Applied Network Science

RESEARCH Open Access

Local memory boosts label propagation for community detection

Antonio Maria Fiscarelli[1,2*], Matthias R. Brust[2], Grégoire Danoy[2,3] and Pascal Bouvry[2,3]

*Correspondence:
antonio.fiscarelli@uni.lu
[1]C2DH, University of Luxembourg, 11 Porte des Sciences, Esch-sur-Alzette, Luxembourg
[2]SnT, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette, Luxembourg
Full list of author information is available at the end of the article

Abstract
The objective of a community detection algorithm is to group similar nodes that are more connected to each other than with the rest of the network. Several methods have been proposed but many are of high complexity and require global knowledge of the network, which makes them less suitable for large-scale networks. The Label Propagation Algorithm initially assigns a distinct label to each node that iteratively updates its label with the one of the majority of its neighbors, until consensus is reached among all nodes in the network. Nodes sharing the same label are then grouped into communities. It runs in near linear time and is decentralized, but it gets easily stuck in local optima and often returns a single giant community. To overcome these problems we propose MemLPA, a variation of the classical Label Propagation Algorithm where each node implements a memory mechanism that allows them to "remember" about past states of the network and uses a decision rule that takes this information into account. We demonstrate through extensive experiments, on the Lancichinetti-Fortunato-Radicchi benchmark and a set of real-world networks, that MemLPA outperforms other existing label propagation algorithms that implement memory and some of the well-known community detection algorithms. We also perform a topological analysis to extend the performance study and compare the topological properties of the communities found to the ground-truth community structure.

Keywords: Network analysis, Community detection, Label propagation

Introduction

Real-world networks often exhibit a community structure where nodes in a community are densely connected while different communities are loosely connected. The objective of a community detection algorithm is to group similar nodes that are more connected to each other than with the rest of the network. Community detection can be seen as a generalization of the partitioning problem, where the network is divided into a fixed number of equally sized partitions. This problem is known to be NP-hard (Brandes et al. 2008). Furthermore, many community detection algorithms are based on modularity optimization. Modularity (Q) is a measure of partition quality (Newman 2004) and finding the partition that maximizes modularity is NP-Hard. Therefore it is important that community detection algorithms maintain a low complexity while possessing high scalability. Due to growing interest, community detection has attracted many researchers from different areas such as computer science (Albert et al. 1999), natural sciences (Jeong et al. 2000) and social sciences (Scott 1988), making it a notably active research field.

Springer Open

© The Author(s). 2019 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Several community detection methods have been proposed in the literature: greedy algorithms based on modularity optimization (Girvan and Newman 2002; Clauset et al. 2004; Blondel et al. 2008; Newman and Girvan 2004), spectral methods (Newman 2006) and methods based on random walk processes (Pons and Latapy 2005; Rosvall and Bergstrom 2008). Many of these methods have high complexity and require global knowledge of the network, making them unsuitable for large-scale networks. The Label Propagation Algorithm (LPA) initially assigns a distinct label to each node that iteratively updates its label following the one of the majority of its neighbors, until consensus is reached among all nodes in the network. This method runs in near linear time, is scalable and requires only local information of the network. It is thus especially suitable for large networks. Unfortunately, it also gets easily stuck in local optima and is thus outperformed by more recent and sophisticated algorithms.

In this paper we propose MemLPA, a variation of the classical LPA where each node implements a memory mechanism that allows them to "remember" about past states of the network and uses a decision rule that takes this information into account. We also adapt some of the improvements proposed in the literature to our method such as node preference and termination criterion based on active nodes. We show that the use of memory improves performance and prevents a single label from overpropagating in the network, forming a single giant community. We conducted extensive experiments on the Lancichinetti-Fortunato-Radicchi (LFR) benchmark and a set of real-world networks. Several LPA variations are compared and MemLPA is tested against other existing label propagation algorithms that implement memory, obtaining better results. It is also tested against well-known community detection algorithms, outperforming some of them for mixing parameter values between 0.5 and 0.8. In this paper we extend our previous work (Fiscarelli et al. 2018) in two different ways. First we compare MemLPA to other state-of-the-art memory-based LPA algorithms. Second we extend the performance study with a topological analysis. We use several metrics to compare the topological properties of the communities found by the different algorithms to the ground-truth community structure.

The remainder of this article is organized as follows. "Related work" section presents a state-of-the-art analysis on community detection and label propagation algorithms. MemLPA is introduced in "MemLPA: a memory-based label propagation algorithm" section and its performance is analyzed and compared to other community detection algorithms on artificial and real-world networks in "Performance study" section. Finally, "Conclusions" section provides our conclusions.

Related work

In this section we present an overview of community detection algorithms. In particular, we discuss LPA and several variations proposed in the literature.

Community detection

Girvan and Newman (2004) were first to propose a divisive hierarchical algorithm based on edge betweenness: given an edge, it measures the number of shortest paths between all pairs of nodes in the network that pass through this edge. Removing edges with high betweenness will enhance the separation of communities. This method ranks edges according to their betweenness and iteratively removes them. At the end, the configuration that achieves the highest modularity is chosen. Its complexity is $O(nm^2)$. A faster version of this method was also proposed (Clauset et al. 2004): it is a heuristic algorithm that, at each iteration, merges nodes into communities to optimize modularity. This method runs in $O(md \log n)$, where d is the depth of the dendrogram. Blondel et al. (2008) proposed a similar method called Louvain. All nodes are initially assigned to a different community and, at each iteration, each node is moved to the community that achieves the highest modularity improvement. Once communities are defined, a new network is built, where nodes represent the communities found. The process iterates until improvement no longer occurs. It runs in $O(n \log n)$.

Walktrap (Pons and Latapy 2005) defines a similarity between nodes according to the transition probability of random walkers. A random walker is an agent that, starting from a random node, moves from one node to another with a uniform probability. It runs in $O(n^2 m)$ or $O(n^2 \log n)$ on sparse networks. Infomap, similarly, is a global optimization method that optimizes a quality function defining the code length of a random walk process in the network. Its complexity is $O(m)$. Newman (2006) also proposed a spectral method based on the Eigenspectrum of the modularity matrix. Its leading eigenvector is computed and the network is split into two subcommunities such that modularity is maximized. The process is then repeated on the communities just found. This method runs in $O(n(m + n))$ or $O(n^2)$ on sparse networks. Finally, Reichardt and Bornholdt (2006) interpreted community detection as the minimization of the energy function of a spin model, where communities are seen as spin configurations. It runs in $O(n^{3.2})$ on sparse networks.

Label propagation algorithm

Many of the algorithms described are not suitable for large-scale networks: they have high complexity and require global information of the network. To overcome this problem, Raghavan et al. (2007) proposed the Label Propagation Algorithm. It initially assigns a distinct label to each node that iteratively updates its label following the majority voting rule, until consensus is reached among all nodes in the network. This method runs in near linear time, is scalable and uses the network's local information only, without the need of optimizing any objective function. Unfortunately, LPA gets easily stuck in local optima and is thus outperformed by more recent and sophisticated algorithms. Furthermore, a certain label may overpropagate and create a single giant community.

Several improvements have been proposed. Barber and Clark (2009) developed a variation that takes into account modularity while applying the majority rule. This method was extended by Liu and Murata (2010) with a greedy method that, given the communities found, merges them in an attempt to improve modularity, allowing the algorithm to escape from local optima. Leung et al. (2009) introduced a decision rule based on node preference, in this case node degree, to improve performance: when a node applies the decision rule, labels of nodes having a higher degree will be assigned a higher score. They also extend the algorithm with hop attenuation: every time a label is propagated through the network, a negative score is assigned to it in order to prevent a certain label from flooding the network. The algorithm is scalable and still runs in near linear time. Xie and Szymanski (2011) proposed another node preference, based on neighborhood overlapping, that is shown to be related to the clustering coefficient. Šubelj and Bajec (2011) elaborated two particular strategies, called defensive preservation and offensive expansion, that adapt node preference to focus on core nodes and border nodes of communities. They are combined and applied hierarchically. They also found that the network structure affects the effectiveness of node preference and hop attenuation. This algorithm runs in $O(m^{1.19})$ and is highly scalable. Xie and Szymanski (2013) also developed LabelRank, a variation of the classical LPA that takes inspiration from the MCL (Markov Cluster Algorithm) (Dongen 2000). Instead of a single label, each node maintains a list of label distributions that is updated at each iteration. An inflation operator is used to enhance the gap between strong and weak labels, while a cutoff operator is applied to remove labels below a certain threshold, in order to shorten these lists and make the computation more efficient.

To our knowledge, there are only few methods that explicitly refer to the use of memory in LPA, where nodes collect labels from previous iterations to keep track of past states of the network. The Speaker-Lister Label Propagation Algorithm (SLPA) (Xie et al. 2011) is based on an information dynamic rule: for each node, its neighbors select one label from their memory according to a speaking rule and the node updates its memory according to a listener rule. After a fixed number of iterations, a thresholding procedure is applied to each node's memory to assign it to one or multiple communities. Another memory-based LPA algorithm (MLPA) (Hosseini and Azmi 2015) implements a memory element that stores the label of each node on each iteration, in order to have a snapshot of the state of the network at each iteration. After a fixed number of iterations, the most frequent label in each node's memory is chosen and a last round of the classical LPA is performed to assign nodes to communities. We also included a more recent LPA variation called fluidC (Fluid Communities) (Parés et al. 2017), based on the idea of fluids expanding and contracting as a result of their interaction. The algorithm initializes a certain number of community seeds in the network and updates the community each vertex belongs to using an update rule based on fluid density. This algorithm requires the number of communities to be set at start. Some work on consensus dynamics also refers to memory: a non-deterministic version of the Naming Game (Reginaldo Filho et al. 2009; Uzun et al. 2011), which is similar in some aspects to LPA, extends the agents with local memory.

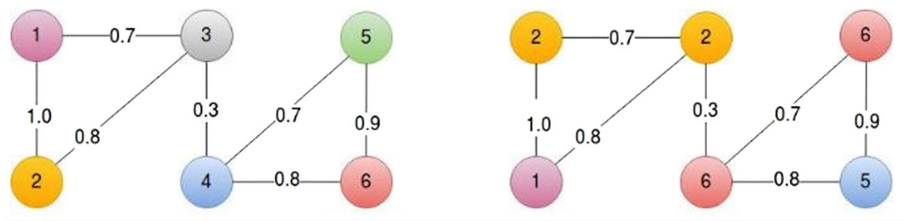
MemLPA: a memory-based label propagation algorithm

In the classical LPA, each node updates its label according to the current state of the network. Each node collects its neighbors' labels and selects the most chosen one according to a majority rule. This mechanism does not consider past states of the network, since each node collects new labels at each iteration and discards the previous ones, making the algorithm memory-less. In this section we introduce MemLPA, a variation of the classical LPA where each node implements a memory mechanism that allows them to "remember" about past states of the network and uses a decision rule that takes this information into account.

Algorithm description

When using memory, labels are not discarded but updated at each iteration. Each node maintains a list of labels with its associated score. Initially, each node is assigned a distinct label (line 2 of the pseudo-code) and its memory is empty (line 3). At each iteration, each node collects its neighbors' labels (line 8) and updates its memory according to edge weight (for weighted networks) and node preference (line 9). If a new label is not in memory already, a new entry is created, otherwise the score for the corresponding label is incremented. Each node then selects a label from its memory using a decision rule that takes into account the labels' score, in this case the label having maximum score (line 11). This mechanism can be applied to directed or undirected as well as to weighted or unweighted graphs. Figure 1 shows how MemLPA works.

In order to keep MemLPA scalable, we propose a synchronous update rule: each node independently updates its label according to the state of the network during the previous iteration. A synchronous update may cause LPA to oscillate between two different configurations, therefore we show in "Performance study" section how the two different update rules affect the convergence of MemLPA. As node preference, we use a heuristic based on neighborhood overlapping, computing the fraction of neighbors that a node shares with another. When updating a node's memory, a higher score will be assigned to labels coming from nodes that have many neighbors in common. In "Performance study" section we show the impact of this heuristic on performance. To speed up the algorithm, we define a cutoff operator to prune each node's memory (line 10). At each iteration, all labels below a certain threshold are deleted, keeping only the most relevant ones. Regarding the termination criterion, several options have been proposed in the literature, based on convergence, modularity improvement, active nodes and scarcity of updates. Many of these options are based on global information of the network, therefore we decided to use a termination criterion based on active node list: a node is considered active if the label chosen during the current iteration is different from the previous one or if any of its neighbors becomes active again. The active node list initially contains all nodes (line 5) and at each iteration a node is removed if it is no longer active or it is added if it becomes active again (line 13). The decision rule is applied only on active nodes and the algorithm terminates when the active node list is empty. This keeps the algorithm decentralized and speeds up the algorithm compared to applying the decision rule on every node. In "Performance study"



Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
2 – 1.0	**1 – 1.0**	1 – 0.7	3 – 0.3	4 – 0.7	4 – 0.8
3 – 0.7	3 – 0.8	**2 – 0.8**	5 – 0.7	**6 – 0.9**	**5 – 0.9**
		4 – 0.3	**6 – 0.8**		

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
2 – 1.7	1 – 1.0	1 – 1.5	3 – 0.3	4 – 0.7	4 – 0.8
3 – 0.7	3 – 0.8	**2 – 1.5**	**5 – 1.5**	**6 – 1.6**	5 – 0.9
1 – 1.0	**2 – 1.8**	4 – 0.3	6 – 1.5	5 – 0.9	6 – 1.7
		6 – 0.3	2 – 0.3		

Fig. 1 Iterations of MemLPA on a weighted undirected graph. Color and node number represent labels. Columns in the table represents nodes memory. In the first iteration, node 1 receives labels 2 and 3 from its neighbors. Using the classical LPA, in the second iteration, it would discard these labels and collect new ones (label 1 and 2). With memory, instead, the old labels are not discarded but updated, therefore node 1 contains labels 1, 2 and 3

section we show how the termination criterion based on active node list affects performance and convergence of the algorithm. The decision rule based on memory that MemLPA uses, as well as MLPA, may result in singleton communities. For these nodes, an additional round of label propagation without memory is performed in order to assign them to a bigger community.

Algorithm 1: MemLPA

```
Input  : Graph G(N, E)
Output: Communities C
1  for $n \in N$ do
2      $c_n \leftarrow l_n$ //Assign unique label to nodes
3      $M_n \leftarrow \emptyset$ //Initialize memory
4  end
5  $AL \leftarrow N$ //Initialize active list
6  while $AL \neq \emptyset$ do
7      for $n \in AL$ do
8          $C_n \leftarrow CollectLabels(Neigh(n))$ ;
9          $M_n \leftarrow UpdateMemory(C_n)$ ;
10         $M_n \leftarrow \{l_n^m \in M_n, m \in N \mid |mean(M_n) - sd(M_n)| \leq l_n^m\}$
11         $c_n \leftarrow ApplyRule(M_n)$ ;
12     end
13     $AL \leftarrow UpdateActiveList(AL)$
14 end
```

Complexity

The complexity of MemLPA on a certain node, where k is the average degree and h is the average memory length, can be assessed this way:

- Collecting labels for a node with k neighbors has complexity $O(k)$.
- Updating a node's memory with k new values has complexity $O(k)$.
- Using the cutoff operator on a node's memory has complexity $O(k)$.
- Choosing a new label from memory has complexity $O(h)$.

Node preference, if used, can also affect complexity. Neighborhood overlapping, on a node with k neighbors, has complexity $O(k)$ (Xie and Szymanski 2011), while node preference based on node degree has complexity $O(1)$ (Leung et al. 2009). Notice that the information needed for node preference must only be computed during the first iteration and nodes can store and reuse this information. In "Performance study" section we show how the cutoff operator keeps the average memory length constant and significantly lower than the average node degree. Iterating on all nodes, the overall complexity of MemLPA is $O(k * n)$ or $O(m)$, therefore comparable to $O(m)$ of the classical LPA (Raghavan et al. 2007). Therefore, the complexity of MemLPA is still near linear w.r.t. network size.

Performance study

We implemented MemLPA and assessed the use of memory and some of the variations proposed in the literature. We then compared it to other memory-based label propagation algorithms and well-known community detection algorithms. We also ran MemLPA to study some of its characteristics that are important for the convergence of the algorithm. For the analysis we ran all algorithms on the LFR benchmark (Lancichinetti et al. 2008), an established benchmark in the literature for community detection, that allows to generate networks with properties similar to real-world networks. As performance metrics, we used classical clustering metrics such as Normalized Mutual Information (NMI) (Danon et al. 2005; Yang et al. 2016) and Adjusted Rand Index (ARI) (Hubert and Arabie 1985), as well as topological metrics such as community size, internal transitivity, scaled density, average distance, hub dominance and internal modularity (Orman et al. 2012). We also applied these algorithms on a set of real-world networks of different nature and used the modularity measure (Newman and Girvan 2004) to evaluate the quality of the community assignments found.

Performance metrics

The most common metrics used to evaluate community detection algorithms come from classical clustering, where communities are seen as partitions of nodes and they are compared to the ground-truth communities. NMI is an information theoretic metric that measures the amount of information that two partitions share. It ranges from 0 to 1, assigning 1 to communities that perfectly match the ground truth and 0 to a completely random assignment. The drawback of NMI is that it depends on the network size and number of communities. For example, if a certain community detection algorithm fails and assigns a different community to each node, NMI will assume a value that is not the same for each network but depends on these network parameters. ARI measures the proportion of pairs of nodes that are correctly assigned to the same community. It ranges from -1 to 1, assigning 1 for a perfect assignment, 0 for a random assignment and -1 for a bad assignment. Unlike NMI, it does not depend on any network characteristics.

Modularity measures the fraction of edges connecting vertices inside the same community and compares it to the same quantity computed on a random graph of the same size and average degree. Modularity will be higher if the network exhibits a community structure. It does not need the ground-truth community assignment to be known and is only based on the structure of the network. The drawback of modularity is the resolution limit: this metric is not accurate when computed on networks containing small communities. In order to solve this issue, many algorithms based on modularity optimization make use of a resolution limit parameter (Lambiotte et al. 2014).

There are also several metrics that allow to study the topological properties of a community assignment. The most common one is the community size. For many real world networks, the community size distribution follows a power-law, meaning that there is a majority of small communities and few large ones. The community size distribution, in general, provides very good information about the quality of a community assignment (Dao et al. 2018). The internal transitivity of a community is defined as the average local transitivity over all nodes, where the local transitivity of a node measures the fraction of links between its neighbors. The formula is the following: $\frac{1}{s_i}\frac{1}{k_i-1}\sum_{h,j\in C}\frac{w_{i,j}+w_{i,h}}{2}a_{i,j}a_{i,h}a_{j,h}$, where s_i is the strength of node i (sum all of the weights of its edges), k_i is the node internal degree, $w_{i,j}$ is the weight of the edge connecting nodes i and j and $a_{i,j}$ is an element of the adjacency matrix. The scaled density is defined as the density of a community weighted by its size. The formula is the following: $\frac{2m_C}{n_C(n_C-1)}$, where m_C is the number of edges in the community and n_C is the number of nodes in the community. The average distance of a community is the average shortest path between all pairs of nodes inside the community. The hub dominance is defined as the maximal internal degree of a node divided by its maximum theoretical value. The formula is the following: $max_{i \in C}\left(\frac{k_i}{n_C-1}\right)$. Finally, the internal modularity is simply defined as the modularity of a community. The formula is the following: $\frac{1}{2m_c}\sum_{i,j\in C}\left[\left(a_{i,j}-\frac{k_ik_j}{2m_c}\right)a_{i,j}\right]$. These metrics have been shown to be a valid complementary tool to evaluate and compare community detection algorithms (Orman et al. 2012; Jebabli et al. 2018). In fact, communities found by different algorithms can score same NMI or ARI and still be topologically different. For example, they may have different community size distributions.

Cluster analysis

In this section we use classical clustering metrics such as NMI and ARI to assess the use of memory and some of the variations proposed in literature. We then compare MemLPA to other memory-based label propagation algorithms. Finally, we compare MemLPA to other well-known community detection algorithms. We also run MemLPA to study its convergence. We run all algorithms on the LFR benchmark and a set of real-world networks.

Artificial networks

The first set of experiments was conducted on the LFR benchmark to investigate the advantages of the LPA variations chosen and the use of memory. A mixing parameter μ controls the portion of intra-community edges. Node degree and community size distribution, like in many real-world networks, follow a power-law distribution. Benchmark graphs were generated with a number of nodes $N = 1000$, minimum community size $C.min = 10$, maximum community size $C.max = 50$, average degree $K.avg = 20$, maximum degree $K.max = 50$, degree exponent $K.exp = 2$ and community size exponent $C.exp = 1$, while μ was dynamically changed.

We compared the classical LPA to different variations of MemLPA that use synchronous (S) and asynchronous update rule, with and without node preference (N), with and without cutoff operator (C). Figure 2 shows that using a synchronous or asynchronous update does not make a significant change in performance (N_C_S vs N_C). Using the cutoff operator does not degrade performance either (C_S vs S and N_C_S vs N_S). This shows that MemLPA can be decentralized and scalable without any loss in performance. For low values of μ all variations obtained optimal results. The classical version of LPA, the only one not using memory, was the first algorithm to drop in performance for $\mu \geq 0.5$. In fact, a label flooded the network and created a single giant community. This confirms that the use of memory improves performance and prevents a label from overpropagating in the network. For $\mu \in [0.5, 0.7]$ the variations that use node preference (N_S, N_C and N_C_S) obtained the best results, but it is not the case for higher values. In fact, the variations that did not use node preference (S and C_S) obtained higher values of NMI for $\mu \in [0.7, 1]$. We must consider that the NMI depends on network size and number of communities. Therefore we decided to look at the ARI to have a more accurate comparison.

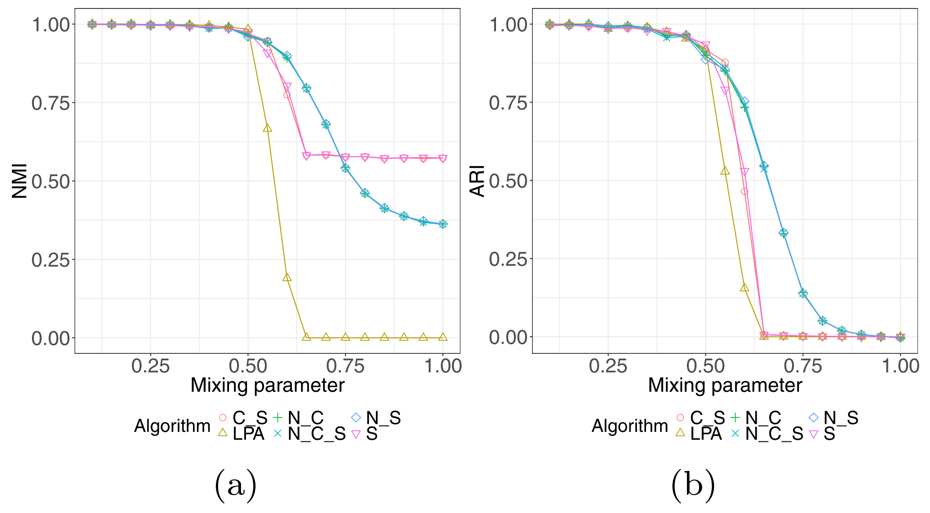


Fig. 2 Experiments on the LFR benchmark. All variations, except the classical LPA, implement memory. N: node preference, C: cutoff operator, S: synchronous update. Experiments are run 20 times and results averaged

We can see that, in this case, variations using node preference actually achieve better results.

The same experiment was conducted to compare MemLPA to other memory-based label propagation algorithms. Figure 3 shows that, for low values of μ, LPA, MemLPA and SLPA achieve perfect results, while fluidC and MLPA do not. For $\mu \geq 0.5$, all algorithms' performance start dropping. LPA's performance is first to drop to zero, showing that the use of memory in any of the algorithms is beneficial. LPA and SLPA both find a single giant community, and MemLPA achieves the best performance overall.

Finally, we compared MemLPA to other well-known community detection algorithms. We chose some of the algorithms described in "Related work" section (all available in the igraph R package (Csardi and Nepusz 2006)). Figure 4 shows that, for low values of μ, most algorithms obtained optimal results, while Greedy gradually decreased in performance. For $\mu \in [0.5, 0.7]$ most of the algorithms started degrading in performance, especially LPA

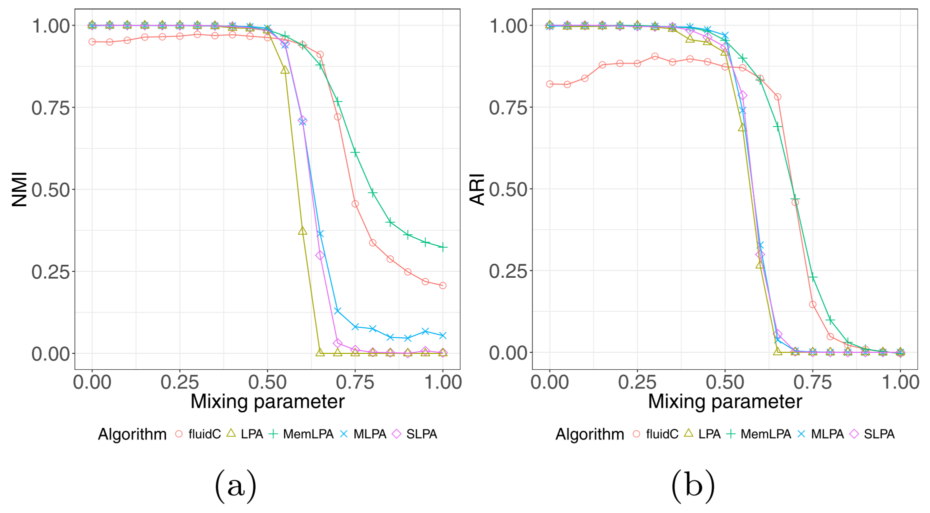


Fig. 3 Experiments on the LFR benchmark. Experiments are run 20 times and results averaged

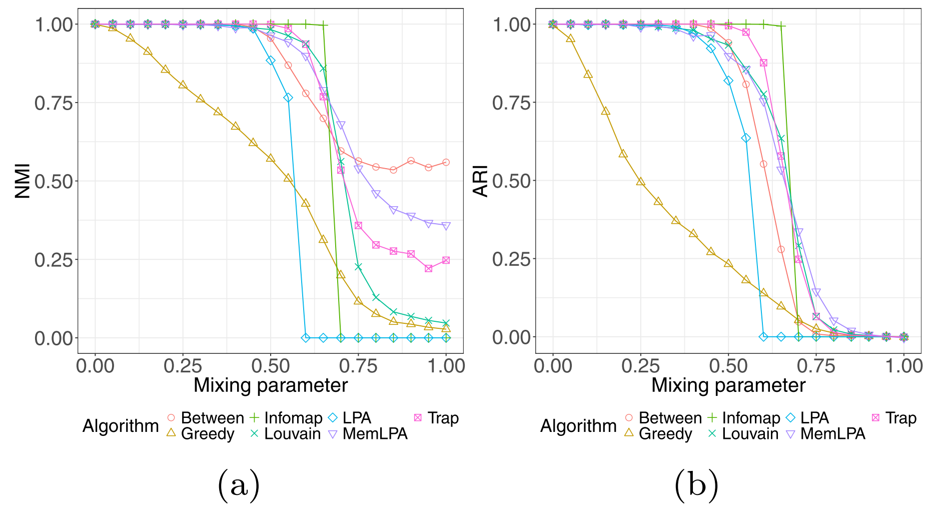


Fig. 4 Experiments on the LFR benchmark. Experiments are run 20 times and results averaged

and Between. MemLPA, in this range, was only outperformed by Infomap and Trap. For $\mu \geq 0.7$ MemLPA was the best algorithm after Between but, looking at the ARI, MemLPA performed slightly better until all algorithms' performance dropped.

We also conducted two experiments to analyze some of the characteristics of MemLPA at runtime. We used $\mu = 0.1$ to generate networks where communities are very well defined and $\mu = 0.6$ for loose communities. As performance measures we recorded NMI, modularity and the ratio between the number of communities found by MemLPA and real communities. The information that we recorded is the percentage of runs that terminated, the number of active nodes and the average ratio between memory length and node degree. Figure 5 shows that, for $\mu = 0.1$, MemLPA increased in performance quickly,

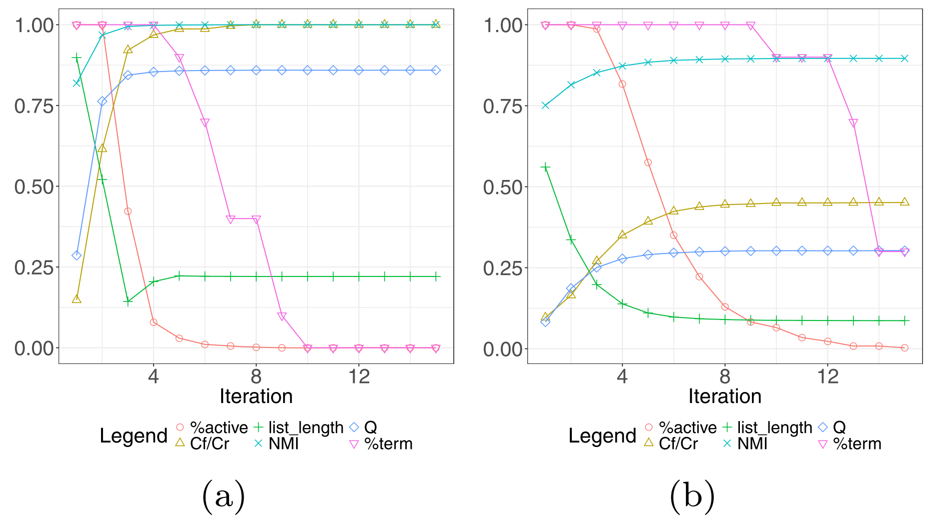


Fig. 5 Experiments on the LFR benchmark. $\mu = 0.1$ (**a**) and $\mu = 0.6$ (**b**) has been used for the two experiments. On the x-axis you can find number of iterations. On the y-axis NMI, modularity, ratio between number of communities found by MemLPA and real communities, percentage of runs that terminated, number of active nodes and average ratio between memory length and node degree. Both experiments have been run 50 times and results averaged

being able to find the correct number of communities. The percentage of active nodes dropped significantly right after the best performance was reached, causing most of the runs to terminate. Average memory length dropped significantly during the first iterations and then stabilized, holding a constant value that is significantly lower than average node degree. For $\mu = 0.6$, as expected, there was a similar behavior but the algorithm converged slower. Surprisingly, the average memory length is lower for $\mu = 0.6$. A possible explanation is that nodes in well defined communities hold very strong labels in their memory, while for loose communities labels are weaker and more likely to be removed by the cutoff operator.

Real-world networks

We conducted similar experiments on a set of real-world networks of different nature. An overview of these networks characteristics is provided in Table 1.

In the first experiment, similarly to "Artificial networks" section for artificial networks, we investigated the advantages of memory and the LPA variations chosen. Figure 6 shows that using a synchronous or asynchronous update did not make a significant change (N_C_S vs N_C) and the cutoff operator did not degrade performance (C_S vs S and N_C_S vs N_S). This allows MemLPA to be scalable, fast and performing. Node preference did not affect performance on unweighted networks significantly, while performance mostly degraded for the weighted ones (N_S vs S and N_C_S vs C_S). A possible explanation is that weight is a more significant factor than neighborhood overlapping when it comes to measuring the similarity between nodes. Additionally, other types of heuristics might be more effective, such as node degree. Implementing memory was beneficial on most networks when compared to the memory-less LPA. In particular, it prevented labels from overpropagating on the *Mail* network where the classical LPA finds a giant community that contains about 95% of the nodes and few very small ones. The only case where the classical LPA obtained better results is for unweighted and undirected networks (*FB* and *PGP*). In the second experiment we compared MemLPA to other memory-based label propagation algorithms. fluidC was not considered since it requires the number of communities as input. MemLPA achieved the best performance on *Jazz*, *Karate*, *UKfaculty* and *USAiports* networks, while still obtaining good results on *Dolphins* and *GPG* networks. Finally, we compared MemLPA to other well-known community detection algorithms. MemLPA was among the most performing algorithms on all networks, obtaining the best results on *Karate* and *UKFaculty* network. Again, MemLPA did not obtain optimal results for unweighted and undirected networks. It must be underlined that modularity may not be an optimal metric, because of the resolution limit and the fact

Table 1 Real-world networks characteristics

	#nodes	#edges	directed	weighted
karate	34	78	no	yes
UKfaculty	81	817	yes	yes
mail	184	2116	yes	no
dolphins	62	159	yes	no
jazz	198	2742	yes	no
USAirports	755	23473	yes	yes
FB	4039	88234	no	no
PGP	10680	24340	no	no

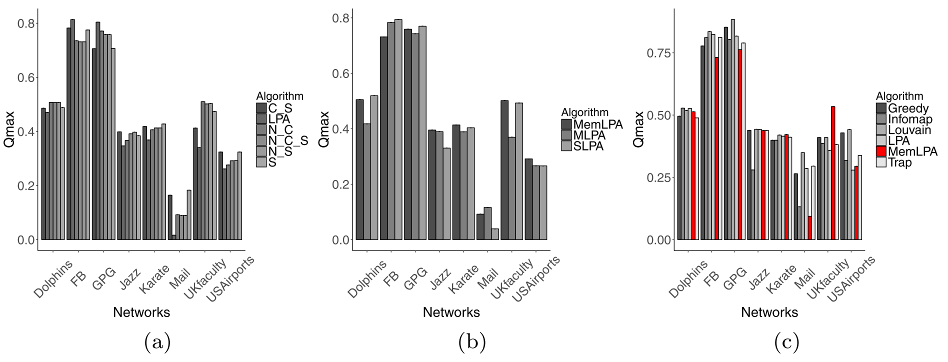


Fig. 6 Experiments on real-world networks. Each barplot represents the results obtained for all algorithms on a specific network. Experiments have been run 100 times and results averaged

that networks may present different community scales. Also, using different resolution limit parameters can affect the results.

Topological analysis

The topological analysis was conducted on the LFR benchmark as supplementary evaluation. We decided to focus on specific values of μ for which the classical LPA starts to fail to identify communities. Benchmark graphs were generated with a number of nodes $N = 1000$, minimum community size $C.min = 10$, maximum community size $C.max = 50$, average degree $K.avg = 20$, maximum degree $K.max = 50$, degree exponent $K.exp = 2$, community size exponent $C.exp = 1$ and mixing parameter $\mu \in [0.55, 0.6]$. Each algorithm was run on each instance of the benchmark. For each community found, all topological metrics presented in 14 were computed and the results averaged on communities having the same size. In order to quantify the agreement between the ground truth and the communities found, we performed a Kolmogorov-Smirnov (KS) test, used to test if two samples are drawn from the same distribution. The KS distance between the two distributions is then computed for each algorithm.

We compared the classical LPA to variations of MemLPA with and without node preference (N), using the ground-truth community assignment as reference. Figure 7, and the KS distance computed between the ground truth and the communities found, show that using memory is beneficial for community size, internal modularity, internal transitivity and scaled density. The classical LPA performs better only for average distance and hub dominance.

The same experiment was performed to compare MemLPA using node preference to other memory-based label propagation algorithms. MLPA was not considered since it generates many disconnected communities and singleton communities for which most of the metrics cannot be computed. Results are shown in Fig. 8. MemLPA achieves best results for community size and hub dominance, and second best results for internal transitivity and average distance.

Finally, we compared MemLPA using node preference to some of the well-known community detection algorithms presented in "Related work" section. We chose Walktrap and Louvain since, in the cluster analysis in "Artificial networks" section, they achieved similar results. Figure 9, and the KS distance, show that MemLPA achieves the second best results only for the average distance.

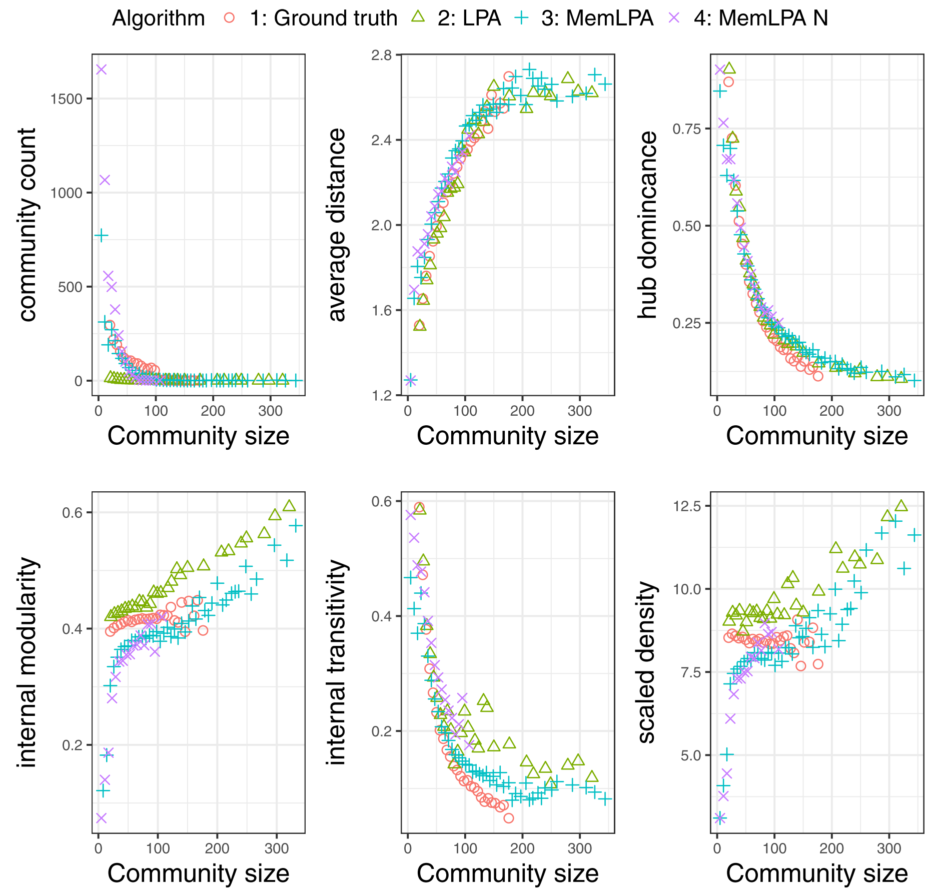


Fig. 7 Experiments on the LFR benchmark. N: node preference. Experiments are run 500 times and results averaged on communities having same size. The KS distance between the ground truth and communities found is shown in the table below.

	LPA	MemLPA	MemLPA N
community size	0.54	0.42	0.34
average distance	0.21	0.60	0.48
hub dominance	0.14	0.50	0.55
internal modularity	0.81	0.60	0.93
internal transitivity	0.47	0.32	0.69
scaled density	0.93	0.63	0.78

MemLPA finds a greater number of smaller communities, which affects the quality of the communities found. When choosing a label from memory, a label that was very frequent in the first iterations but not as much in the last ones will be still selected. A node may select a label from one of the first iterations and a neighboring node a label from the last iterations. As a consequence two smaller communities will form instead of a single bigger one. Finally, for the topological properties, the use of node preference based on node overlapping is not always beneficial, compared to the classical decision rule.

Conclusions

In this paper we proposed MemLPA, a variation of LPA where nodes implement a memory mechanism that allows them to "remember" past states of the network and use a decision rule that takes this information into account. It runs in near linear time, is scalable and only uses local information of the network. We gave an overview on community

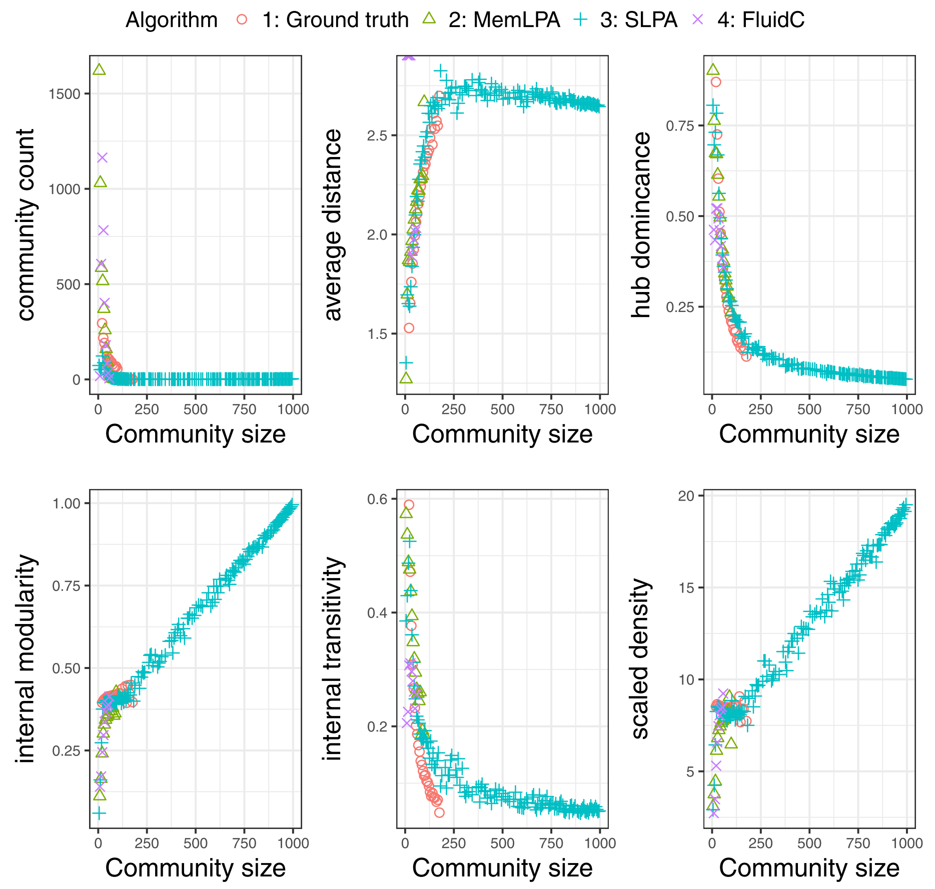


Fig. 8 Experiments on the LFR benchmark. N: node preference. Experiments are run 500 times and results averaged on communities having same size. The KS distance between the ground truth and communities found is shown in the table below.

	MemLPA N	SLPA	FluidC
community size	0.34	0.51	0.46
average distance	0.48	0.79	0.44
hub dominance	0.55	0.69	0.77
internal modularity	0.93	0.76	0.89
internal transitivity	0.69	0.38	0.77
scaled density	0.78	0.75	0.59

detection algorithms, LPA and the variations proposed in the literature. We investigated the advantages of memory and we found that its usage increases performance and prevents labels from overpropagating over the entire network, resulting in a single huge community. We conducted extensive experiments on the LFR benchmark and used NMI and ARI as performance metrics. We tested MemLPA against other existing label propagation algorithms that implement memory to show that it provides better results. We also compared MemLPA to well-known community detection algorithms to show that it outperforms some of them for values of the mixing parameter between 0.5 and 0.8. Then, we conducted experiments on a set of real-world networks of different nature, using modularity to evaluate the quality of the community assignments found, that further confirmed our findings. Finally, we performed a topological analysis on the LFR benchmark, comparing the topological properties of the communities found to the ground-truth community structure. As future work, automatic methods can be used to tune the algorithm to find

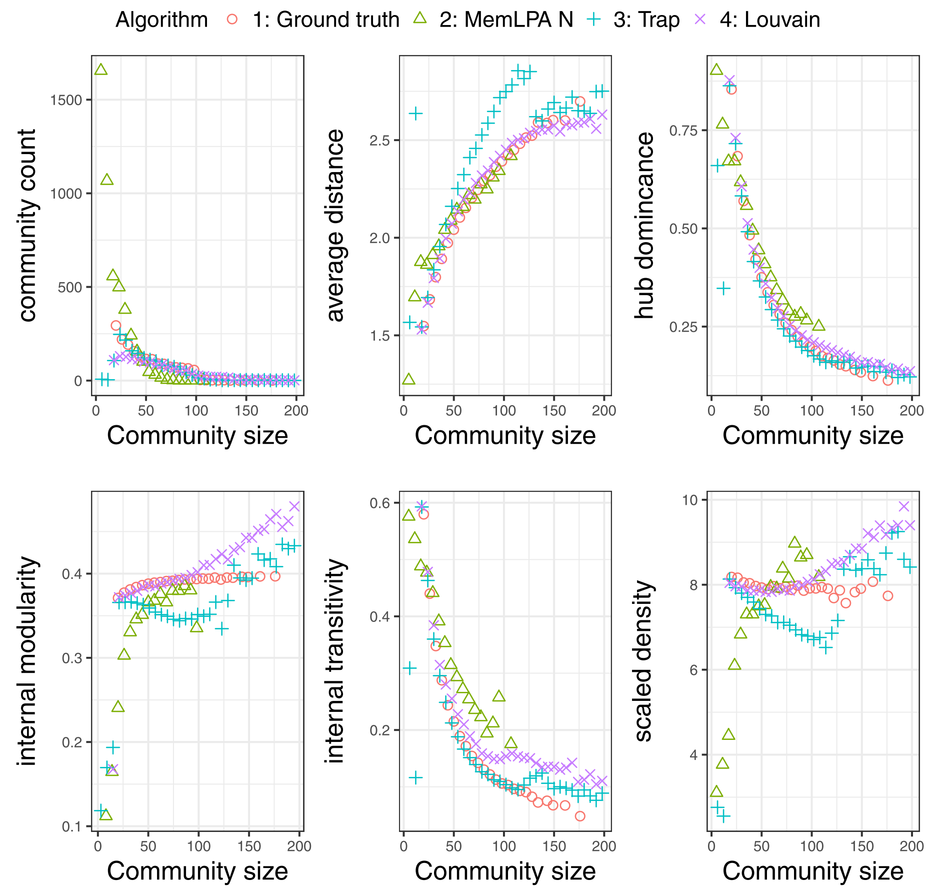


Fig. 9 Experiments on the LFR benchmark. N: node preference. Experiments are run 500 times and results averaged on communities having same size. The KS distance between the ground truth and communities found is shown in the table below.

	MemLPA N	Trap	Louvain
community size	0.34	0.23	0.23
average distance	0.48	0.53	0.22
hub dominance	0.55	0.19	0.12
internal modularity	0.93	0.67	0.58
internal transitivity	0.69	0.22	0.43
scaled density	0.78	0.48	0.52

the best performing variations for LPA, such as initialization method, node preference and termination criterion, as well as the best set of parameters. Different variations may also achieve different results depending on the type of network (Šubelj and Bajec 2011). Also, extending the algorithm to work with overlapping community is a natural consequence when implementing memory.

Abbreviations
ARI: Adjusted rand index; LFR: Lancichinetti-Fortunato-Radicchi; LPA: Label propagation algorithm; MCL: Markov cluster algorithm; NMI: Normalized mutual information; Q: Modularity

Acknowledgements
The authors thank Saharnaz E. Dilmaghani for helpful comments.

Authors' contributions
AMF and MRB conceived the study. AMF implemented the code and conducted the computational research. AMF and MRB analyzed the data and interpreted the results. AMF wrote the manuscript. All authors reviewed and approved the paper.

Funding
This work is supported by the Luxembourg National Research Fund (FNR) (10929115). This work is partially funded by the joint research programme UL/SnT-ILNAS on Digital Trust for Smart-ICT.

Availability of data and materials
The code for the LFR benchmark to generate artificial networks is available at https://sites.google.com/site/santofortunato/inthepress2. The datasets for the real-world networks analyzed during the current study are available at http://konect.cc/networks/.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]C2DH, University of Luxembourg, 11 Porte des Sciences, Esch-sur-Alzette, Luxembourg. [2]SnT, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette, Luxembourg. [3]FSTC-CSC, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette, Luxembourg.

Received: 3 April 2019 Accepted: 10 September 2019
Published online: 29 October 2019

References
Albert R, Jeong H, Barabási A-L (1999) Internet: Diameter of the world-wide web. Nature 401(6749):130–131
Barber MJ, Clark JW (2009) Detecting network communities by propagating labels under constraints. Phys Rev E 80(2):026129
Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech Theory Exp 2008(10):P10008
Brandes U, Delling D, Gaertler M, Gorke R, Hoefer M, Nikoloski Z, Wagner D (2008) On modularity clustering. IEEE Trans Knowl Data Eng 20(2):172–188
Clauset A, Newman ME, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70(6):066111
Csardi G, Nepusz T (2006) The igraph software package for complex network research. InterJournal Complex Systems 1695(5):1–9
Danon L, Diaz-Guilera A, Duch J, Arenas A (2005) Comparing community structure identification. J Stat Mech Theory Exp 2005(09):P09008
Dao V-L, Bothorel C, Lenca P (2018) Estimating the similarity of community detection methods based on cluster size distribution. In: International Conference on Complex Networks and Their Applications. Springer. pp 183–194
Dongen S (2000) A cluster algorithm for graphs
Fiscarelli AM, Brust MR, Danoy G, Bouvry P (2018) A Memory-Based Label Propagation Algorithm for Community Detection. In: International Conference on Complex Networks and their Applications. Springer. pp 171–182
Girvan M, Newman ME (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826
Hosseini R, Azmi R (2015) Memory-based label propagation algorithm for community detection in social networks. In: 2015 The International Symposium on Artificial Intelligence and Signal Processing (AISP). IEEE. pp 256–260
Hubert L, Arabie P (1985) Comparing partitions. J classif 2(1):193–218
Jebabli M, Cherifi H, Cherifi C, Hamouda A (2018) Community detection algorithm evaluation with ground-truth data. Phys A Stat Mech Appl 492:651–706
Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási A-L (2000) The large-scale organization of metabolic networks. Nature 407(6804):651–654
Lambiotte R, Delvenne J-C, Barahona M (2014) Random walks, markov processes and the multiscale modular organization of complex networks. IEEE Trans Netw Sci Eng 1(2):76–90
Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110
Leung IX, Hui P, Lio P, Crowcroft J (2009) Towards real-time community detection in large networks. Phys Rev E 79(6):066107
Liu X, Murata T (2010) Advanced modularity-specialized label propagation algorithm for detecting communities in networks. Phys A Stat Mech 389(7):1493–1500
Newman ME (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6)
Newman ME (2006) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74(3):036104
Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113
Orman GK, Labatut V, Cherifi H (2012) Comparative evaluation of community detection algorithms: a topological approach. J Stat Mech Theory Exp 2012(08):P08001
Parés F, Gasulla DG, Vilalta A, Moreno J, Ayguadé E, Labarta J, Cortés U, Suzumura T (2017) Fluid communities: a competitive, scalable and diverse community detection algorithm. In: International Conference on Complex Networks and their Applications. Springer. pp 229–240
Pons P, Latapy M (2005) Computing communities in large networks using random walks. In: ISCIS, vol. 3733. pp 284–293
Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106
Reginaldo Filho J, Brust MR, Ribeiro CH (2009) Consensus dynamics in a non-deterministic naming game with shared memory. arXiv preprint arXiv:0912.4553
Reichardt J, Bornholdt S (2006) Statistical mechanics of community detection. Phys Rev E 74(1):016110
Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105(4):1118–1123

Scott J (1988) Social network analysis. Sociology 22(1):109–127
Šubelj L, Bajec M (2011) Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction. Phys Rev E 83(3):036103
Xie J, Szymanski BK (2011) Community detection using a neighborhood strength driven label propagation algorithm. In: 2011 IEEE Network Science Workshop. IEEE. pp 188–195
Xie J, Szymanski BK (2013) Labelrank: A stabilized label propagation algorithm for community detection in networks. IEEE, New York
Xie J, Szymanski BK, Liu X (2011) Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: 2011 ieee 11th international conference on data mining workshops. IEEE. pp 344–349
Uzun TG, Da Silva-Filho RJ, Brust MR, Ribeiro CH (2011) Influence of Sha red Memory and Network Topology in the Consensus Dynamics of a Naming Game. In: XXXVIII Seminário Integrado de Software e Hardware (SEMISH). Anais do XXXI Congresso da Sociedade Brasileira de Computação
Yang Z, Algesheimer R, Tessone CJ (2016) A comparative analysis of community detection algorithms on artificial networks. Sci Rep 6:30750

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

► Convenient online submission
► Rigorous peer review
► Open access: articles freely available online
► High visibility within the field
► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com