A Memory-Based Label Propagation Algorithm for Community Detection

Antonio Maria Fiscarelli[1], Matthias R. Brust[2], Grégoire Danoy[3], and Pascal Bouvry[4]

[1] C2DH, University of Luxembourg, 11 Porte des Sciences, Esch-sur-Alzette
SnT, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette
antonio.fiscarelli@uni.lu

[2] SnT, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette
matthias.brust@uni.lu

[3] FSTC-CSC-ILIAS, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette
gregoire.danoy@uni.lu

[4] SnT, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette
FSTC-CSC-ILIAS, University of Luxembourg, 6 avenue de la Fonte, Esch-sur-Alzette
pascal.bouvry@uni.lu

Abstract. The objective of a community detection algorithm is to group similar nodes in a network into communities, while increasing the dissimilarity between them. Several methods have been proposed but many of them are not suitable for large-scale networks because they have high complexity and use global knowledge. The Label Propagation Algorithm (LPA) assigns a unique label to every node and propagates the labels locally, while applying the majority rule to reach a consensus. Nodes which share the same label are then grouped into communities. Although LPA excels with near linear execution time, it gets easily stuck in local optima and often returns a single giant community. To overcome these problems we propose MemLPA, a novel LPA where each node implements memory and the decision rule takes past states of the network into account. We demonstrate through extensive experiments on the Lancichinetti-Fortunato-Radicchi benchmark and a set of real-world networks that MemLPA outperforms most of state-of-the-art community detection algorithms.

Keywords: Network analysis, graph theory, community detection, label propagation

1 Introduction

Real-world networks often exhibit a community structure where nodes in a community are highly connected and few links connect the different communities. The objective of a community detection algorithm is to partition similar nodes of the network in groups, while increasing the dissimilarity between the groups.

This problem is NP-hard [4], 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 [1], natural sciences [11] and social sciences [23], making it a notably active research field.

Several community detection methods have been proposed in the literature such as greedy algorithms that incorporate modularity optimization [3, 5, 9, 16], spectral methods based on modularity matrix [15] and random walk-based methods [18, 22]. In particular, the Label Propagation Algorithm (LPA) assigns a unique label to every node and propagates the labels locally, while applying the majority rule to reach a consensus. Nodes sharing the same label are then grouped into communities. This method runs in near linear time, is scalable and requires only local information, thus it is especially suitable for large networks. On the other hand, it gets easily stuck in local optima and is thus outperformed by more sophisticated algorithms.

The method that we propose is called MemLPA, a variation of the classic LPA where each node implements memory and the decision rule takes past states of the network into account. We also adapt some of the improvements proposed in the literature to our method such as neighborhood preference and termination criterion based on active nodes. The use of memory improves performance and prevents a single label from flooding the network. We conducted extensive experiments on the Lancichinetti-Fortunato-Radicchi benchmark and a set of real-world networks. The algorithm is tested against state-of-the-art community detection algorithms and it outperforms most of them for values between 0.5 and 0.8 of the mixing parameter.

The remainder of this article is organized as follows. Section 2 presents a state-of-the-art analysis on community detection algorithms and label propagation algorithms. Our contribution, MemLPA, is introduced in Section 3 and its performance is analyzed and compared to other community detection algorithms on artificial and real-world networks in Section 4. Finally, Section 5 provides our conclusions.

2 Related Work

In this Section we provide an overview of community detection algorithms that do not require a priori information about the network, such as the number of communities. In particular, we discuss LPA and several variations that have been proposed in the literature.

Girvan and Newman [16] proposed a divisive hierarchical algorithm that removes edges with high betweenness to enhance the separation of communities. This algorithm runs in $O(nm^2)$. A faster version was proposed [5] that iteratively merges nodes into communities to optimize modularity. The complexity for this method is $O(md \log n)$, where d is the depth of the dendrogram. Blondel, Guillaume, and Lambiotte [3] developed a similar method called Louvain. It repeatedly merges nodes into communities that achieve the highest modularity

improvement and builds a new network where nodes represent the communities found. This method runs in $O(n \log n)$. Walktrap [18] defines a similarity between nodes according to the transition probability of random walkers. In fact, random walkers are more likely to stay within the same community. It runs in $O(n^2 m)$ or $O(n^2 \log n)$ on sparse networks. Infomap [22], similarly, models flow patterns in networks using the transition probability of random walkers and runs in $O(m)$. Newman [15] proposed a spectral method based on the Eigenspectrum of the modularity matrix that runs in $O(n(m+n))$ or $O(n^2)$ on sparse networks. Finally, Reichardt and Bornhol [21] interpreted community detection as the minimization of the energy function of a spin model. It runs in $O(n^{3.2})$ on sparse networks.
Many of the algorithms proposed in the literature have downfalls that make them inefficient on large-scale networks: they have high complexity and require global information of the network. To overcome this problem, Raghavan [19] proposed the Label Propagation Algorithm (LPA): it runs in near linear time, is scalable and uses the network's local information only, without the need of optimizing any objective function. It assigns a unique label to every node and propagates the labels locally, while applying the majority rule to reach a consensus. Nodes sharing the same label are then grouped into communities. On the other hand, LPA gets easily stuck in local optima and is thus outperformed by more recent and sophisticated algorithms. Also, a certain label may "flood" the network and create a single giant community. Several variations have been proposed. Clark [2] developed a variation that includes modularity optimization to improve performance. This method was further enhanced by Liu and Murata [14] with a greedy method that allows it to escape from local optima. Leung [13] introduced a decision rule based on node preference, such as node degree, to improve performance and hop attenuation to prevent a label from flooding the network. The algorithm is scalable and still runs in near linear time. Xie and Szymanski [27] proposed another node preference, based on neighborhood similarity, that is related to the clustering coefficient. Šubelj and Bajec [24] elaborated two particular strategies, defensive preservation and offensive expansion, that adapt node preference to focus on core nodes and border nodes of communities. They also found that the network structure affects the effectiveness of node preference and hop attenuation. The algorithm runs in $O(m^{1.19})$ and is highly scalable. Xie and Szymanski [26] also developed LabelRank, a variation of the classic LPA that takes inspiration from the MCL algorithm [8]. Each node maintains a list of label distributions that are propagated through the network. An inflation and a cutoff operator are applied to shorten these lists.

3 MemLPA: A Memory-Based Label Propagation Algorithm

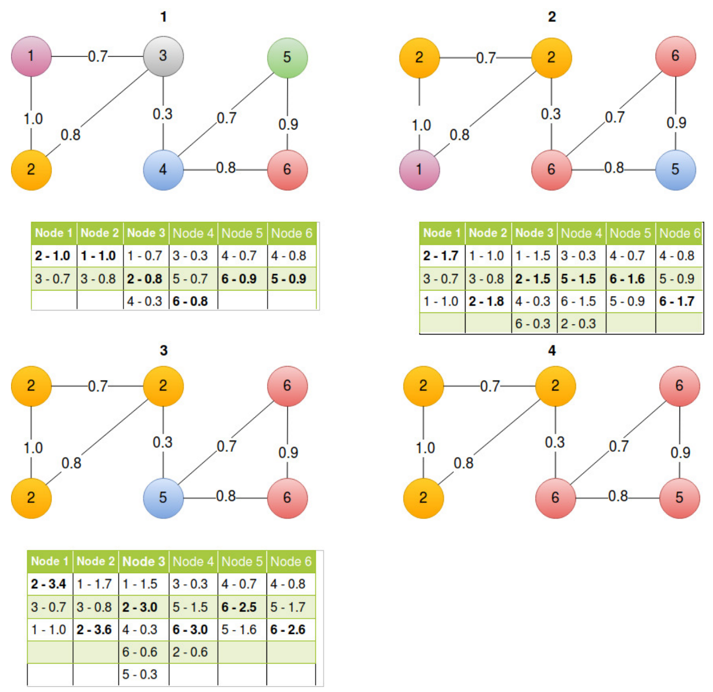
The classic LPA updates nodes' labels according to the current state of the network. During a certain iteration, each node collects its neighbors' labels and selects the most chosen one. At each iteration, all these labels are discarded and new ones are collected. This mechanism makes the algorithm memory-less, since it does not consider past states. In this Section we introduce MemLPA, a variation of the classic LPA where nodes implement memory: the use of memory increases performance with limited increase in complexity and without affecting scalability.

3.1 Algorithm Description

When using memory, labels are not discarded but updated at each iteration. Each node maintains a list of labels where every element of the list contains a counter associated to that label. Initially, each node is assigned a unique label (line 3 of the pseudocode) and their label lists are empty (line 4). At each iteration, each node collects its neighbors' labels (line 8) and updates its label list according to weight (for weighted networks) and node preference (line 9). If a new label appears, a new entry is created in the list, otherwise the counter for the corresponding label is incremented. Each node then selects a label from the list using a decision rule that takes into account the labels' counters, the maximum element in this case (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 state according to the network's state during the previous iteration. It has been shown that a synchronous update can cause LPA to oscillate between two different configurations and an asynchronous update can lead to better results, but in Section 4 we show how the two different update rules affect the convergence of MemLPA. As node preference, we use a heuristic based on neighborhood similarity: we compute the percentage of neighbors that a node shares with another. In Section 4 we show the impact of this heuristic on performance. To speed up the algorithm, we define a cutoff operator to prune label lists (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 that are based on convergence, modularity improvement, active nodes and scarcity of updates. Modularity is based on global information of the network, therefore we decided to use a termination criterion based on the active node list: a node is considered active if, in an attempt to update its label, it chooses a different label. The active node list initially contains all nodes (line 1) and at each iteration a node is removed if it is no longer active or it is added if any of its neighbors becomes active again (line 13). This keeps the algorithm decentralised and speeds up the algorithm significantly. In Section 4 we show that using an active node list allows the algorithm to stop right after NMI or modularity has reached an optimal value.

To our knowledge, the only method that explicitly refers to memory in LPA is the Speaker-Lister Label Propagation Algorithm (SLPA) [28]. We would like to point out how our method is different: in SLPA, each node selects only one label from the collection of labels received from its neighbors, while nodes in MemLPA



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		

Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
2 - 3.4	1 - 1.7	1 - 1.5	3 - 0.3	4 - 0.7	4 - 0.8
3 - 0.7	3 - 0.8	**2 - 3.0**	5 - 1.5	**6 - 2.5**	5 - 1.7
1 - 1.0	**2 - 3.6**	4 - 0.3	**6 - 3.0**	5 - 1.6	**6 - 2.6**
		6 - 0.6	2 - 0.6		
		5 - 0.3			

Fig. 1: Iterations of MemLPA on a weighted undirected graph. Color and node number represent labels. Columns in a table represents label lists maintained by a node.

update their label lists with all the labels received. SLPA uses an asynchronous update rule, while our method is synchronous. SLPA uses a thresholding procedure that is performed on label distributions once the algorithm converges, while MemLPA applies a cutoff operator during each iteration. Finally, SLPA terminates after a fixed number of iterations, while MemLPA's termination criterion is based on active nodes. Some work on consensus dynamics also refers to memory. For example, a non-deterministic version of the Naming Game [20, 25], which is similar in some aspects to LPA, extends the agents with local memory but also uses a shared memory, making it not decentralized, which is the main difference to our approach.

3.2 Complexity

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

- Computing the neighborhood intersection for node preference with k neighbors has complexity $O(k * k)$. Notice that it only needs to be computed once and nodes can store this information.

Algorithm 1: MemLPA

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

- Updating the label list with k new values has complexity $O(k)$.
- Using the cutoff operator has complexity $O(h)$.
- Choosing a new label has complexity $O(h)$.

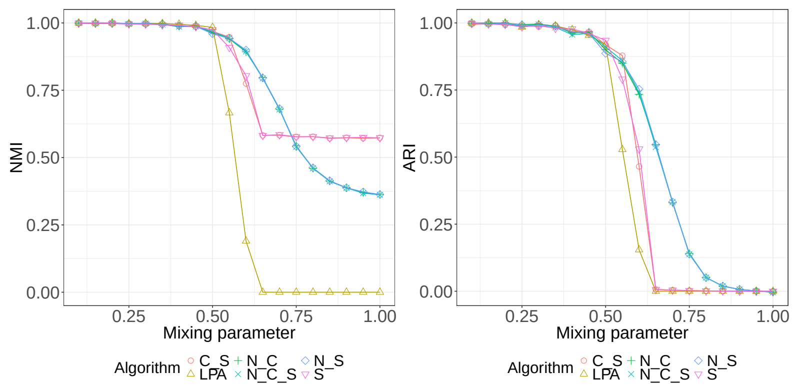
In Section 4 we show how the cutoff operator keeps the average list length constant and significantly lower than the average degree. Iterating on all nodes, the overall complexity is $O(k^2{*}n)$ or $O(k{*}m)$, instead of $O(m)$ of the classic LPA. This means that the complexity of MemLPA is still near linear w.r.t. network size.

4 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 state-of-the-art community detection algorithms to show how it outperforms most of them. We also ran MemLPA to study some of its characteristics that are important for convergence. For the analysis we ran all algorithms on the Lancichinetti-Fortunato-Radicchi (LFR) benchmark [12], 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 the Normalized Mutual Information (NMI) [7, 29] and Adjusted Rand Index (ARI) [10]. We also applied these algorithms on a set of real-world networks of different nature and used the modularity measure to evaluate the quality of the solutions.

4.1 Artificial Networks

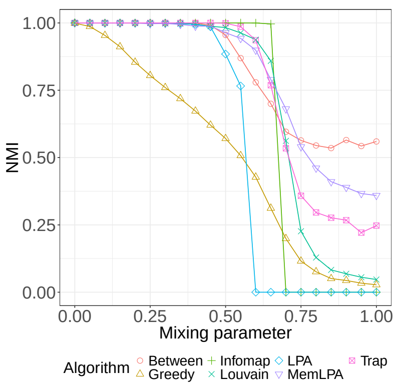
The first set of experiments was conducted on the Lancichinetti-Fortunato-Radicchi (LFR) benchmark to investigate the advantages of the LPA variations



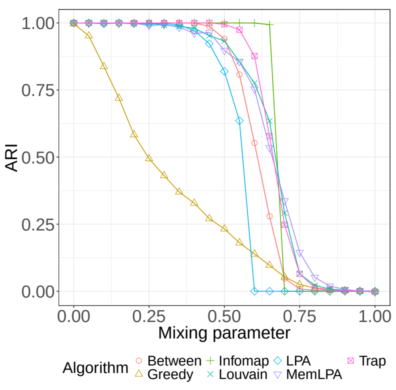
(a) (b)

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

chosen and the use of memory. A mixing parameter μ controls the portion of intra-community and inter-community edges. Node degree and community size are not fixed but drawn from a power-law distribution. Benchmark graphs were generated with 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 classic 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 kept decentralized and scalable without compromising performance. For low values of μ all variations obtained optimal results. The classic version of LPA, the only one not using memory, was the first algorithm to drop in performance for $\mu \geq 0.5$ because, for these networks, a single label overpropagated and created a single giant community. This confirms that the use of memory improves performance and prevents a label from flooding 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 constant values of NMI for $\mu \in [0.7, 1]$. We must consider that the NMI is dependent on network size and number of communities. Therefore we decided to use the Adjusted Rand Index to have a more accurate comparison. As a result, we can see that node preference was dominant for any $\mu \geq 0.5$.



(a)



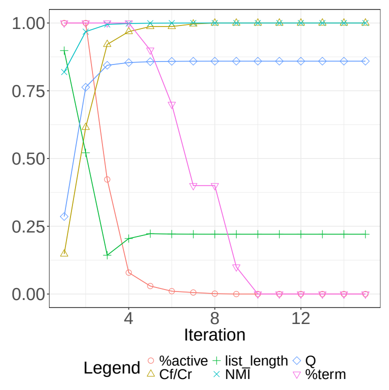
(b)

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

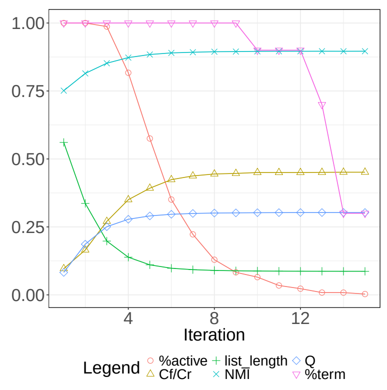
The same experiment was performed to compare MemLPA to other state-of-the-art community detection algorithms. We chose the algorithms described in Section 2 and they are all available in the igraph package [6]. Notice that Eigen was not used because it did not converge on some networks, while Spinglass cannot work with unconnected graphs.

Figure 3 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 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, considering ARI, MemLPA performed slightly better until all algorithms' performance dropped to zero.

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 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 label list length and node degree. Figure 4 shows that, for $\mu = 0.1$, MemLPA increased in performance quickly, 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. The length of label lists, w.r.t. node degree, dropped significantly during the first iterations and then stabilized. For $\mu = 0.6$, as expected, there was a similar behavior but the algorithm converged slower. Surprisingly, the average list length is lower for $\mu = 0.6$. A possible explanation is that nodes



(a)



(b)

Fig. 4: Experiments on the LFR benchmark. $\mu = 0.1$ and $\mu = 0.6$ has been used for the two experiments. Both experiments have been run 50 times and results averaged.

in well defined communities hold very strong labels in their label lists, while for loose communities labels are weaker and more likely to be removed by the cutoff operator.

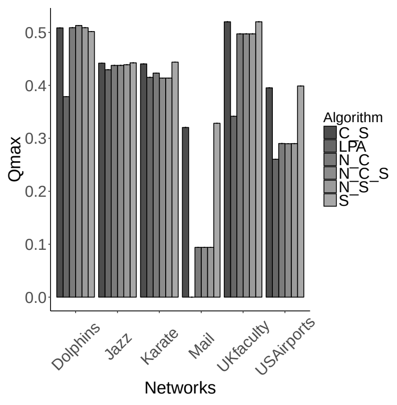
4.2 Real-world Networks

We conducted similar experiments on a set of real-world networks of different nature. An overview of all the networks is available in Table 1.

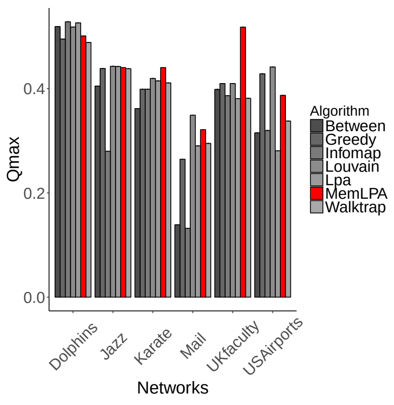
Table 1: Real-world networks characteristics

	#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

In the first experiment, like Section 4.1 for artificial networks, we investigated the advantages of memory and the LPA variations chosen. Figure 5 shows that, as previously seen on artificial networks, 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 or the level of nodes' interaction. Additionally, other types of heuristics might be more effective, such as node degree. Implementing memory was beneficial on all networks when compared to the memory-less LPA. In particular, it prevented labels from overpropagating on the *Mail* network where the classic LPA finds a single huge community. In the second experiment we compared MemLPA to state-of-the-art community detection algorithms. MemLPA was among the most performing algorithms on all networks. In particular, it obtained the best result for *Karate* and *UKFaculty*.



(a)



(b)

Fig. 5: Experiments on real-world networks. Each barplot represents the results obtained for all algorithms on a specific network. Experiments have been run 100 times.

4.3 Discussion

We conducted extensive experiments on the LFR benchmark, investigated the advantages of MemLPA and found that memory improves performance and prevents labels from overpropagating. We compared MemLPA to several state-of-the-art community detection algorithms: it outperforms most of them for values of the mixing parameter between 0.5 and 0.8, while still running in near linear time and using only local information. Then, we conducted two experiments to analyze some of its characteristics at runtime, such as number of iterations before termination, number of active nodes and label list length, using NMI, modularity and number of communities found as performance metrics. We also ran MemLPA on a set of real-world networks of different nature and compared the quality of the solutions found to the ones of all the algorithms considered. With these findings, we can conclude that the use of memory is beneficial and the algorithm is competitive.

5 Conclusions and Future Work

In this paper we proposed MemLPA, a variation of LPA where nodes are seen as agents that interact with their neighbors, implement memory and use a decision rule based on past states of the network. It runs in near linear time, only uses local information of the network and is scalable. We gave an overview on community detection algorithms, LPA and the variations proposed in the literature. We investigated the advantages of memory and we found that its usage prevents labels from overpropagating and increases performance. We conducted extensive experiments on the Lancichinetti-Fortunato-Radicchi benchmark and used Normalized Mutual Information and Adjusted Rand Index as performance metrics. We compared MemLPA to state-of-the-art community detection algorithms to show how it outperforms most of them, especially for values of the mixing parameter between 0.5 and 0.8. We also conducted experiments on a set of real-world networks of different nature and evaluated the quality of the solutions found using the modularity measure. For future work, we plan on implementing other LPA variations and compare them in order to improve MemLPA. We also want to investigate the correlation between network structure and the use of memory. Finally, as suggested in [17], we want to combine topological properties with the clustering metrics already used for a better comparison of the different variations and understanding of community structure.

acknowledgement This work is partially funded by the joint research programme UL/SnT-ILNAS on Digital Trust for Smart-ICT.

References

1. Albert, R., Jeong, H., Barabási, A.L.: Internet: Diameter of the world-wide web. nature **401**(6749), 130–131 (1999)
2. Barber, M.J., Clark, J.W.: Detecting network communities by propagating labels under constraints. Physical Review E **80**(2) (2009)
3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10) (2008)
4. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE transactions on knowledge and data engineering **20**(2), 172–188 (2008)
5. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Physical review E **70**(6) (2004)
6. Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal **Complex Systems** (2006). URL http://igraph.org

7. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment **2005**(09) (2005)
8. Dongen, S.: A cluster algorithm for graphs (2000)
9. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the national academy of sciences **99**(12), 7821–7826 (2002)
10. Hubert, L., Arabie, P.: Comparing partitions. Journal of classification **2**(1), 193–218 (1985)
11. Jeong, H., Tombor, B., Albert, R., Oltvai, Z.N., Barabási, A.L.: The large-scale organization of metabolic networks. Nature **407**(6804), 651–654 (2000)
12. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Physical review E **78**(4) (2008)
13. Leung, I.X., Hui, P., Lio, P., Crowcroft, J.: Towards real-time community detection in large networks. Physical Review E **79** (2009)
14. Liu, X., Murata, T.: Advanced modularity-specialized label propagation algorithm for detecting communities in networks. Physica A: Statistical Mechanics **389**(7), 1493–1500 (2010)
15. Newman, M.E.: Finding community structure in networks using the eigenvectors of matrices. Physical review E **74**(3) (2006)
16. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Physical review E **69**(2) (2004)
17. Orman, G.K., Labatut, V., Cherifi, H.: Comparative evaluation of community detection algorithms: a topological approach. Journal of Statistical Mechanics: Theory and Experiment **2012**(08) (2012)
18. Pons, P., Latapy, M.: Computing communities in large networks using random walks. In: ISCIS, vol. 3733, pp. 284–293 (2005)
19. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical review E **76**(3) (2007)
20. Reginaldo Filho, J., Brust, M.R., Ribeiro, C.H.: Consensus dynamics in a non-deterministic naming game with shared memory. arXiv preprint arXiv:0912.4553 (2009)
21. Reichardt, J., Bornholdt, S.: Statistical mechanics of community detection. Physical Review E **74**(1) (2006)
22. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences **105**(4), 1118–1123 (2008)
23. Scott, J.: Social network analysis. Sage (2017)
24. Šubelj, L., Bajec, M.: Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction. Physical Review E **83**(3) (2011)
25. Uzun, T.G., Da Silva-Filho, R.J., Brust, M.R., Ribeiro, C.H.: Influence of shared memory and network topology in the consensus dynamics of a naming game
26. Xie, J., Szymanski, B.K.: Labelrank: A stabilized label propagation algorithm for community detection in networks. In: Network Science Workshop (NSW). IEEE
27. Xie, J., Szymanski, B.K.: Community detection using a neighborhood strength driven label propagation algorithm. arXiv preprint arXiv:1105.3264 (2011)
28. Xie, J., Szymanski, B.K., Liu, X.: Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: Data Mining Workshops (ICDMW). IEEE (2011)
29. Yang, Z., Algesheimer, R., Tessone, C.J.: A comparative analysis of community detection algorithms on artificial networks. Scientific reports **6** (2016)