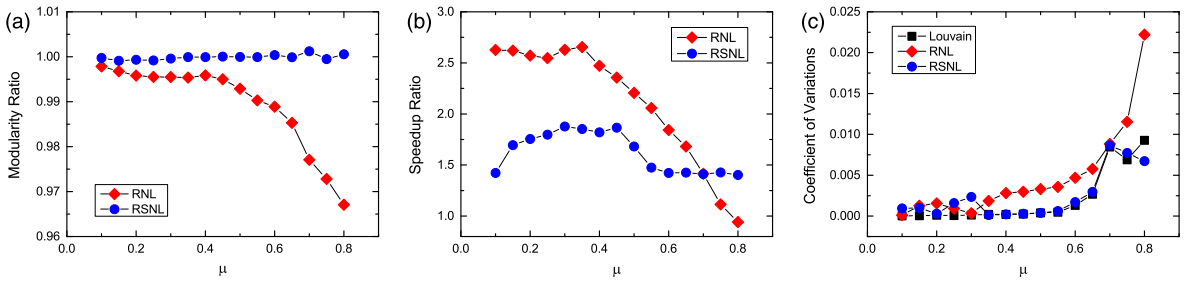

Fig. 4. The result of the algorithms on different LFR benchmarks with different μ.

Table 1
Empirical network overview.

Network	n	m	$\langle k \rangle$	k_{max}	c
Karate	34	78	4.588	17	0.256
UK faculty	81	817	20.173	62	0.473
Dolphins	62	159	5.129	12	0.309
Yeast	2 617	11 855	9.060	118	0.469
Facebook(NIPS)	2 888	2 981	2.064	769	0.000359
Train	64	2 818	7.564	29	0.561
Youtube	1 134 890	2 987 624	5.265	28 754	0.00622
ArXiv astro-ph	18 771	198 050	21.102	504	0.318

more likely to pick up a wrong node and therefore the disadvantages of the Random Neighbor Louvain algorithm appears more obviously.

By contrast, the method we propose uses an estimation of the proportion of correct nodes among the neighbors and the number of neighbors to pick up is based on this estimation. The method does not need an assumption of a priori knowledge. Although it is a roughly estimation, it is self-adaptive and is derived by the measurement during the procedure of the algorithm. So that though there is not a distinct community structure in the graph topology, our method can gain a result with almost the same modularity as the original Louvain algorithm in a lower runtime. And the influence of randomness of the RSNL is also maintained at the same level as the original Louvain algorithm.

4.2. Empirical networks

To verify the performance of our improved algorithm, we also make experiments on empirical networks. Table 1 shows the profiles of the datasets we choose. We conduct 20 times of experiments on each network.

The result depicted in Fig. 5 shows that our improved method can make a good performance on the empirical networks. Compared with the original Louvain algorithm, our method can uncover a comparable partition planted in the topology with a faster speed. The modularity ratios of the RSNL are maintained in about 1, which are larger than that of the RNL. And although the speedup ratios are at most about 25% smaller than that of the RNL, the RSNL is still faster than the original one, and the speedup ratio is about 1.5 times. The result indicates that our method can indeed decrease the runtime of the original Louvain algorithm on the empirical networks, and this acceleration does not involve reduction of accuracy, compared with the Random Neighbor Louvain algorithm.

Fig. 5(c) shows the ratio of coefficient of variation of modularity between the two improved algorithms and the original Louvain algorithm. From the figure we can see that the randomness in our algorithm is smaller than that of the Random Neighbor Louvain algorithm, in some cases it is even smaller than the original algorithm, such as the karate and the dolphins. Especially in the network of Facebook (NIPS), the results of the RSNL and the original Louvain do not change during the experiments, thus the coefficient of variation of these two algorithms is 0. However, the coefficient of variation of modularity of the Random Neighbor Louvain algorithm is 0.26, which is much larger than that of the original Louvain. The ratio of these coefficients of variation cannot be presented accurately on the figure because the denominator of the ratio is 0. The result indicates that our strategy of using the principle of small probability events is helpful to decrease the influence of randomness caused by the randomly chosen procedure and makes the result more stable.

4.3. Equivalent computing time

As we mentioned before, although the RNL runs faster than the RSNL for just one iteration on most of the networks in our test, the influence of the randomness of RNL is larger. To obtain as good result by the RNL as the original Louvain algorithm, one may need to conduct more iterations of experiments. So, the total runtime of the RNL may be actually increased.

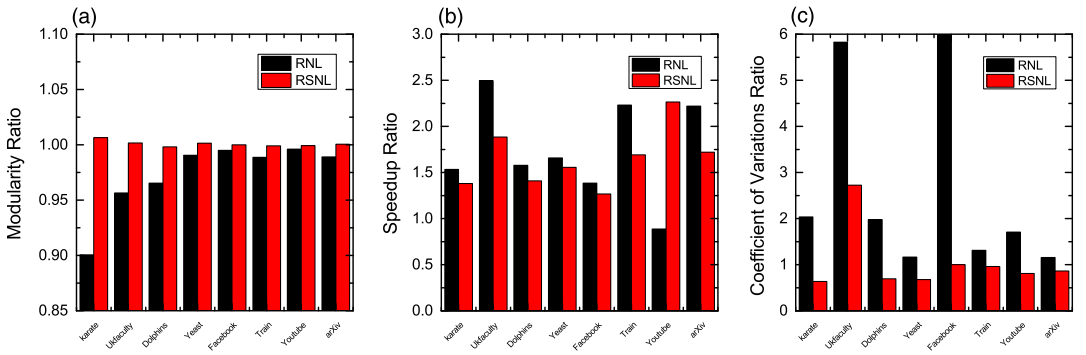

Fig. 5. The result of the test on the empirical networks.

To verify this conjecture, here we propose a new measure, the equivalent computing time. The modularity of the results obtained by the algorithms are fluctuant, and we hope to obtain a result with relatively large modularity. Those runtime of the iteration which obtains a relatively small modularity can be considered as the cost of the algorithm. So, given a standard modularity as the requirement, the equivalent computing time is the expectation runtime of the algorithm to obtain a result which achieves this requirement. In our test, we ran the algorithms several times on each network(50 times for the LFR benchmarks and 20 times for the empirical datasets), and obtained a set of results. Let $X = \{x_1, x_2, \ldots, x_h\}$ to be the set of the modularity of the results of an algorithm obtained by several iterations of experiments, and h is the total number of the iterations. Given the standard modularity Q_c, $N_c = |\{x_i | x_i \geqslant Q_c, x_i \in X\}|$ is the number of the results of which the modularity is not smaller than Q_c. And thus we have:

$$T_e = t * \frac{h}{N_c}, \tag{9}$$

where T_e is the equivalent computing time and t is the average runtime of this algorithm. The standard modularity Q_c can be arbitrary, and in our test, $Q_c = \alpha * Q_{Louvain}$, which means that we let the standard modularity change in the neighborhood of the average of the modularity obtained by the original Louvain algorithm.

Figs. 6–8 show the N_c of the algorithms on several certain datasets. It can be figured out that in most cases, the N_c of all these three algorithms drops to 0 if $\alpha > 1$, which indicates that it is hard to obtain results of which the modularity is even 0.1% larger than the average modularity of the original Louvain algorithm. While when $\alpha = 1$, which means that we just require the three algorithms to obtain as good result as the average modularity, it can be seen that our RSNL can reach this requirement. And in some cases, such as the network of LFR benchmark with $\mu = 0.8$ and the UK faculty, the N_c of RSNL is even larger than that of the original Louvain. On the contrary, the N_c of the RNL is 0 persistently, which means that the RNL cannot obtain a result with the same modularity with the average modularity of original Louvain algorithm even at once. And when $\alpha < 1$, the N_c of RSNL is still larger than that of the RNL, which shows the ability of finding good partitions of our algorithm.

Figs. 9–11 show the equivalent computing time of these three algorithms. It is shown in Fig. 9 that if the requirement of the modularity is less restrict, i.e. $\alpha = 0.995$, the RNL can achieve this requirement and its equivalent computing time is still smaller than that of the other two algorithms. And if the standard modularity is slightly larger, i.e. $\alpha = 0.998$, the RNL cannot achieve this requirement in most cases, where the $N_c = 0$ and the $T_e \to \infty$. But the RSNL can still achieve the requirement of modularity in $\alpha = 0.998$ with smaller T_e than that of the original Louvain. Fig. 10 shows a consistent conclusion, and from this figure one can find that when $\mu > 0.5$, the computing time to obtain a good result by RNL is significantly larger than the other two algorithms. Fig. 11 shows the speedup ratio calculated by the equivalent computing time. It is easy to find that even $\alpha = 0.995$, the speedup ratios of the RNL in most of the datasets in our test are 0, for its $T_e \to \infty$. On the contrary, the performance of our RSNL algorithm is much better. Especially on the network of UK faculty, the speedup ratio of T_e when $\alpha = 1$ reaches over 5 times. The results of equivalent computing time of these three algorithms show that if we consider the modularity, runtime and fluctuation as a whole, our RSNL performs much better than the RNL and the original Louvain algorithm.

5. Conclusion

Many networks are considered to contain community structure embedded in the topology of the graphs. Finding such communities is important in many different fields. The Louvain algorithm is one of the most widely used algorithms to optimize the modularity. In this paper, we propose a new method to improve the original Louvain algorithm. Compared with the Random Neighbor Louvain algorithm, we suggest to randomly choose several neighboring nodes to calculate the increment of modularity, neither all neighbors nor only one. To maintain the accuracy of the algorithm, we argue that by

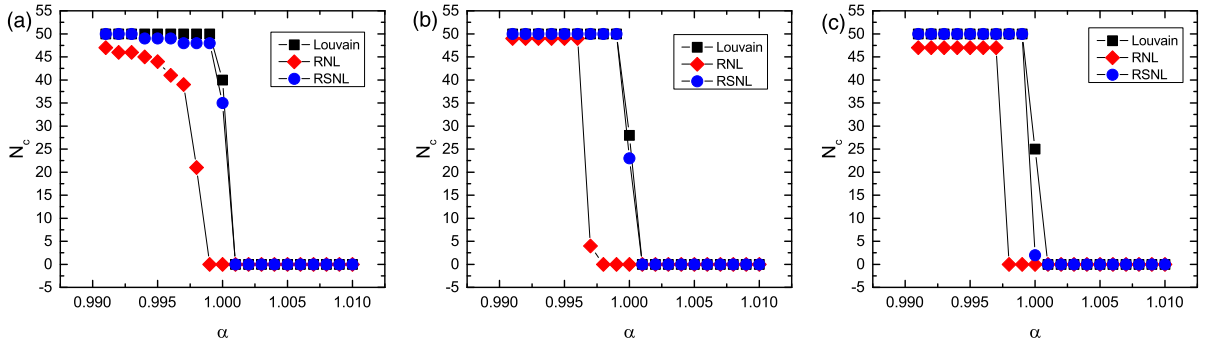

Fig. 6. The N_c of the algorithms on different LFR benchmarks with different size. (a) $n = 5000$ (b) $n = 10\,000$ (c) $n = 15\,000$.

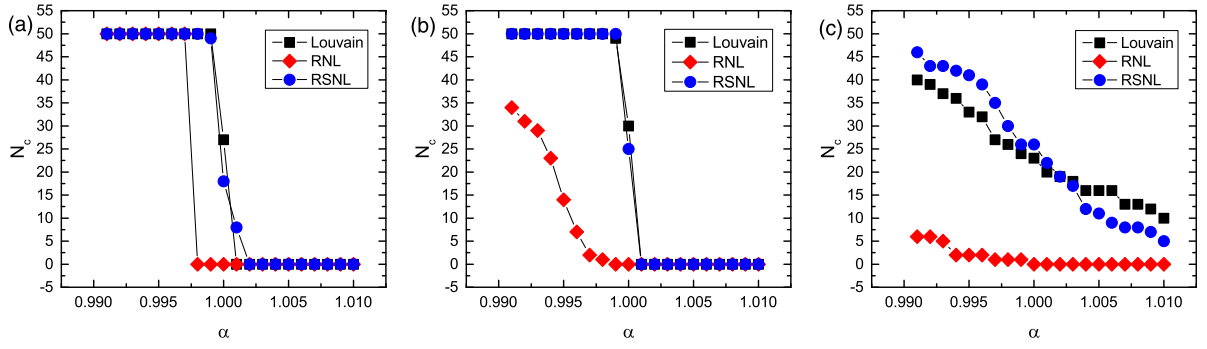

Fig. 7. The N_c of the algorithms on different LFR benchmarks with different μ. (a) $\mu = 0.1$ (b) $\mu = 0.5$ (c) $\mu = 0.8$.

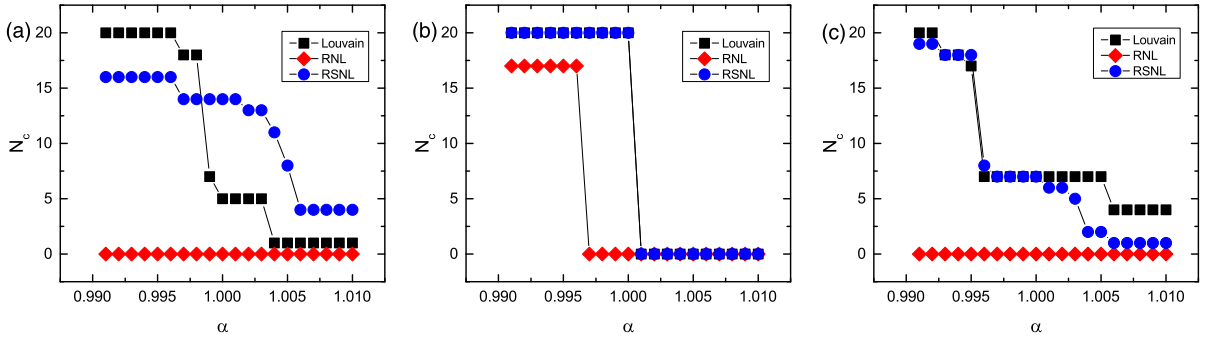

Fig. 8. The N_c of the algorithms on three empirical networks. We conduct 20 times of iteration on each networks, so that the maximum of N_c is 20 here. (a) UK faculty (b) Facebook (c) Dolphins.

using the principle of small probability event, C_i, the number of neighbors to be choose, should be able to make sure that the probability that all of the chosen neighbors are not correct nodes is less than 0.05. We also propose a simple method to estimate p_i, the proportion of the correct nodes among all of the neighbors of the node i, which is used in the derivation of C_i. The method of estimation is simple but useful. It does not rely on the assumption that there are more neighbors located in the same community with the node being manipulated in the ground-truth, as the Random Neighbor Louvain algorithm does, and it makes it possible to maintain the modularity of the result and does not involve much computing time.

We have tested our improved method on artificial benchmark networks and empirical networks. The improvement of our method is obvious across various benchmark network datasets, especially when using our algorithm on networks without distinct community structure. The result on the empirical networks also shows an excellent performance of our improved method. Our method can maintain the accuracy of algorithm, reduce the runtime, and diminish the influence of randomness involved by the random choosing procedure. In this paper, we proposed a new measure, the equivalent computing time, to consider the modularity, runtime and the influence of randomness as a whole to evaluate the performance of the algorithm. The results of the equivalent computing time of the three algorithms show that our RSNL algorithm can make it possible

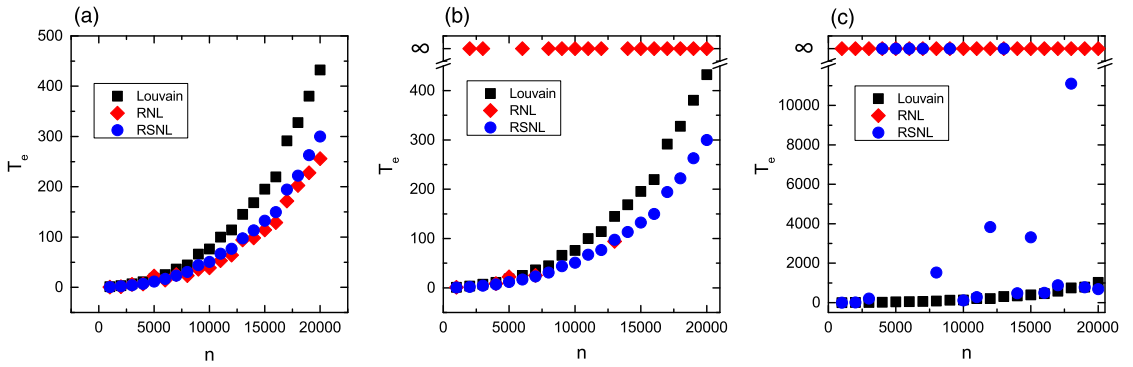

Fig. 9. The T_e of the three algorithms on different LFR benchmarks with different size. (a) $\alpha = 0.995$ (b) $\alpha = 0.998$ (c) $\alpha = 1$.

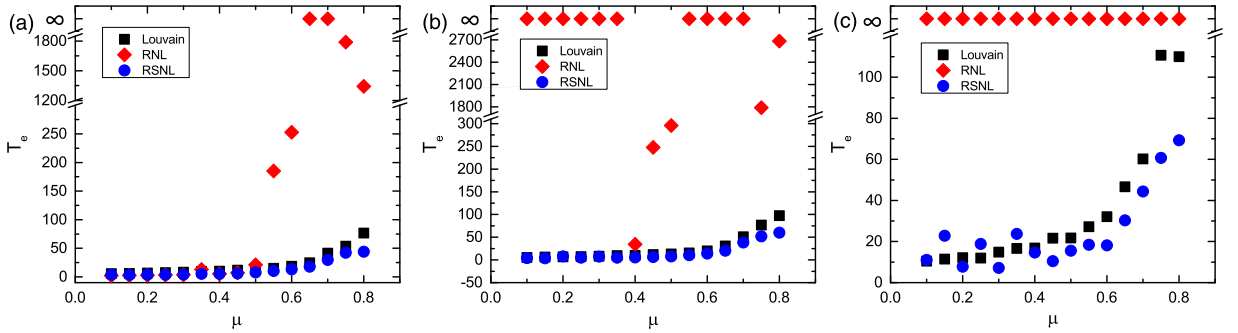

Fig. 10. The T_e of the three algorithms on different LFR benchmarks with different μ. (a) $\alpha = 0.995$ (b) $\alpha = 0.998$ (c) $\alpha = 1$.

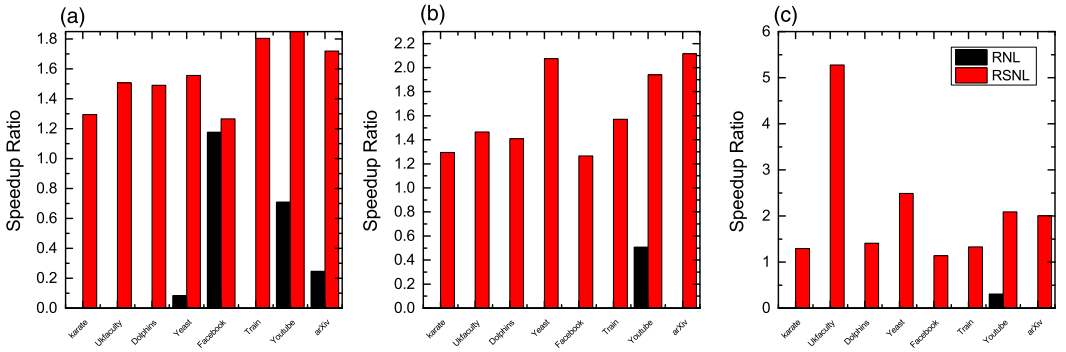

Fig. 11. The speedup ratio calculated by T_e of the algorithms on empirical networks. (a) $\alpha = 0.995$ (b) $\alpha = 0.998$ (c) $\alpha = 1$.

for researchers of many disciplines to detect communities faster on larger networks and need not to iterate many times of experiments to obtain a relatively good result.

Our RSNL can decrease the influence of the randomness caused by the randomly choosing procedure, but the shortcoming of the original Louvain algorithm that the result is dramatically influenced by the order of the sequential sweep over the vertices is still exist. Some further works on solving this problem are still required to improve the Louvain algorithm. In addition, there might be more suitable methods to estimate the value of p_i used in the RSNL. If conducting further researches on the estimation methods, the RSNL may have a better performance.

Furthermore, although our algorithm is not designed for detecting overlapping communities, there are still some ways to use our algorithm to detect overlapping community structure. For example, Shen et al. proposed that any modularity optimization algorithms can be used to partition the so-called maximal clique network to identify the overlapping community structure planted in the original network [39]. Thus, by exploiting their maximal clique network, our algorithm might be capable of detecting overlapping communities. While, more direct and efficient application of our algorithm on overlapping community structure still needs discussion and verification. It is worthy of studying further in the future.

V.A. Traag [32] mentioned that the idea of the Random Neighbor Louvain algorithm is that a random neighbor is likely to be in a good community. This might be not true in some networks where there is not distinct community structure. However, our improved method avoid this assumption. The idea of our method is that the neighbors chosen randomly is likely to contain at least one node being in the correct community, which might be workable on the improvement of other algorithms.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Contracts Nos. 11075057 and 11421505.

References

[1] M. Newman, Networks: An Introduction, Oxford University Press, Inc, 2010.
[2] R. Albert, A.L. Barabasi, Statistical mechanics of complex network, Rev. Mod. Phys. 74 (1) (2002) xii.
[3] M.E.J. Newman, The structure and function of complex networks, SIAM Rev. 45 (2003) 167–256.
[4] S.N. Dorogovtsev, J.F.F. Mendes, Evolution of Networks: From Biological Nets to the Internet and WWW (Physics), Oxford University Press, Inc., 2013.
[5] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (7043) (2005) 814–818.
[6] Mislove, Alan, Marcon, Massimiliano, Gummadi, P. Krishna, et al., Measurement and analysis of online social networks, 2007.
[7] G.W. Flake, S. Lawrence, C.L. Giles, Efficient identification of Web communities, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 150-160), ACM, 2000.
[8] A. Vespignani, Evolution and structure of the internet: A statistical physics approach, in: Evolution and Structure of the Internet: A Statistical Physics Approach, Cambridge University Press, 2004.
[9] Santo. Fortunato, Community detection in graphs, Phys. Rep. 486 (3–5) (2010) 75–174.
[10] M.E. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[11] Vittoria Colizza, Alain Barrat, Marc Barthlemy, Alessandro Vespignani, The role of the airline transportation network in the prediction and predictability of global epidemics, Proc. Natl. Acad. Sci. USA 103 (7) (2006) 2015–2020.
[12] R. Guimerà, M. Sales-Pardo, L.A.N. Amaral, Nat. Phys. 3 (2007) 63.
[13] A.W. Rives, T. Galitski, Modular organization of cellular networks, Proc. Natl. Acad. Sci. USA 100 (3) (2003) 1128–1133.
[14] V. Spirin, L.A. Mirny, Protein complexes and functional modules in molecular networks, Proc. Natl. Acad. Sci. USA 100 (21) (2003) 12123–12128.
[15] J. Chen, B. Yuan, Detecting functional modules in the yeast proteinCprotein interaction network, Bioinformatics 22 (18) (2006) 2283–2290.
[16] G.W. Flake, S. Lawrence, C. Lee Giles, F.M. Coetzee, Self-organization and identification of web communities, IEEE Comput. 35 (2002) 66–71.
[17] Y. Dourisboure, F. Geraci, M. Pellegrini, Extraction and classification of dense communities in the web, in: WWW 07: Proceedings of the 16th International Conference on the World Wide Web, ACM, New York, NY, USA, 2007, pp. 461–470.
[18] J.S. Coleman, An Introduction to Mathematical Sociology, Collier-Macmillan, London, UK, 1964.
[19] L.C. Freeman, The Development of Social Network Analysis: A Study in the Sociology of Science, BookSurge Publishing, 2004.
[20] C.P. Kottak, Cultural Anthropology, McGraw-Hill, New York, USA, 2004.
[21] J. Moody, D.R. White, Structural cohesion and embeddedness: A hierarchical concept of social groups, Amer. Sociol. Rev. 68 (1) (2003) 103–127.
[22] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (1970) 291–307.
[23] M. Fiedler, Algebraic connectivity of graphs, Czechoslovak Math. J. 23 (98) (1973) 298–305.
[24] A. Arenas, A. Díaz-Guilera, C.J. Préz-Vicente, Synchronization reveals topological scales in complex networks, Phys. Rev. Lett. 96 (11) (2006) 114102.
[25] U.N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (3) (2007) 036106.
[26] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99 (12) (2002) 7821–7826.
[27] M.E.J. Newman, Fast algorithm for detecting community structure in networks, Phys. Rev. E 69 (6) (2004) 066133.
[28] J.M. Pujol, J. Bejar, J. Delgado, Clustering algorithm for determining community structure in large networks, Phys. Rev. E 74 (1) (2006) 016107.
[29] H. Du, M.W. Feldman, S. Li, X. Jin, An algorithm for detecting community structure of social networks based on prior knowledge and modularity, Complexity 12 (3) (2007) 53–60.
[30] R. Guimerà, M. Sales-Pardo, L.A.N. Amaral, Modularity from fluctuations in random graphs and complex networks, Phys. Rev. E 70 (2) (2004) 025101(R).
[31] V.D. Blondel, J.L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.
[32] V.A. Traag, Faster unfolding of communities: Speeding up the Louvain algorithm, Phys. Rev. E 92 (3) (2015) 032801.
[33] N. Ozaki, H. Tezuka, M. Inaba, A simple acceleration method for the Louvain algorithm, Int. J. Comput. Electr. Eng. 8 (3) (2016) 207.
[34] B. Hu, W. Li, X. Huo, et al. Improving Louvain Algorithm for Community Detection, in: International Conference on Artificial Intelligence and Engineering Applications, 2016.
[35] W. Li, W. Jiang, S. Hua-Wei, C. Xue-Qi, An improvement of the fast uncovering community algorithm, Chin. Phys. B 22 (10) (2013) 108903.
[36] R.A. Fisher, Statistical Methods for Research Workers, Genesis Publishing Pvt Ltd, 1925.
[37] A. Lancichinetti, S. Fortunato, F. Radicchi, Benchmark graphs for testing community detection algorithms, Phys. Rev. E Stat. Nonlinear & Soft Matter Phys. 78 (2) (2008) 046110.
[38] C. Moore, X. Yan, Y. Zhu, et al. Active learning for node classification in assortative and disassortative networks, 2011, 841–849.
[39] H.W. Shen, et al., Quantifying and identifying the overlapping community structure in networks, J. Stat. Mech. Theory Exp. 2009 (7) (2009) 07042.