Physica A 534 (2019) 122058

Contents lists available at ScienceDirect

Physica A

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

Minireview

Community detection with the Label Propagation Algorithm: A survey

Sara E. Garza *, Satu Elisa Schaeffer

Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, San Nicolás de los Garza, Mexico

HIGHLIGHTS

- The Label Propagation Algorithm (LPA) has caught attention for its advantages.
- Abundant literature finds LPA efficient, scalable, conceptually clear, and simple.
- We review LPA-related proposals, including enhancements and extensions.
- An empirical study of 13,834 variants of LPA is presented.
- Experiments indicate that most enhancements are beneficial in specific scenarios.

ARTICLE INFO

Article history:
Received 27 April 2018
Received in revised form 3 June 2019
Available online 17 July 2019

MSC:
05C82
68R10
62H30

Keywords:
Community detection
Graph clustering
Label Propagation Algorithm
LPA

ABSTRACT

Community detection aims at discovering the structure, behavior, dynamics, and organization of a complex network by finding cohesive groups where nodes (entities) are, in some sense, more similar within the group and groups are in some fashion separated from the other groups. The Label Propagation Algorithm (LPA), which mimics epidemic contagion by spreading labels, is a popular algorithm for this task. Its variants (enhancements, extensions, combinations) seek to improve or to adapt it while preserving the advantages of the original formulation. This survey gathers, classifies, and summarizes LPA-related advances from 2007 to mid 2019. We also experiment with combinations of numerous LPA "building blocks" on two types of small synthetic networks; essentially all of the 13,834 resulting LPA variants have their niche in terms of either higher solution quality or lower computational load with acceptable quality, but none is universally superior.

Contents

* Corresponding author.
E-mail address: sara.garzavl@uanl.edu.mx (S.E. Garza).

https://doi.org/10.1016/j.physa.2019.122058
0378-4371/

1. Introduction

Networks have become pervasive and ubiquitous; they can be found in a wide variety of contexts, from specific to general and from a person's daily life to worldwide phenomena. For example, in a social context, networks model friendships and acquaintances [1–3]; in biology, networks capture metabolic processes in the organism [4,5]. Other contexts include physics (e.g., spin models [6]), medicine (e.g., gene expression [7]), engineering (e.g., power grids [8,9]), information sciences (e.g., telecommunications [10]), and commerce (e.g., recommendation systems [11–15]). The elements of the network are often called *nodes* and the connections among them are referred to as *edges*.

1.1. Community detection

A problem that has caught interest in the past decades consists of finding *cohesive groups* within networks — e.g., detecting groups of friends in social networks [16,17] or groups of web pages with the same topic in information networks [18,19]. The identification of such *communities* then enables other tasks such as recommendation [20–22], marketing [23], load distribution [24], document organization [25,26], and visualization [27,28].

The task of cohesive-group identification in a network is known as *community detection* (in many ways similar to *graph clustering*), whereas the resulting groups are referred to as communities, clusters, modules, partitions, or just groups. No universal, precise definition has been established for what constitutes a community, even though several broad definitions have been commonly accepted, namely the *structural* and the *functional*: the former states that a community is a group of nodes in which the connectivity can be distinguished either from the rest of the network (e.g., tightly knit groups of nodes loosely connected [3]) or from a random network [29], while the latter states that a community is a group of nodes gathered around a common interest, theme, function, or attribute [30–33].

Not only the concept of *community* lacks a universally accepted definition, but also the task of community detection itself has a *multi-faceted* nature: one can minimize *cuts*, maximize internal density (precisely referred to as *clustering*), optimize a *block-diagonal structure* of a connection matrix (stochastic equivalence), or seek for groups of nodes that are in some fashion consistent in their behavior over time in a *dynamical* process, as detailed by Schaub et al. [34].

Thus far, a considerable number of community-detection algorithms have been proposed, and many of these have successfully tackled the different angles of the community-detection problem. The earliest efforts were divisive approaches, which are based on edge removal to split communities apart [35] or spectral properties [36]. The nature of such approaches is global (the network is processed as a whole) and hence is computationally demanding; to overcome this, improvements [37] and local, agglomerative approaches gained popularity. In particular, modularity-maximization methods became mainstream [38]. However, modularity suffers from the resolution limit [39]: small communities go undetected. As a consequence, other alternatives to modularity have been proposed [40] and a plethora of methods have emerged to surpass the success obtained with modularity maximization.

Other concerns (besides finding high-quality disjoint communities with respect to some defined fitness function) include the identification of overlapping and hierarchical community structures. Two seminal approaches devoted to overlapping community detection are the works of Palla et al. [41] and Lancichinetti et al. [42], based on finding k-cliques and local optimization of a density-based function, respectively. An outstanding contribution to hierarchical community detection is the modularity-based technique of Blondel et al. [43]. Overlapping (and several hierarchical) community detection schemes are reviewed in depth by Xie et al. [44], whereas surveys of community detection in complex networks in general include the works of Schaeffer [30], Fortunato [29], Papadopoulos et al. [32], and Fortunato and Hric [45].

Special interest has also been paid to finding *significant* communities in the sense that the communities differ from structures found in random networks in a statistically significant manner [46], as well as blending the results for the multiple executions of community detection algorithms, known as *consensus clustering* [47].

The inception of big data has caused networks to grow massive, noisy, and dynamic; this, in turn, has pushed the state of the art towards approaches that favor scalability, adaptability, robustness, and simplicity (e.g., having as few parameters as possible). An algorithm that has gained popularity for its efficiency, conceptual clarity, and ease of implementation is

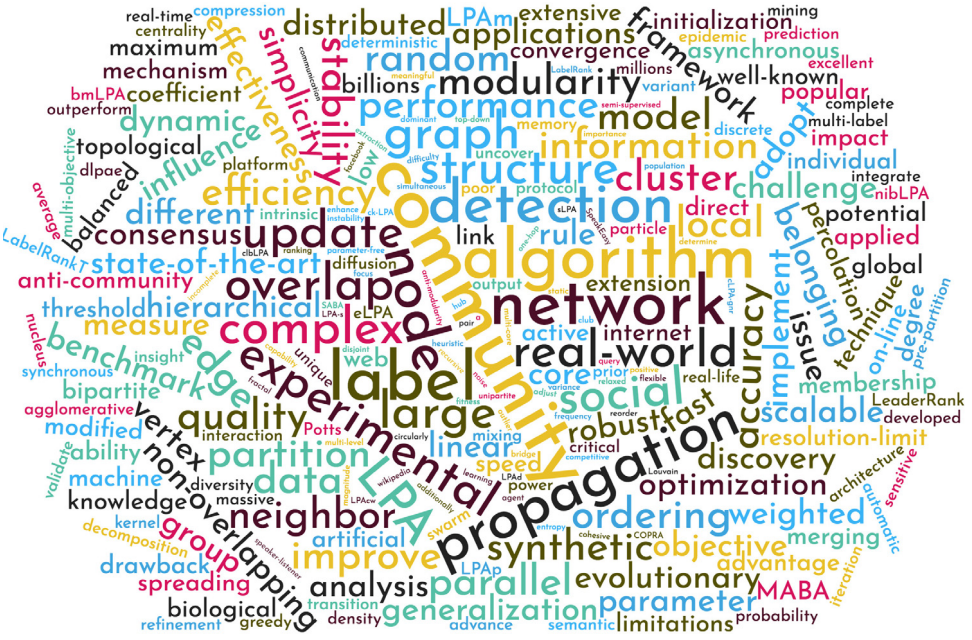


Fig. 1. A tag-cloud made at https://www.wordclouds.com of the abstracts of the works proposing enhancements, extensions, and other uses of LPA cited in this survey. The font size of each term is proportional to the base-two logarithm of its frequency in the combined abstracts.

the *Label Propagation Algorithm* (LPA) [48]. This algorithm, inspired on epidemic spreading, detects a community by the iterative propagation of node labels until convergence (nodes with the same label constitute a community).

LPA fits, in spirit, the fourth facet of Schaub et al. [34]: the label propagation is a dynamical process and ending up with the same label is a shared behavior of the resulting community. Whether these communities end up being internally dense or have small cuts between them is a consequence of how the steps of each variant of the algorithm are tuned, as the way in which label frequencies are computed largely governs the selection of the maximal label and hence guide the outcome of each iteration. Also partitional approaches to the LPA exist [49,50] (see Section 7).

An advantage of the LPA is its asymptotic worst-case complexity: only $\mathcal{O}(n+m)$ steps are needed *per iteration*, where n is the number of nodes and m the number of edges. If a constant number of T iterations are considered, the total time complexity of the algorithm is $\mathcal{O}(T(n+m))$, quadratic with respect to the number of nodes and linear with respect to the number of edges, as $m \in \mathcal{O}(n^2)$ in the worst case. For sparse graphs with few edges ($m \in \mathcal{O}(n)$) and a constant T, LPA can be considered a linear-time algorithm. Raghavan et al. [48] report that after only five iterations, the labels of 95% of the nodes have converged to their final values.

LPA has been widely studied, extended, and enhanced, attracting the attention of the research community in several directions. One popular challenge is to add functionality (an overlapping scheme, for instance) without undermining the time complexity. Also the applications of LPA-inspired algorithms to social network analysis are numerous (cf. [51,52]), including the use of social-network data for user authentication [53], political analysis [54,55], and influence ranking [56–58]. The applied works that cite LPA-inspired literature include analyses of the human brain [59,60], proteins of the E. coli [61], and power grids [62] as well as spectrum sensing [63]. Also the design of attacks against network structures [64] along with stock-market analysis [65] have been developed based on such methods.

1.2. Structure of the survey

This survey gathers, classifies, and summarizes relevant literature regarding LPA. Our methodology for the selection of cited works is detailed in Section 2. Basic notions on graph theory are presented in Section 3, after which LPA is described in Section 4. Sections 5, 6, and 7 present *enhancements*, *extensions*, and other uses of LPA; the costs of implementing these are also briefly enumerated. Fig. 1 shows a tag-cloud visualization of the abstracts of the works cited in these sections to provide a quick glimpse on the topics covered. Open problems and future directions are presented in Section 10 and finally conclusions are given in Section 11.

2. Methodology

Our work selection methodology encompasses three steps: (a) the *identification* of preliminary pertinent material, (b) the *removal* of duplicate material, and (c) the *evaluation* of this material. Preliminary pertinent material was first identified

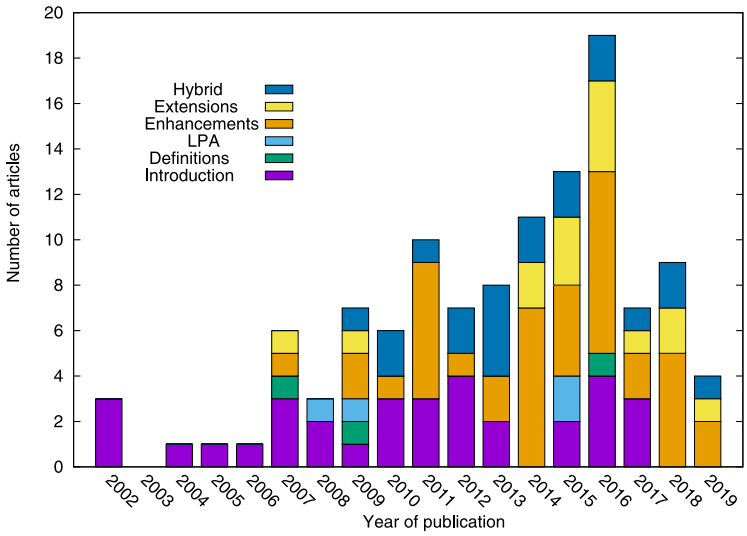


Fig. 2. The number of articles (on the vertical axis) cited in this survey, using the year of publication as the horizontal axis. The bar color indicates the section of the survey that cites the works: introduction in Section 1, definitions of Section 3, LPA itself in Section 4, its enhancements in Section 5, its extensions in Section 6, and the hybrids and other uses of Section 7.

by introducing the queries "label propagation community detection" and "RAK[1] community detection" in three of the relevant databases and search engines: Web of Science, Scopus, and Google Scholar. The selected time period to search was from 2007 to 2019, given that the algorithm was proposed in 2007. From the resulting set of documents, seminal works were detected as the oldest and the most cited among these works, after which a second set of documents was obtained by exploring later publications that cited these seminal works.

The works to include in the survey were then selected based on four criteria, applied in the following order: relevance to the topic, number of citations, membership in the Web of Science, and recency. Works complying with several of these factors at the same time were preferred. If several versions of one same work were encountered (such as pre-print, conference, and journal versions of a single contribution), the most recent one was chosen for inclusion.

A grand majority of the citations to the articles proposing variants and extensions of the LPA are in turn proposals of algorithms themselves. We extracted from Google Scholar the number of citations for each work included in this survey and also reviewed the first result page, scanning for citing works that are applications of the proposed algorithms rather than other algorithms as such. Fig. 2 visualizes the year of publication of the articles cited in this survey, classified by the section in which they are cited. Observing this figure, we can note that LPA has evolved along with the community-detection field by means of enhancements and extensions, and has also blended with other algorithms into hybrid approaches. It is also visible that the majority of modifications made to LPA are enhancements. Table 1 lists the top ten of the included works in terms of their authority score [66] in a citation network where each of the included works is a node and a directed edge represents each citation among these works. In that sense, we can appreciate the most relevant contributions made to the LPA algorithm along with the authors involved.

It is worthwhile to mention the existence of other LPA-related compilations. This is the case of the valuable chapter by Šubelj [77], which is complementary to the present survey. In contrast to this chapter, our survey presents a wider and more recent compilation from the state-of-the-art; our survey additionally

- provides an empirical study that tests 13,834 LPA variants by exchanging different feature options of the algorithm,
- covers topics such as hybrid approaches (that is, combinations between LPA and other algorithms), initialization, and stopping conditions,
- summarizes the main variants and their distinct features (see Table 2), and
- analyzes cited works in terms of their importance and recency (see the current section).

The previously mentioned chapter, meanwhile, focuses also on different applications of LPA, an in-depth analysis of the main variants and equivalences, and provides a distinct empirical comparison.

[1] The LPA algorithm has also received this name in literature for the last names of its authors: **R**aghavan, **A**lbert, and **K**umara. Let us also note that the term "label propagation" is used as well in the field of machine learning where it refers to designating an algorithm for automatically labeling classification examples. Being unrelated to community detection, works referring to this use of the term are not discussed in this review.

Table 1
Top ten LPA-related works in terms of the authority score [66]. Citations refer to Google Scholar citations (global citations) up to May 2019.

Rank	Title	Authors	Year	Journal	Citations	Score
1	Towards real-time community detection in large networks	Leung et al. [67]	2009	Physical Review E	311	0.0532
2	Finding overlapping communities in networks by label propagation	Gregory [68]	2010	New Journal of Physics	702	0.05
3	Detecting network communities by propagating labels under constraints	Barber and Clark [69]	2009	Physical Review E	303	0.0486
4	Advanced modularity-specialized label propagation algorithm for detecting communities in networks	Liu and Murata [70]	2010	Physica A: Statistical Mechanics and its Applications	197	0.0279
5	Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction	Šubelj and Bajec [71]	2011	Physical Review E	102	0.0233
6	SLPA: Uncovering overlapping communities in social networks via a speaker–listener interaction dynamic process	Xie et al. [72]	2011	Data Mining Workshops (conference)	280	0.0174
7	On the equivalence of the label propagation method of community detection and a Potts model approach	Tibély and Kertész [73]	2008	Physica A: Statistical Mechanics and its Applications	71	0.0165
8	LabelRank: A stabilized label propagation algorithm for community detection in networks	Xie and Szymanski [74]	2013	Network Science Workshop (conference)	86	0.0157
9	Robust network community detection using balanced propagation	Šubelj and Bajec [75]	2011	The European Physical Journal B	124	0.0129
10	Detecting community structure using label propagation with weighted coherent neighborhood propinquity	Lou et al. [76]	2013	Physica A: Statistical Mechanics and its Applications	47	0.0091

3. Definitions

The mathematical model for a network is a *graph* $G = (V, E)$ that consists of a set V of n nodes and a set E of m edges; the nodes are often called *vertices* in mathematical texts. When an edge is not reciprocal (i.e., node u connects to node v but not vice versa), the edge is said to be *directed*. Otherwise, it is said to be *undirected*. It is often necessary to assign numerical *edge weights* $w(v, u)$, resulting in a *weighted* graph; otherwise, the graph is said to be *unweighted* and all edges are assumed to have a unit weight. When an edge joins more than two nodes, it is called a *hyper-edge* and the graph is known as a *hyper-graph*. A *self-loop* is a reflexive edge from a node to itself. An undirected, unweighted graph with no self-loops and no multiple edges is called *simple*. The *density* of a graph is the proportion of edges present from the maximum possible amount (that is, if all the nodes were connected to all of the other nodes).

A pair of nodes that share an edge are *adjacent* to each other and are said to be *neighbors*; we denote the *neighborhood* of v as $\Gamma(v)$. The number of neighbors of a node is called its *degree* and nodes with high degrees are called *hubs*.

One graph is a *subgraph* of another if it included a subset of its nodes and such a subset of its edges that both of the endpoints are included in the node subset; a subgraph is *induced* if all such edges are included. A subset of nodes that are all neighbors to each other is called a *clique* whereas a subset of nodes in which none are neighbors to one another is called an *independent set*. The *structural similarity* [78] of nodes v and u (which is based on the cosine similarity of two vectors, proposed by Salton and McGill [79]) is

$$s(v,u) = \frac{|\Gamma(v) \cap \Gamma(u)|}{\sqrt{|\Gamma(v)||\Gamma(u)|}}, \tag{1}$$

which is easy to extend to weighted versions by incorporating the edge weights [80].

When the set of nodes can be partitioned into k disjoint subsets in such a way that the edges exclusively join nodes from different subsets (e.g., author–publication or actor–movie networks), the graph is *k-partite* (when there is only one vertex set, the graph is *unipartite* or *monopartite* and for $k = 2$ the term *bipartite* is used). A *k-coloring* of a graph is a k-partitioning: the coloring problem is about determining whether the nodes can be assigned colors such that no neighboring nodes share a color. Finding the lowest possible number of k is a hard problem for $k \geq 3$, but coloring a graph heuristically allowing for a non-minimal number of colors can be achieved efficiently [81].

A *path* is a sequence of edges that connects v to u and its length is the number of edges in it — the length of the shortest path between two nodes is called their *distance*. A node v is *connected* to another node u if at least one path exists from node v to node u. If all pairs of nodes are connected, the graph is said to be connected.

Given a partitioning of the nodes, the *modularity* [82] of the partitioning is the difference between the proportion of edges that are internal to the partitions and the expected value of the proportion of internal edges in a statistical null model, where this statistical null model is a random network (configuration model) preserving degree sequence; it is a commonly used fitness function for the quality of a community, as the member nodes of structurally good community are expected to be densely connected among each other.

4. The label propagation algorithm

The *label propagation algorithm* (LPA) is a local (bottom-up) partitional algorithm inspired by epidemic spreading; a pseudo code for the basic form of the LPA is given in Algorithm 1.

First, pre-processing can be done on the input graph, such as removing zero-degree singleton nodes or iterative removal of degree-one nodes to clean up the graph. Then, in the initialization phase, a (commonly unique) label is assigned to each node (lines 2–4). After this, the iterative propagation of the labels begins: in each iteration (lines 8–16), the label of each node is *updated*. The computations required during the iteration may be carried out sequentially, but are often easy to parallelize, as discussed in Section 5.3.

Algorithm 1 A basic outline of the LPA with possible enhancements.

1:	Pre-processing	▷ (e.g. iterative leaf-removal)
2:	**for** $v \in V$ **do**	
3:	$\ell(v) \leftarrow v$	▷ Initialization, Section 5.1
4:	**end for**	
5:	**repeat**	▷ Label-Propagation Loop
6:	*synchronously* or *asynchronously*	▷ Section 5.2
7:	*sequentially* or *in parallel*	▷ Section 5.3
8:	**for** $v \in$ (a random permutation of V) **do**	▷ Ordering, Section 5.4
9:	$\Gamma(v) \leftarrow \{u \mid (u,v) \in E\}$	▷ Neighborhood, Section 5.5
10:	$\mathcal{L}(\Gamma(v)) \leftarrow$ *maximum-frequency labels* for $\Gamma(v)$	▷ Frequency, Section 5.6
11:	**if** $\lvert\mathcal{L}(\Gamma(v))\rvert = 1$ **then**	
12:	$\ell(v) \leftarrow \mathcal{L}(\Gamma(v))$.pop()	▷ Flat communities, Section 6
13:	**else**	
14:	$\ell(v) \leftarrow$ random($\mathcal{L}(\Gamma(v))$)	▷ Selection, Section 5.6
15:	**end if**	
16:	**end for**	
17:	**until** $\forall v \in V$: $\ell(v) \in \mathcal{L}(\Gamma(v))$ (with $\lvert\mathcal{L}(\Gamma(v))\rvert = 1$)	▷ Stopping condition, Section 5.7
18:	Post-processing	▷ (incl. DFS for community disambiguation)

Regardless of whether or not the computation is done in parallel, the label updates may be executed *synchronously* (i.e., the new label of each node is computed based on the old labels of the neighbors and then all the label updates are applied, requiring the simultaneous storage of both the old and the new labels) or *asynchronously* (i.e., once a node updates its label, the neighbors that have not yet updated their labels will now observe the new label for the remainder of the iteration) — this is discussed in detail in Section 5.2. In the latter case, the *node update order* (line 8) becomes relevant; the original formulation uses a random order, but alternatives are discussed in Section 5.4.

Table 2
A comparison of the LPA variants according to the features that are modified from the original formulation. The abbreviation "multithr" refers to *multi-threaded*, "comp" to *compatible*, "att" to some type of an *attenuation*, "spec" to a *specific* (non-random) node ordering, "comm" to *community*, "modul" to *modularity*, "add" to *additive*, "thr" to *threshold*, "rnd" to *random*, "prop" to *propinquity*, and "stable" refers to *the labels no longer being altered from one iteration to the next* — "NA" indicates that a characteristic does not apply to the variant in question and "—" that it was not specified by the authors; we hope that all other abbreviations are obvious. The numbers in the neighborhood refer to the number of hops: 0–1 includes the node itself, 1 just the neighbors, and 2 also the neighbors of the neighbors. The most common options for each column were shaded by value for clarity.

Year	Publication	Variant	Comm. Type	Graph	Mode	N. Order	Update rule		Initial	Stopping condition	Neigh.	Paral.	Basis
							Frequency	Selection					
2014	Zhang et al. [151]	LPAp	flat	simple	async	rnd	raw	percol	uniq	stable	1	no	LPA
2009	Barber and Clark [6]	LPAm	flat	simple	async	rnd	constr	max self	uniq	stable	1	no	LPA
		LPAr	flat	simple	async	rnd	constr	max rnd	uniq	stable	1	no	LPA
2013	Lou et al. [86]	LPA-CPN	flat	simple	async	rnd	pref	max rnd	uniq	stable	prop	no	LPA
2014	Weng et al. [137]	ILPA	flat	simple	async	spec	raw	$\sum$ degree	cores	stable	1	no	LPA
2014	Lin et al. [76]	CK-LPA	flat	simple	async	spec	pref	rnd	cores	stable & modul (thr.)	0-1	no	LPA
2015	Sun et al. [129]	CenLP	flat	simple	async	spec	pref	max	uniq	stable	1	no	LPA
2011	Šubelj and Bajec [126]	BPA BPA_L	flat	simple	async	spec	pref	max	uniq	stable	1	no	LPA
2014	Sun et al. [128]	α-NILP	flat	simple	async	spec	pref	max	uniq	stable ratio	tunable	no	LPA
2014	Xing et al. [147]	NIBLPA	flat	simple	async	spec	pref	max	uniq	stable	1	no	LPA
2019	Deng et al. [29]	LPA-FCM	flat	simple	async	spec	pref	min. dist.	cores	threshold	1	no	LPA
2018	Zhang et al. [154]	unnamed	flat	simple	async	spec	pref	2-phase	cores	stable	1	no	LPA
2018	Gui et al. [49]	LBN	flat	simple	async	2-phase	pref	max	cores	stable	1	no	LPA
2018	Zhou et al. [160]	SELP	flat	simple	async	2-phase	pref	max	supervis	threshold	1	no	SLP
2018	Meng et al. [92]	SSLP	flat	simple	async	2-phase	pref	max	supervis	max iter & thres.	1	no	SELP
2016	Chin and Ratnavelu [20]	CLPA-GNR	flat	simple	async	fixed	raw	restr	cores	modul	1	no	LPA
2016	Shang et al. [119]	*unnamed*	flat	simple	async	—	member	max	cores	max iter	1	no	LPA
2014	Zong-Wen et al. [162]	LPAcw	flat	simple	both	rndasync	consensus	max	uniq	stable	1	no	LPA
2012	Cordasco and Gargano [26]	Semi synch. LPA	flat	simple	semi	spec	raw	max self	uniq	stable	1	no	improved LPA
2017	Chin and Ratnavelu [21]	SSCLPA	flat	simple	semi	spec	raw	restr	cores	modul	1	no	CLPA-GNR
2011	Šubelj and Bajec [125]	DDALPA ODALPA	flat	weighted	async	rnd	pref	max	uniq	stable	att 0-1	no	LPA-δ
		BDPA	flat	weighted	async	rnd	pref	2-phase	cores	stable	att 0-1	no	DDALPA ODALPA
2016	Staudt and Meyerhenke [123]	PLP	flat	weighted	async	2-phase	raw	max	uniq	stable & thr.	1	yes	LPA
2011	Soman and Narang [122]	*unnamed*	flat	weighted	sync	NA	pref	max	uniq	stable	1	CPU GPU	LPA
2015	Li et al. [73]	LPA-S	flat	weighted	sync	rnd	raw	prob	uniq	stable	0-1	comp	LPA
2009	Leung et al. [70]	LPA-δ	flat	weighted	both	rndasync	Eq (6)	att	uniq	stable	0-1	comp	LPA
2019	Jokar and Mosleh [60]	BLDLP	flat	directed	async	rnd	raw	max	uniq	stable	1	no	LDPA
2009	Barber and Clark [6]	LPAb	flat	bipartite	async	rnd	constr	max	uniq	stable	1	no	LPA
2009	Liu and Murata [83]	improved LPA	flat	bipartite	semi	rnd	raw	max rnd	uniq	stable	1	no	LPA
2011	Xie et al. [144]	SLPA	overlap	simple	async	rnd	belonging	add thr	uniq	stable	1	no	COPRA
2015	He-Li et al. [55]	DLPA	overlap	simple	async	rnd	belonging	add thr	uniq	—	1	no	COPRA SLPA
2016	Liu et al. [81]	CLBLPA	overlap	simple	async	rnd	belonging	max	cores	comm size	1	no	COPRA
2017	Chen et al. [18]	LPA-E	overlap	simple	async	rnd	pref	add thr	uniq	stable & iter	2	no	COPRA
2011	Xie and Szymanski [142]	*unnamed*	overlap	simple	async	2-phase	pref	max	uniq	stable	1	no	LPA
2016	Liu et al. [82]	ELPA	overlap	simple	async	—	edge labels	max	cores	stable	edge comm	no	LPA
2012	Wu et al. [141]	BMLPA	overlap	simple	sync	NA	pref	add	cores	stable	1	no	COPRA
2016	Zhang et al. [153]	PGLPA	overlap	simple	sync	NA	pref	thr	uniq	max iter	1	yes	LPA
2015	Kuzmin et al. [65]	multithr SLPA	overlap	simple	sync	2-phase	raw	max	cores	max iter	1	yes	SLPA
2016	Peng et al. [102]	*unnamed*	overlap	weighted	async	—	pref	max	uniq	stable	1	no	LPA
2011	Šubelj and Bajec [125]	DPA	hierarc	weighted	async	rnd	pref	2-phase	uniq	stable	att 0-1	no	BDPA
2010	Gregory [45]	COPRA	overlap	bipartite	sync	NA	belonging	add thr	uniq	comm size	1	no	LPA

The new label for each node is chosen among the labels of its neighborhood (the definitions of which may vary, as discussed in Section 5.5). The original formulation chooses the label with the highest frequency; in case that more than one label have the maximum frequency, one is chosen uniformly at random. The chosen label is sometimes referred to as the *maximal label*. Modifications to the way in which a label is selected among the candidates are discussed in Section 5.6.

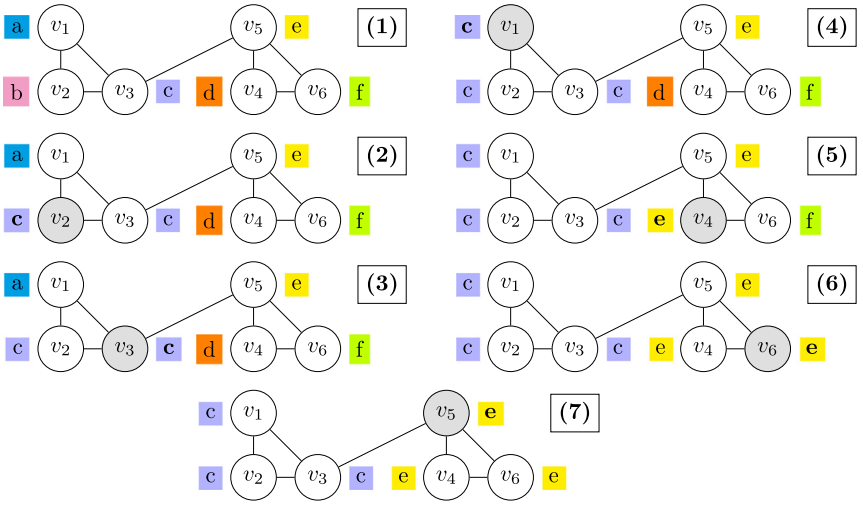


Fig. 3. Example of LPA in asynchronous mode. Node ordering corresponds to $\langle v_2, v_3, v_1, v_4, v_6, v_5 \rangle$. (1) Each node receives a unique label, (2) v_2 chooses label **c** randomly (tie), (3) v_3 chooses label **c** randomly (tie), (4) v_1 chooses maximum frequency label **c**, (5) v_4 chooses label **e** randomly (tie), (6) v_6 chooses label **e** randomly (tie), and (7) v_5 chooses maximum frequency label **e**. Note that the label of the node itself is not taken into account for the frequency calculation. Communities created so far correspond to $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$.

Variants that permit assigning more than one label to each node allow the definition of *overlapping* communities; we discuss these in Section 6.

The algorithm terminates when each node holds the same label with the majority of its neighbors (see Section 5.7). Formally, the maximal labels of v are

$$\mathcal{L}(v) = \arg\max_{\ell} f(\ell, v), \tag{2}$$

where

$$f(\ell, v) = |\{u \mid u \in \Gamma(v) \wedge \ell(u) = \ell\}|, \tag{3}$$

is the number of neighbors of v that have label ℓ, that is, the frequency of that label within the neighborhood. When the algorithm concludes, the communities are deduced from the labels, disambiguating any disconnected label sets: each connected induced sub-graph that shares the same label is considered a community — these can be identified by unmarking all nodes and then executing a DFS from an arbitrary, unmarked node to span over all reachable, unmarked nodes that share its label, iterating over the set of unmarked nodes until none remain. A simple example of how LPA works is given in Fig. 3.

It has been shown that LPA is equivalent to minimizing the Hamiltonian of a simple zero-temperature kinetic Potts model [73]. Barber and Clark [69] view LPA as an optimization problem where the aim is to maximize the number of edges that connect nodes with the same label, and their enhanced version is equivalent to modularity maximization; the resulting community structure is a *local* optimum, as the *global* optimum corresponds to trivial solutions (all nodes with their own label at one extreme, all nodes with the same label at the other, depending on the specifics of the objetive function utilized). It has also been stated that LPA is a simplification of other community detection algorithms [83,84]. Furthermore, LPA is one of the few community detection algorithms that requires no parameters — hence the challenge in extending its functionality without undermining neither the time complexity nor the locality while introducing no parameters and keeping the formulation simple.

5. Enhancements to LPA

Despite being one of the fastest algorithms for community detection, the original version of LPA has ample room for improvement:

Stability The resulting community structure may substantially vary from one execution of the LPA to another [67,75,85]. The main sources of instability are the random node-processing order and the random tie breaking at the update. Proposed enhancements include preferential node orderings (see Section 5.4) and label scoring, as opposed to raw label frequency, for the update rule (see Section 5.6).

Quality The quality of the resulting communities (in terms of a measure such as modularity) may not be very high. Specifically, the rise of a giant or *monster* community that swallows the majority of the nodes, thus occluding any finer community structure within that sub-network, has been observed [67,69,86]. One proposed work-around is synchronous updating (see Section 5.2), and also limits on community size or quality have been proposed [71,87] (see Section 5.6).

Convergence The number of iterations required varies and, on some inputs, LPA does not converge, but instead the label assignment *oscillates*: one node exchanges labels with a neighboring node and the same exchange is reversed in a future iteration, resulting in back-and-forth label exchanges. This may arise from tie-breaking rules or from structural properties of the network itself, such as bipartite structures or ring topologies, and can be dealt with to some extent by the asynchronous mode (see Section 5.2) and alternately updating independent node subsets [85,88] (see Section 5.6); note that this approach has been handled via *graph coloring* techniques. Since graph coloring is known to be NP-hard, the use of a technique for addressing this problem could increase LPA's time complexity — unless it is extremely efficient. Cordasco and Gargano [85] suggest to perform the coloring in parallel.

One-hop horizon LPA only takes into account adjacent nodes (neighbors) for label selection, which is believed to cause loss of topological information and instability [76] (see Section 5.5).

In this section we discuss the addition of features with the aim to solve one of the aforementioned problems or improve LPA in some other sense. Table 2 summarizes the works cited in this section according to the type of improvements proposed. An important consideration is how the proposed enhancements affect the worst-case computational complexity of the resulting LPA variant. Authors often recur to assumptions regarding the network given as input — the network is assumed sparse ($m \in \mathcal{O}(n)$), the number of iterations is assumed a fixed constant or that the number of iterations is low, such as $T \in \mathcal{O}(\log n)$, and that the maximum degree $\Delta = \max_{v \in V} |\Gamma(v)|$ is low, meaning $\Delta \in \mathcal{O}(\log n)$, or a known constant — and formulate the complexity of their proposed enhancement in terms of such assumptions that are not universally valid over all possible input networks, making it difficult to compare and interpret the reported complexities; some report only the complexity per iteration and leave the expected number of iterations as an unattended detail.

5.1. Initialization

An initialization option that results additional to assigning each node a unique label consists of pre-computed communities. Existing knowledge of pre-computed communities can be used to assign the initial labels in such a manner that each community (as opposed to each node) is assigned its unique label. These pre-communities are often called *cores* or *kernels*. The computation of such cores may be expensive (many common quality measures for communities are NP-hard to optimize [89]), for which low-cost constructive heuristics are preferred. The incorporation of cores has two general purposes in the literature: pre-processing for quality-increasing and incorporation as an integral part of the specific LPA variant.

The usual procedure for core extraction consists of two phases: initial structure extraction and expansion. Initial structure extraction consists of detecting a particular kind of structure within the network, either seed nodes [78,90], edges [91], or small sub-graphs (e.g. *dense pairs* [92,93]) — just to mention some examples; this structure normally counts with a special property, such as a high degree [94,95] or a short distance among vertices [96]. After extraction, expansion takes place, since nodes are associated to the structure, thus conforming cores. Two options exist: either all remaining nodes in the network must adhere to cores or only some nodes under certain rules. In the latter case, it is possible to exclusively assign a label to those nodes that are in a core, thus leaving singleton nodes unlabeled [90]; labels are assigned to these nodes at later stages of the algorithm. Expansion rules include allowing a node to be added if it has a certain number of neighbors within the core created so far [97]. Cores can have a pre-defined size and node degree, with the cost of introducing these two as parameters into the LPA. It is also possible for seed nodes, instead of following an expansion stage, to leverage another task, such as the detection of boundary nodes between communities [98]. Note that finding cores is similar to finding cliques, which is known to be NP-hard. In that sense, the inclusion of cores into an LPA variant could increase time complexity. In any case, it would be desirable to extract relaxed structures (in comparison to cliques) with an efficient procedure to avoid undermining LPA's time complexity.

Another initialization option, besides cores and node unique labels, consists of *manually* assigning particular labels to particular nodes; these labels are then propagated to the rest of the nodes using the update rule. Variants following this strategy are known as *semi-supervised* [99–101].

5.2. Synchronization

There are two possible modes for label update: synchronous and asynchronous. This is a common choice in multi-agent systems — either each agent changes its state based on the *previous* states of its neighbors (meaning that both the old and the new states need to be known simultaneously), which is the *synchronous* mode, or each updates its states based on the present observable states of the neighbors (whether or not they have been updated yet for this iteration), which is the *asynchronous* mode.

The asynchronous mode can be even relaxed to dispense with the notion of iterations — one node could update its label several times while another makes no updates at all or just a single one. The latter version makes parallel implementations easier, as there is no need to verify whether the neighbors have already updated their labels, but also makes the theoretical analysis of the convergence of the algorithm much less straightforward. Most of the LPA variants operate under the asynchronous mode (see Table 2).

The asynchronous mode is often used to avoid oscillation at the cost of losing stability and parallelization capability [67]. Variants that address oscillation usually combine the advantages of both modes into semi-synchronous versions of LPA, where these usually involve the synchronous update of only a selected group of nodes [87] or a group of nodes at a time (such as independent sets) [85].

5.3. Parallelization

The computations required by LPA are *local* in the sense that each node can carry out its update based on information of its neighborhood, without knowledge of the rest of the network. This makes it easily parallelizable, with the main challenges being dependency handling and load balancing. The proposed parallel versions have demonstrated a considerable performance speedup while maintaining quality. These versions conventionally assign groups of nodes to threads. Determining these groups is, nevertheless, usually an issue which has been resolved, for example, with guided scheduling [102], parallel bitonic sort [103], and pre-partitioning of the graph [104]. Frameworks such as MapReduce and Spark have also been employed to address scalability [105].

5.4. Node ordering

The original LPA processes the nodes at each iteration in a random order. The ordering of the nodes (also known as *label propagation strategy*) affects the results: it has been observed that nodes updated last have a smaller probability of propagating their labels [75]. Furthermore, any unfairness in the propagation may distort the resulting community structure, including the rise of a monster community (see an example in Ref. [86]). To mitigate this problem, *specific* (non-random) orderings have been proposed. Because these orderings (and related solutions) have generally been proposed in combination with modified update rules, specific approaches will be discussed in Section 5.6.

Another feature related to node ordering is *phasing*, which consists of forming different node groups to be updated or initialized in a particular order (the group could also not be updated at all). For example, a group of *active* nodes could be determined to be the only ones to update [106], or nodes could be split into *assigned* and *not assigned* in a semi-supervised variant (and the latter are updated last) [99], or into *core* and *boundary* nodes, where the former are labeled first [98], or disjoint subsets of nodes in bipartite networks that take turns. We refer to this node ordering as *2-phase* in Table 2.

5.5. Neighborhood definition

The basic version of the LPA updates node labels in terms of their neighbors using the traditional sense of adjacent nodes, under which scheme the algorithm is essentially local — property known as the *one-hop horizon*. Note that whether or not the node itself is included in the neighborhood (a 0–1 hop neighbor) may well affect the propagation of the labels. Consequently, using other hop counts and defining the label frequencies within sub-networks centered at the node of interest in various ways (for example by executing a breadth-first search up to depth k) produces variants of the algorithm. In particular, using the one-hop neighborhood results in difficulties related to stability and performance, for which alternatives to the neighborhood definition have been proposed.

For example, several works consider all nodes at distance ≤ 2 [76,107], whereas more complex variants define a tunable parameter for the distance up to which the near-by nodes are included upon computing label frequencies, also using this distance as a score for node (i.e. neighbor) preference in the update rule [108]. Although these variants solve the one-hop horizon issue, the cost lies on losing LPA's local nature. Another cost, yet, is the increase in time complexity whenever two hops or more are used for neighborhood determination.

5.6. Update rule: frequency computation and label selection

The update rule (also known as *node preference*) in the original LPA consists of selecting the label with the maximum frequency among the updated node's neighbors, with ties being resolved via random selection; each neighbor is given the same importance (preference) in the computation — frequency being thus computed in a *raw* manner. This update rule has been identified as one of the sources for instability, since random tie breaking may result in different outcomes at each run and also in an unwanted preference towards particular neighbors for the updated node (which, in turn, can degenerate in the monster community or simply in suboptimal partitions). Hence, the principal enhancement for the update rule consists of providing a weight or score for each neighbor to minimize the need of tie breaking (we refer to this as *preferential label frequency computation*). Other enhancements obey the prevention of the monster community, conditional update, and outlier detection. Preferential label frequency computation is generally coupled with a particular node ordering that takes advantage of the metric used to modify the computation.

A common approach for preferential label frequency computation is the definition of a *node-preference function* $p(u)$ that assigns a weight to each neighbor such that

$$\ell(v) = \arg\max_{\ell} \sum_{u \in \Gamma(v,\ell)} p(u), \tag{4}$$

where

$$\Gamma(v, \ell) = \{u \mid u \in \Gamma(v) \wedge \ell(u) = \ell\}, \tag{5}$$

that is, the set of neighbors of v that presently hold the label ℓ. A logical choice for $p(u)$ has been degree [67,92,109] or a degree-based metric [110], with nodes additionally ranked either in ascending or descending order. Degree centrality, eigenvector centrality, and clustering coefficient [71] have also been used. Other types of centrality have been proposed as well. For example, preference is given to density peak nodes (density being defined as a function of weighted degree and structural similarity [80]), this metric serving additionally as a measure of centrality for node ordering [111]; a similar approach consists of employing Leader Rank [112] for core node extraction and preference. More complex metrics for node preference and ordering include k-shell values [113,114] and α-neighborhood impacts [108], where the α-neighbor of a node lies within α hops and its preference is calculated based on the preference for $\alpha - 1$ for $\alpha \geq 1$, setting the preference for $\alpha = 0$ to one. In addition to these metrics, Lin et al. [97] propose node preference and ordering based on three factors: a base weight, the distance of the node to its core, and the centrality of the node (calculated as its normalized degree).

Even though a plethora of functions can be used for computing label frequency in the update rule, the choice of a preference function and the adjustment of any possible parameters depends on network topology. Furthermore, other alternatives exist for addressing preference and ordering issues; for instance, Šubelj and Bajec [75] highlight an inverse relationship between node ordering and preference; this situation is balanced by increasing the preference for nodes that are updated last.

With node preference and ordering, it has been shown that LPA gains robustness and improves quality [75]. The cost of implementing these enhancements sometimes, nevertheless, lies in losing the local nature of LPA, as global calculi (centrality, for example) have to be performed over the entire network.

Another issue related to the update rule is the possible formation of the monster community (described in Section 5). To prevent the rise of this structure, the observed general strategy consists of enforcing rules and constraints over label selection and update. Leung et al. [67], in a first attempt, replace the original update rule with

$$\ell(v) = \arg\max_{\ell} \sum_{u \in \Gamma(v,\ell)} s(\ell(u)) p(u)^k w(v, u), \tag{6}$$

where $s(\ell(u))$ is a *hop score* for label $\ell(u)$ and k is a tuning parameter (initially, each hop score is set to 1). The hop score is iteratively penalized by a δ attenuation ratio as the label traverses the network:

$$s(\ell(i)) = \left(\max_{j \in \Gamma(v,\ell)} s(\ell(j))\right) - \delta. \tag{7}$$

With hop attenuation, labels do not spread too far, thus avoiding the monster community. To make δ adaptive, it can be set according to the current number of iteration. Šubelj and Bajec [71] extend this idea by initially setting δ to 0, and making it change either according to the number of nodes that changed their label or according to the number of communities that disappeared. Although no formal evidence is provided, the authors state that both strategies successfully cope with the monster community.

Barber and Clark [69] treat the monster community as the global optimum of the objective function that pertains to the LPA cast as an optimization problem. To overcome this issue, two constraints are proposed for unipartite networks: one that attempts to produce communities with balanced sizes and another one that attempts to produce communities with similar degrees (this last one, equivalent to modularity maximization). Because the constrained LPA variant tends to get stuck in local optima, it has undergone several enhancements (see Section 7).

The monster community has also been addressed by modifying the update rule such that small communities are preferred [86]; in other variants, communities that surpass a given threshold (e.g. a particular internal degree) are exempted from propagation [87]. While these alternatives have helped, the cost in general has been the introduction of parameters in the algorithm and overhead in the process, which depends on the technique used.

Another enhancement related to the update rule is *conditional update*. This enhancement, proposed by Xie and Szymanski [106], consists of exclusively updating nodes that would alter the label in the current iteration (called *active* nodes). The purpose of the conditional update is to no longer revise the labels on nodes that undergo no label changes within their neighborhoods. This kind of update has proven to be useful in certain contexts, such as temporal graphs [115] and structures with convergence issues [116].

Semi-supervised Evidential Label Propagation variants [99,101] use mass functions from belief theory to, among other things, detect outliers — that is, nodes that do not belong to any community. Each neighbor is seen as a source of information with a degree of reliability (this degree being calculated in terms of node similarity); the label of the neighbor with the greatest calculated mass is selected as the target node's label. If the calculated mass lies below a threshold, the node is classified as an outlier — hence the need of a introducing a parameter (outlier threshold).

5.7. Stopping condition and post-processing

The original stopping condition of the LPA is formulated in terms of node labels no longer changing (several works relax this condition by using a threshold in the proportion of changed labels [108]). However, other stopping conditions have also been proposed, such as reaching a maximum number of iterations [78,105] and stability in community sizes [68,90]. Terminating the algorithm after modularity ceases to increase is additionally considered as a stopping criterion [97]. When using LPA as part of consensus clustering, the stopping condition depends on variation between consensus weights [117]. With regard to additional post-processing operations, several variants merge the resulting communities either to remove nested groups [68] or improve modularity [87,109].

6. Extensions of LPA

The original LPA is restricted to the detection of partitions in simple networks. In this section, we discuss the *extensions* to LPA to convert it into an algorithm for other types of networks and an algorithm for other community types. The original LPA works as such on simple, undirected, and unweighted networks and produces flat, disjoint communities. As shown in Table 2, some of the variants discussed in the previous section already admit bipartite networks and produce overlapping community structures, for example; the present section covers some additional approaches as well as characteristics that none of the LPA enhancements take into account.

Bipartite networks Oscillation in bipartite networks has been addressed by separately updating each node subset (as proposed by Liu and Murata [118]); Cordasco and Gargano [85] formally prove that their semi-synchronous variant (generalized also for unipartite networks) of this method converges under several stopping criteria. The computation on each side can be done in parallel if desired, and this can be trivially extended to k-partite networks using k-core parallelism. In addition, k-partite hyper-graphs have been processed using an LPA variant that adapts the objective function of compression-based community detection [119] into the node preference function for the update rule.

Weighted networks Generalizing the update rule of Eq. (4) for weighted networks is straightforward: edge weight can be incorporated in label selection [67,115] or as part of centrality calculation for node ordering [111,120]. Furthermore, weighted LPA variants have been employed for solving the one-hop horizon problem (see Section 5.5) [76], consensus clustering [117], and overlapping community detection [121]. Note that using consensus clustering involves executing LPA several times, which potentially increases the overall time complexity.

Directed networks Variants for directed networks are scarce and often only take into account incoming edges in the neighborhoods for label selection [115,122] or sum weights in both directions [67]. More recently, edge direction has been cast as a kind of node preference in terms of in and out-degree, and the update rule has been modified to distinguish between in-neighbors and out-neighbors with the same label, where the higher preference is selected as representative for the label [123].

Labeled networks Kianian et al. [124] propose an extension for social networks represented with the Resource Description Framework (RDF). This extension grabs labels from tags that are acquired by user posts on specific topics, adding RDF-specific pre and post-processing as well as different stopping conditions; the pre-processing stage creates the networks and assigns node labels, while the post-processing stage refines tags according to an ontology.

Temporal networks Variants for *dynamic networks* usually rely on conditional update. LabelRankT [115], for instance, only updates nodes that have changed between two consecutive snapshots, where the changes considered are node or edge additions and deletions. Liu et al. [125] consider the current network snapshot and the previous one (i.e., times t and $t-1$) for node preference in the update rule, where the priority of each snapshot is regulated by a parameter α. More recently, Sattari and Zamanifar [126] propose to incorporate the Cascade Information Diffusion Model to favor the creation of new communities, as opposed to the inclusion of new nodes in already existing communities. Clementi et al. [127] work with Intermittently-Connected Mobile Networks (ICMNs) by means of a *dynamic planted bisection model*, which consists of different temporal snapshots with intermittent edges and only two existing communities. A distributed five-phase protocol based on label propagation is proposed (source-node label propagation, unlabeled node processing I and II, controlled saturation, and simple majority vote), accompanied with a formal analysis that reveals a logarithmic bound over the studied model and over more relaxed classes of networks.

Cyclic structures Ring topologies, which are prone to oscillation issues, have been addressed by means of a probabilistic LPA variant with conditional update and label assignment with roulette selection [116].

Singleton communities In the semi-synchronous variant by Chin and Ratnavelu [87], nodes are divided into two non-conflicting groups: unclustered and clustered nodes. While the first group is processed synchronously, the second is processed asynchronously.

In addition to adapting to different kinds of input networks, extensions for different kinds of community detection schemes have also been proposed. For instance, with regard to *overlapping* community detection with LPA, the three main approaches are COPRA, SLPA, and overlapping-node detection methods.

***Community Overlap Propagation Algorithm* (COPRA)** The LPA variant proposed by Gregory [68] allows multiple labels per node by assigning a *belonging coefficient* to each of these. A node v adopts, synchronously, the labels of its neighbors and calculates its *belonging coefficient* b for community c in iteration t as

$$b_t(c, v) = \frac{1}{|\Gamma(v)|} \sum_{u \in \Gamma(v)} b_{t-1}(c, u). \tag{8}$$

Only labels with a coefficient above $1/k$ (where k is a pre-defined parameter) are kept, unless no labels comply with the former, in which case the label with the highest coefficient is chosen (ties are resolved randomly). We refer to these threshold-based labelselection strategies as *additive* (see Table 2).

***Speaker–Listener Label Propagation Algorithm* (SLPA)** In the LPA variant proposed by Xie et al. [72], each node is equipped with a *memory* for storing multiple labels. During label update, a node acts as a *listener* and its neighbors as *speakers*; the listener accepts one label from its speakers according to a *listening rule* and, conversely, the speakers send one label to their listener according to a *speaking rule*. The proposed listening rule consists of adopting the most frequently received label, while the proposed speaking rule consists of selecting a label randomly with probability proportional to the label's frequency in memory. Labels with relative frequency above a threshold are held as final labels.

Overlapping-node detection The identification of overlapping nodes consists of applying a metric (either local or global) over the nodes and evaluating whether the resulting value exceeds a given threshold — in a positive case, the node is considered overlapping. Metrics defined for this purpose include neighbor *purity* [67], which consists of the number of different communities the node participates in, and weight variance [121]. To calculate neighbor purity, community co-occurrence can be obtained via multiple LPA runs [128], which could potentially increase running time.

Other In the *dominant label* approach [129], which is based on COPRA and SLPA, every neighbor awards its label with the highest belonging coefficient (called the *dominant label*) to the updated node, but the coefficients are weighted according to neighbor confidence (calculated as normalized neighborhood similarity) and an overlap rate. The final labels for a node v are chosen as those above a threshold of $1/|\Gamma(v)|$. Lately, Lu et al. [130] have incorporated a *preferred historical dominant label* to improve stability; in case of ties with the maximum belonging coefficient, the dominant label of the previous iteration is selected. Other recent overlapping LPA variants are based on epidemic spreading [131] and activation spreading [132]; the latter, which is based on SLPA, initializes each node label with an *activation value* that allows the given label to traverse the network in a breadth-first fashion according to a *decay factor* (when the activation value is below the factor, it cannot be further spread during the initialization phase). By doing so, each node acquires multiple labels, which are updated according to a rule similar to the one used by SLPA.

Let us note that, thus far, all overlapping variants introduce parameters. This incurs in an extra cost with regard to the original LPA.

Hierarchical LPA variants identify nested communities. One common approach has been to alternately run the algorithm and create a super-network with the resulting communities as meta-nodes and the number of inter-community edges as edge weights [67,120]; a variety of metrics can be used to calculate the super-network weights [133]. A similar approach consists of alternately inducing a sub-network from each detected community (similar to a "drill-down") and applying the algorithm over each induced sub-network [134]. Other approaches include gradually relaxing the hop attenuation parameter (Eq. 6) [67] and combining LPA with classical hierarchical clustering [135].

7. Hybrid approaches and other uses of LPA

LPA has been combined with other algorithms to generate more powerful *hybrid approaches*: MSG [70] blends LPAm [69] with a multi-step greedy algorithm to escape local maxima by merging communities to increase modularity (this combination being termed LPAm), whereas meta-LPAm+ employs the Record-to-Record Travel metaheuristic [136] (similar to simulated annealing) to find better solutions than LPAm [137]. LP&BRIM [118] carries out community detection in large-scale bipartite networks, using a semi-synchronous LPA for finding an initial partition; let us note that the worst-case time complexity of this hybrid algorithm is $\mathcal{O}(n^2)$. DEMON [138] is a local overlapping algorithm that uses LPA at its core to detect communities in massive networks. LabelRank [74] combines LPA with the Markov Cluster Algorithm [139]: a label distribution is maintained at each node and labels are updated to strengthen the strong and weaken the weak, keeping only labels above a threshold and performing conditional update. Recently, LPA has been combined with

the Density Peaks Clustering algorithm [140] to overcome the latter's cluster assignment sensitivity [141]; however, let us note that this combination introduces extra parameters.

LPA has also been widely adopted for community detection with *evolutionary computing* (e.g. swarm intelligence) and memetic algorithms — which combine the former with local search. In these contexts, LPA is normally used as an initialization strategy to produce a high-quality first population [122,142], specially in multi-objective problems [143]. In other cases (also within the community detection context), LPA has been used as the local search strategy in memetic algorithms [144] and to implement specific operators, such as mutation in fireworks swarm algorithms [145]. LPA has been combined, as well, with ant algorithms and simulated annealing to overcome the resolution limit by propagating labels probabilistically using modularity as the node preference function [146]. Conversely, LPA has also benefited from evolutionary algorithms — for example, by utilizing a swarm of agents that traverse the network to propagate labels [147]. Even though there exists a synergy when LPA is combined with evolutionary algorithms, the extra cost is the aggregation of a number of parameters, since this kind of algorithms have a non-negligible number of tuning parameters.

Outside the community-detection context, Boldi et al. [148] propose LPA for network compression by ordering nodes such that access to network information is facilitated; they use LPA in combination with the Absolute Potts Model [149] to obtain communities of different granularity, such that the labels of these communities define node order — nodes with the same label being placed close to each other. In the context of network partitioning (a problem similar to community detection but with a different objective), LPA has been used to iteratively coarsen massive networks until an appropriate partitioning algorithm can be applied [50]; in addition, Ugander and Backstrom [49] propose an LPA variant to network partitioning that is constrained to produce balanced partition sizes: starting with a random partition, linear programming is applied to calculate the optimal amount of nodes to conveniently move from one partition to another.

Chen et al. [150] address the problem of *anti-community* detection: identifying groups of nodes where the majority of the edges fall outside the group (e.g. in cases of bipartite networks). Their LPA variant propagates labels onto non-connected nodes and breaks ties by choosing the smallest label (by some established ordering). Zhao et al. [151] use LPA for *influence maximization*, where this problem consists of finding a node subset of size k in which, under a specific diffusion model, the expected number of reached nodes is maximum in a spreading process.

Many of the aforementioned works use LPA for pre-processing, but techniques inspired in LPA can also be applied to community post-processing [152].

Lately, LPA has also been applied for telephone fraud detection [153]. In this case, fraudulent keywords and text from phone calls are modeled as network nodes, and an edge is present between these when one of the fraudulent keywords was used into a phone call. A weighted version of LPA is used, where selection probability at the update is influenced by edge weight, hence causing fraud keywords and call words to have the same label when sharing heavy weights.

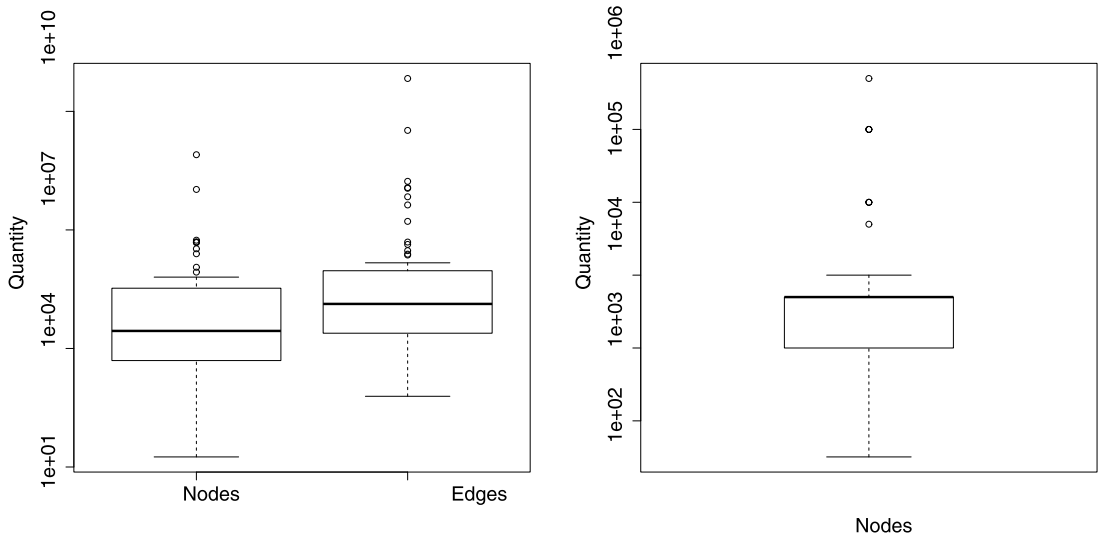
8. Resolving community detection issues with LPA

Section 5 describes endemic issues that have been addressed by LPA variants, such as stability and convergence. Nevertheless, issues properly related to community detection also need to be handled, especially since the inception of social media and big data, which have led to massive, noisy, dynamic networks. One of these issues is precisely being able to process this kind of graphs. While the original LPA is already capable of finding communities in large networks, several variants have specifically tackled scalability problems, such as parallel versions [102–104]. In addition, versions for temporal networks handle dynamic graphs (see Section 6).

To further analyze the capability of LPA and its current variants to handle massive networks, we have illustrated in Fig. 4 the size and order of the largest networks processed by the works cited in the present review. For real-world networks, we take into account the number of nodes and edges, and for synthetic networks only the number of nodes, since the number of edges is variable. As we can see, LPA and its variants have been tested with networks of at most several hundred million nodes and several billion edges. These networks account for social media such as Facebook, the Web, and — in a lesser extent — information networks such as Wikipedia.

Another important issue that has caught attention lately concerns employing real-world networks along with an annotated community structure (also termed as *metadata*) as *ground truth* and attempting to compare the output of an algorithm against such metadata to assess the quality of the algorithm (for example, via normalized mutual information). It has been argued and analyzed that this approach may not be the most appropriate one for evaluation, since metadata does not always reflect network structure and, thus, a poor match between metadata groups and the communities output by an algorithm does not necessarily imply a poor performance by the algorithm [154]. It would be, therefore, important to use other evaluation criteria and not only metadata as ground truth to evaluate. In that sense, we have also analyzed to what extent is metadata used as a source of evaluation in our cited works. We found out that only approximately 13% of these works evaluate on the basis of metadata as ground truth and 2% solely use metadata as a source of evaluation (that is, do not use modularity, synthetic networks, or another metric to evaluate). Consequently, up to now, works that portray different LPA variants have complementary views of the obtained results.

The issue of evaluation with metadata as ground truth has been found to be specially relevant for real-world large networks; in particular, Hric et al. [155] report a low normalized mutual information (NMI) for this kind of networks. For this reason, we analyzed the quality reported for real-world networks in our repository of cited works. Because NMI is used only in some works to evaluate real networks (and basically only applied over the *Karate*, *Dolphins*, and *Football* networks),



(a) Real networks (b) Synthetic networks

Fig. 4. (a) shows the number of nodes (order) and edges (size) for the largest real-world networks processed by our cited works, whereas (b) shows the number of nodes for the largest synthetic networks processed.

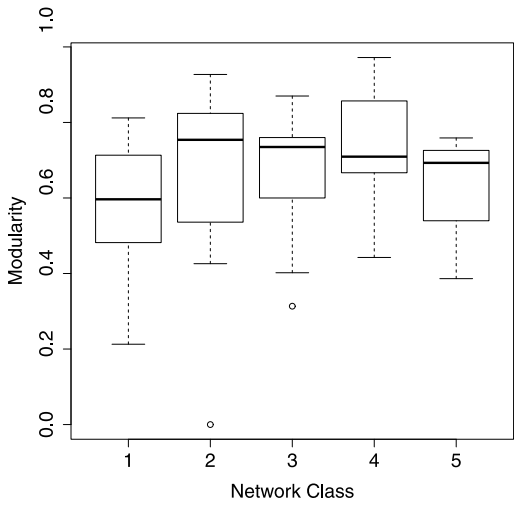


Fig. 5. Average reported modularity per network class. Class 1 consists of $n < 1{,}000$ (*Very Small*), Class 2 of $1{,}000 \leq n < 10{,}000$ (*Small*), Class 3 of $10{,}000 \leq n < 100{,}000$ (*Medium*), Class 4 of $100{,}000 \leq n < 1{,}000{,}000$ (*Large*) and Class 5 of $n > 1{,}000{,}000$ (*Very Large*).

we decided to instead take modularity as the analyzed quality metric — considering that only a very low percentage of our cited works do not employ it. Thus, we gathered the reported modularity for a sample of networks, which were divided into five classes according to node quantity (n), where *Class 1* consisted of $n < 1{,}000$, *Class 2* of $1{,}000 \leq n < 10{,}000$, *Class 3* of $10{,}000 \leq n < 100{,}000$, *Class 4* of $100{,}000 \leq n < 1{,}000{,}000$ and *Class 5* of $n > 1{,}000{,}000$; these classes could also be termed respectively as *Very small*, *Small*, *Medium*, *Large*, and *Very Large*. The aim was for each class to be represented by at least ten different networks — considering that most works use the same networks for evaluation. The only exception was *Class 5*, where the modularity reported could be gathered only for three different networks; this is because only a handful of works uses massive networks and sometimes these are not evaluated by means of modularity.

Our results for average reported modularity per network can be appreciated in Fig. 5. As we can see from this figure, *Very Small* networks (Class 1) have from moderate to slightly low reported modularity, in contrast to *Small* networks (Class 2), which tend to have high modularity; *Large* networks (Class 4) also tend to have high reported modularity. With *Very Large* networks, it is no simple to draw conclusions on the basis of only three networks; it can only be said that these networks had different reported modularity (in some cases low, in some cases high). In that sense, the lowest reported

modularity is being exhibited by *Very Small* networks and the highest modularity by *Large* and *Small* networks. As a consequence, it could be said that this result supports the findings by Hric et al. [155], since structural quality metrics do not necessarily agree with metadata-based metrics; it is therefore possible for large networks to have a high quality according to structure while having a low quality according to metadata.

Among our cited work, $\approx$ 53% of the approaches use a planted-partition model for evaluation with synthetic networks, and $\approx$ 87% of these cases use specifically the Lancichinetti–Fortunato–Radicchi (LFR) benchmark. The LFR benchmark is apt for the stochastic-blockmodeling facet and also related to the graph-clustering facet of Schaub et al. [34] (as discussed in Section 1).

9. Experimental results

In this section, we report computational experiments on combinations of various building blocks for LPA variants, where each building block is a feature option, such as initialization or node order. We first describe, per stage of LPA (see Algorithm 1) which options were implemented and how we understood each option — the way in which literature states algorithms is quite ambiguous more often than it should.

Initially, a *pre-processing* is carried out to eliminate artifacts that often complicate community detection: singletons and path subgraphs. The presence of outlier nodes such as isolated nodes or path-shaped subgraphs consisting communities reduces the internal density, whereas counting such outlier structures as communities of their own easily results in a high number low-quality communities. Hence, we first prune such structures: all isolated (zero-degree) nodes are removed, after which all single-neighbor nodes are removed, iteratively, until none remain. This, effectively, eliminates all trees[2].

Then, all combinations of the mix-and-match LPA implementation are executed on the remaining graph. The building blocks of our LPA implementation are the following:

Initialization The initial labels, with a unitary score, are assigned by

1. using the *unique* node identifier of each node as its label,
2. selecting a label uniformly at *random* among $k\lceil \log_2 n \rceil$ options with $k = 1$,
3. selecting a label uniformly at *random* among $k\lceil \log_2 n \rceil$ options with $k = 2$,
4. computing *rough cores*[3] [94].

Stopping condition Determining when to cease iteration, the following mechanisms are in place, with a fail-safe to detain after 10 iterations[4] if the condition is not met.

1. a *fixed* number of five iterations are executed, as literature reports that suffices for most graphs [48],
2. continue until the labels remain *stable* for s iterations in a row, or
3. continue until the absolute value of the change in modularity between iterations is below a *threshold* ρ.

Phasing The node labels are updated in one of two fashions:

1. the nodes are grouped into two disjoint sets, labeled by a breadth-first search alternating between phase labels "one" and "two" in each wave, and at each iteration, a *two-phase* process is carried out: the first set updates while the second set it *passive*, after which the second set updates while the first set is passive, or
2. all nodes update in a *single* phase.

Ordering The order in which the node labels are updated is one of the following:

1. at each iteration, the order *changes* as a new random ordering is applied,
2. a random ordering is created and that *same* order is used in all iterations,
3. the nodes are updated in *increasing* order of their degree, or
4. the nodes are updated in *decreasing* order of their degree.

Neighborhood The nodes that are taken into consideration for computing label frequencies for a specific node v are:

1. the neighbors $\Gamma(v)$ as well as the node v itself (*0–1 hop*), or
2. only the direct neighbors of the node in question (*1 hop*), or
3. neighbors of the neighbors, $\bigcup_{w\in\Gamma(v)} \Gamma(w)$ (*2 hop*, which includes 0 hop by definition).

2 A tree is an acyclic graph, that is, a graph where each node is connected to each other node by exactly one path.

3 We first assign each rough core a label of its own and then set unique labels for the nodes that are outside the cores.

4 A low limit is set due to the large number of combinations used in the experiments; in practice, time permitting, a larger number of iterations is recommended as a starting point.

Preference The contributions of the nodes are weighted in the frequency computations are either

1. *unitary* weights, giving all nodes the same preference,
2. the *clustering* coefficient of each node, or
3. the *eigenvector* centrality of each node.[5]

Frequency The label frequencies upon which the selection is made are computed either with

1. for each label, the score of the label is multiplied by the preference of the node that it holds, summing over all the nodes in consideration that hold that specific label (*absolute*), or
2. for each label, the highest score-preference product is used, subtracting the proportion of labels that changed in the previous iteration, unless the label is not the same as the current one; if a negative score were to result, it is considered zero — on the first iteration, nothing is subtracted (*dynamic hop attenuation* [71]).

Selection Resolving possible ties in the selection of the highest-frequency label as two options:

1. if *current* label is a candidate, it is chosen, or one is chosen uniformly at random,
2. the selection is *uniform* at random, regardless.

Mode The label update is performed either

1. *synchronously*, computing new labels using the present ones and then applying all changes at once, or
2. *asynchronously*, applying each label update the moment it is computed, affecting any subsequent updates within the neighborhood.

Community type The labeling is either

1. *flat* with a single label per node, with a unitary score, or
2. *overlapping* where a node can hold multiple labels with different scores; all maximal labels are retained with their resulting scores at each step.

The *post-processing* is always the same: for each label, the connected components of nodes holding that label is computed with a depth-first search, requiring a score of ξ or more in the overlapping scenario. Components with d or less nodes are *discarded* as too small — we use $\xi = 0.5$ and $d = 3$ in our experiments — and each of the larger ones is considered to be a separate community. The change threshold for modularity is set at $\rho = 0.05$.

This yields $4 \times 3 \times 2 \times 4 \times 3^2 \times 2^4 = 13{,}834$ variants of the LPA. In the face of this plethora of combinations, we leave parallelism as well as numerous other options for each block reported in the literature for future work for brevity, hoping to dive into this in further detail soon. We examine the resulting community structures in terms of

- the *number of communities* in the end result, especially whether there were none or if the algorithm produced a giant community consisting of the entire graph,
- the *modularity* of the resulting communities, filtering out the cases in which there were no communities,
- the *number of iterations* executed and whether or not that matches the maximum number permitted, and
- the total *run time* in milliseconds of the execution.

It is important to keep in mind that our threshold for significant communities was set at least four nodes, for which communities smaller than four do not appear in the results, and also that the maximum number of iterations permitted was set to ten in our experiments (double the amount the literature documents as adequate).

All graphs are generated using their similarly named NetworkX [156] routines, and the same library is employed for the calculations of clustering coefficients and eigenvalue centrality. The modularity computations are performed with `python-louvain` [157].

We execute each resulting LPA variant three times on each graph. Each execution is timed and the actual iterations that take place are counted. The experiments are implemented in Python (interpreter version 3.6.2) and executed on an iMac with macOS Mojave 10.14.5 with a 4 GHz processor and 32 GB main memory.

Our first experiment uses connected *caveman* graphs, varying the number of caves $\{3, 4, 5\}$ and the number of nodes in each individual cave in $\{6, 8, 10\}$ — an example is shown in Fig. 6 with five caves of ten nodes, with a flat labeling computed using a single phase, synchronous updates, initializing by rough cores, using the 1-hop neighborhood, updating the nodes in order of increasing degree, using the eigenvector centrality for preference, computing the absolute frequencies, and resolving ties uniformly at random; the community structure has a modularity of 0.78 and only two random LPA variants were executed in order to arrive at a community structure with five communities.

[5] We, compute this separately for each connected component for disconnected graphs.

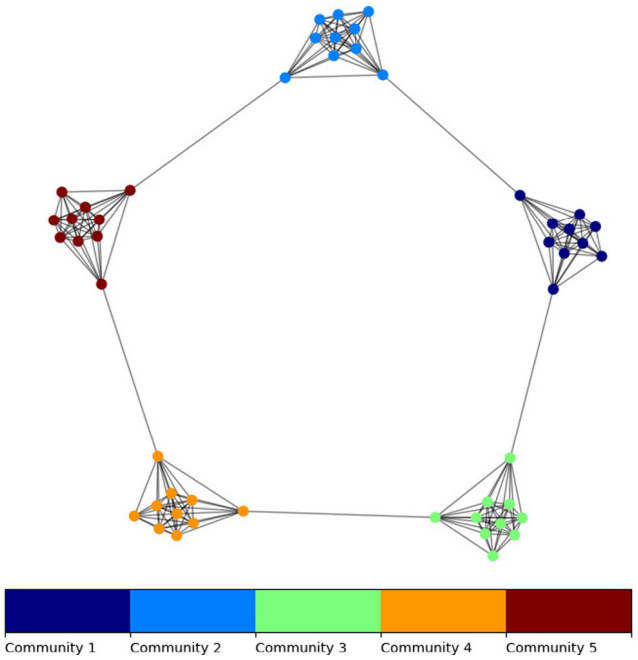


Fig. 6. An example caveman graph with communities computed with a random LPA variant.

We carry out $13{,}834 \times 3^3 = 373{,}248$ executions on caveman as three replicas are carried out on the execution of the algorithm for the nine combinations of cave count and the node count per cave. In this case, a correct community structure would naturally consist in the caves that are present in it. We therefore first examined how many communities did the LPA variants report and what modularity they had, shown in Figs. 7 and 8 respectively. In the former, it is easy to see that the community gets fragmented quite often. Also, both the case where no communities of sufficient order are found and the case where one giant community forms are of similar frequency with the desired number of communities. The latter figure, with the values of modularity, illustrates that low-quality community structures are as frequent as high-modularity ones.

For the second experiment, we first create two disjoint cliques, one with ten nodes and another with fifteen, and connect them by a single edge, like an imbalanced *barbell* graph with $n_b = 25$ nodes and $m_b = 151$ edges that has two very clear communities. Then we introduce "noise" by merging this with a randomly generated graph that has no natural communities, approximately controlling the order and size of the random graph to the extent that the parameters of the model grant us control over these two values. We then pre-process the resulting graph by pruning out the singletons and the paths.

ER Uniform random graphs G_{n_b,m_b} as proposed by Erdős and Rényi [158,159],

GR Geometric random graphs [160] in a unit square with $r = \sqrt{2m_b/\pi n_b^2}$ as the expected degree is approximately πr^2 and the average degree is $2m_b/n_b$,

WS Connected Watts–Strogatz (small-world) graphs [161] with $\lceil 2m_b/n_b \rceil$ edges per node and a one-percent rewiring probability, and

BA Scale-free Barabási–Albert graphs [162] with $\lceil 2m_b/n_b \rceil$ edges per node.

An example of such a graph, Fig. 9 shows one where the noise is a WS graph (essentially a cycle with some rewirings) and a third, degenerate community appears on the noisy part. Also the barbell communities have absorbed some of the noise. The community structure has a modularity of 0.11 and was computed in a single phase, asynchronously, using belonging coefficients for the labels, which are initially unique. Five iterations were computed, using the 1-hop neighborhood, a random ordering that is kept the same, preferences being the clustering coefficients, applying attenuation and using the current label to resolve ties — it took 21 attempts with random LPA variants to arrive at one with two or three communities.

The introduction of the four generation models increases the number of executions in such a manner that using replicas for graphs or executions of a single configuration becomes time-prohibitive, especially since with graphs of as few as a

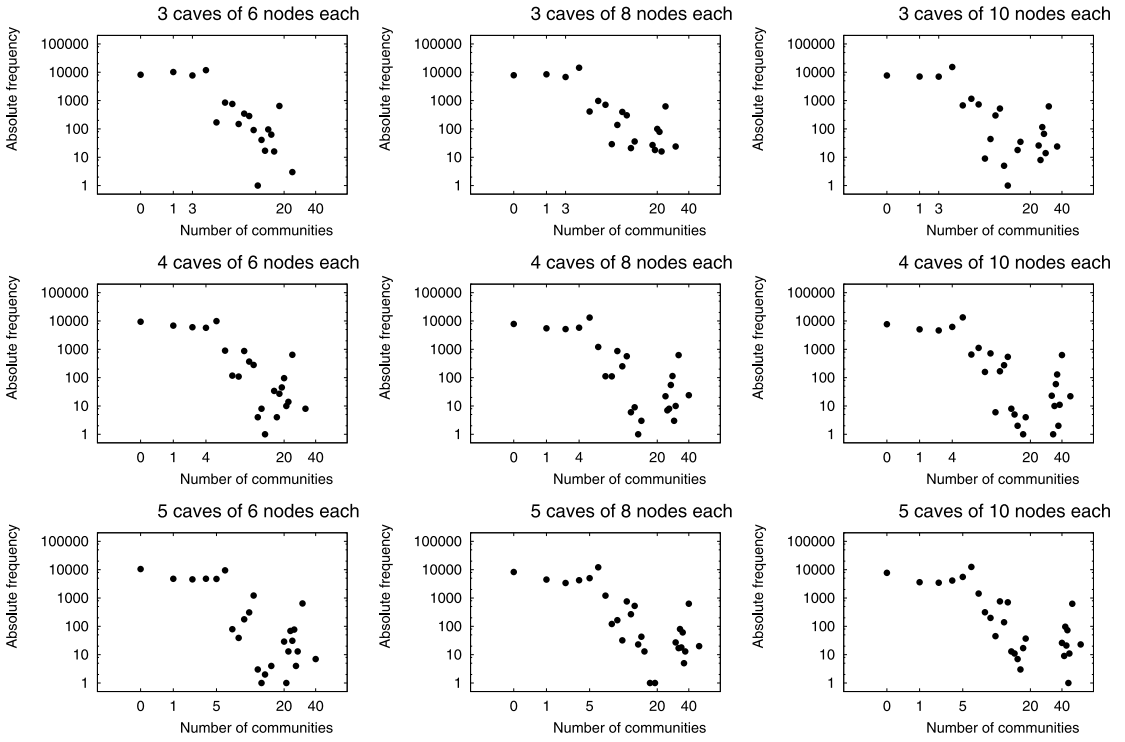


Fig. 7. Absolute frequencies in percentages for the number of communities in the caveman-graph executions.

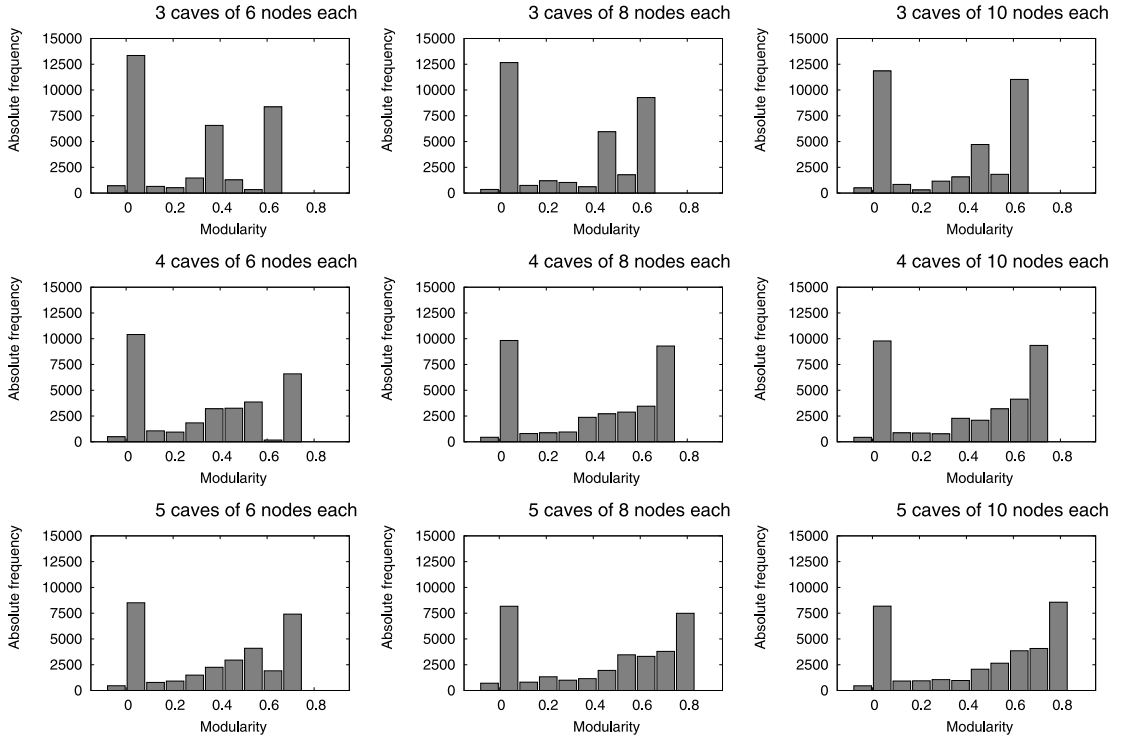


Fig. 8. Distributions of modularity in the caveman-graph executions.

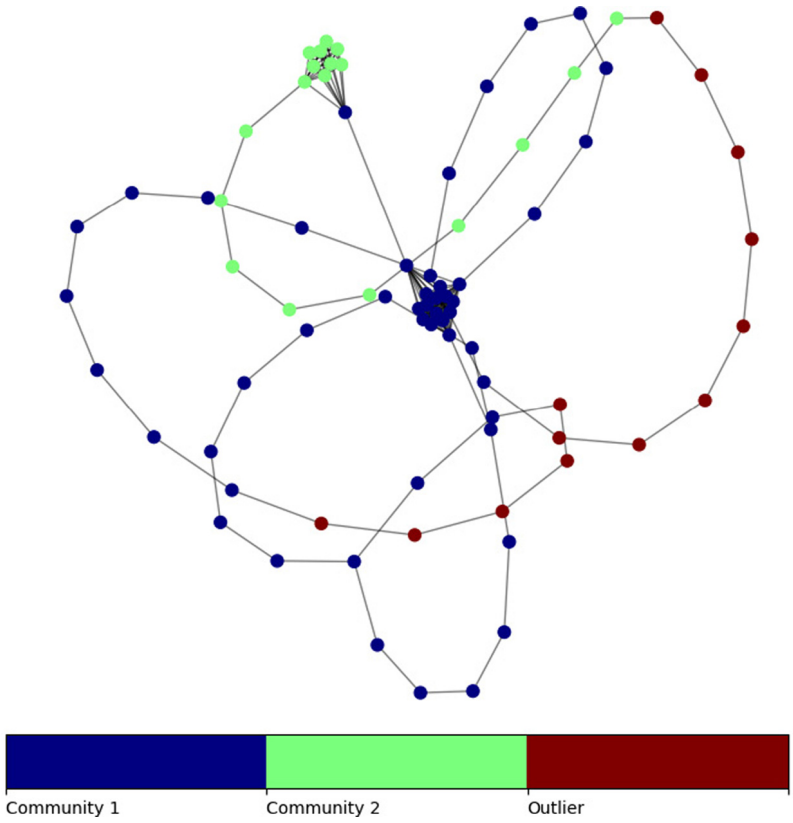


Fig. 9. An example barbell graph where the noise is a WS graph, with two communities identified by a random LPA variant and some nodes (appropriately) left as outliers.

hundred nodes begin requiring close to a minute to complete just ten iterations on some of the combinations. We hence set a time limit and interrupt any executions that require more than five seconds — for reference, the longest of the caveman-graph executions took 1244 ms. This yields a total of $13{,}834 \times 4 = 55{,}336$ executions, in which we set the generation parameters for the noise graph as $n_n = 100$ and $m_n = 150$ and record for each resulting input graph its *true* order $n < n_n$ and size m after the merging and the pruning is done, also the rounding up of the ceiling function puts the number of edges for the WS and the BA graphs above the value used to set the parameters. Also recall that we are combining the edges from the barbell to those of the noise (discarding duplicates), for which $m < m_b + m_n$.

The uniform graphs have a rather high change of pruning unless the graph is very dense, resulting in an average order of $\langle n_{\text{ER}} \rangle \approx 80$ and average size of $\langle m_{\text{ER}} \rangle \approx 281$. The geometric graphs undergo a similar level of pruning with an average order of $\langle n_{\text{GR}} \rangle \approx 78$ and average size of $\langle m_{\text{GR}} \rangle \approx 273$. The small-world graphs are somewhat less likely to prune, but the rewiring may well create a dangling path when the average degree is low (in this case, only two), resulting in an average order of $\langle n_{\text{WS}} \rangle \approx 93$ and average size of $\langle m_{\text{WS}} \rangle \approx 221$. The BA graphs are by construction connected, with degree of at least two for each node, for which $n_{\text{BA}} = 100$, and the average size was higher, $\langle m_{\text{BA}} \rangle \approx 414$.

As the random graphs may well be disconnected and hence the merged graph may also be disconnected, we compute the modularity per each component separately and then take a weighted average of those (the weight being the order of each component) to be the modularity over the graph as a whole, so as to avoid errors in the library used for the modularity computations.

For a more detailed analysis, we study those executions that were fast enough and resulted in a community structure that can be considered "promising" in terms of the number of communities. For the caveman graphs, these are the ones in which the number of resulting communities matches the number of caves, whereas those with no communities or one giant community are considered trivial, degenerate solutions and those with $\lfloor n/2 \rfloor$ or more communities are considered as fragmented, whereas those that are smaller than giant but are more numerous than the actual caves are considered coarse community structures and those that are more numerous than the caves but not many enough to consider scattered are considered fine-grained community structures. For the barbell graphs, we also consider zero or one communities to be degenerate, but as the noise from the random graph may, with low probability, introduce meaningful communities — we allow from two up to $\lfloor \log_2 n \rfloor$ communities to be considered as promising (for $n = 100$, this means six), more than two but below that threshold as fine-grained, and above that threshold as fragmented. The expected number of

Table 3
The frequency of each community-structure scenario for the two sets of experiments.

Scenario	Caveman graphs		Barbell graphs							
			ER		GR		WS		BA	
Attempted executions	373,248	(100%)	13,824	(100%)	13,824	(100%)	13,824	(100%)	13,824	(100%)
Time-limit exceeded	0	(0%)	78	(0.6%)	32	(0.2%)	0	(0%)	161	(1.2%)
Completed executions	373,248	(100%)	13,746	(99.4%)	13,792	(99.8%)	13,824	(100.0%)	13,663	(98.8%)
No communities	75,172	(20.1%)	2,568	(18.6%)	2,520	(18.2%)	2,558	(18.5%)	2,511	(18.2%)
Giant community	56,009	(15.9%)	5,799	(41.9%)	2,891	(20.9%)	4,295	(31.1%)	5,365	(38.8%)
Coarse communities	221,218	(59.3%)	*Not applicable*		*Not applicable*		*Not applicable*		*Not applicable*	
Promising communities	3,576	(1.0%)	1,594	(11.5%)	1,648	(11.9%)	2,037	(14.7%)	1,767	(12.8%)
Fine-grained communities	8,878	(2.4 %)	2,172	(15.7%)	4,034	(29.2%)	2,129	(15.4%)	2,359	(17.1%)
Fragmented communities	8,395	(2.2%)	1,613	(11.7%)	2,699	(19.5%)	2,805	(20.3%)	1,661	(12.0%)

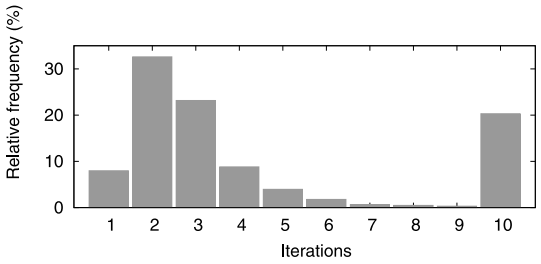


Fig. 10. The number of iterations under the modularity-threshold stopping condition on small caveman graphs.

significant communities is two for the barbell structure inserted into each graph, a community structure finer than the giant community but coarser than the expected structure cannot exist.

The frequency of each of these scenarios for both sets of experiments is given in Table 3. For the caveman graphs, the variants rarely arrive at the correct number of caves (the row corresponding to the promising communities is highlighted), as was already seen in Fig. 7. Success is slightly more frequent for the barbell graphs that also have a more flexible rule for when to consider a community structure promising, as we do not require strictly two communities in that case. Regardless, the grand majority of all the executions end up with a community structure where the number of communities is far from the desired value, giant communities are very common, as is the absence of communities.

Table 4 reports the frequency with which each building-block option appears among the executions so as to identify whether in the limited context of these small, synthetic graphs there are any choices that are better than others. The table reports also the run time in milliseconds, averaged over those executions, as well as the average graph order of the inputs used in that set of executions, in order to identify variations in computational load and if an option appears to perform better or worse as the graph order increases or decreases. The complete execution logs (`caveman.log` and `barbell.log`) are included in supplementary material, together with the Python code (`lpa.py`). It is important to note that the label-stability stopping condition was *never* fulfilled, but instead the fail-safe was triggered when the time limit was exceeded. Fig. 10 shows the distribution of the number of iterations executed under the modularity-threshold stopping conditions when the condition was in fact fulfilled within the maximum number of iterations allowed (which was set at ten). It is easy to see that the modularity either stabilizes early on within the first couple of iterations or not at all, with the threshold set at $\rho = 0.05$.

Keeping in mind that these experiments were conducted on rather small graphs with very dense known communities, with added noise in the second case, Table 4 shows that the only building block that was *never* the best choice was initialization with unique labels. All the other blocks found their place in the spotlight, either by incrementing the number of times the resulting community structure was promising or by reducing the computational effort. One "improves quality but takes time" types of a building block is the initialization by rough cores, whereas the 2-hop neighborhood is only costly without added benefit. Including the node itself in the neighborhood (0–1) appears beneficial in most cases.

The label-stability reaching stopping condition seems harsh, especially in the overlapping scenario where we consider any change in the belonging coefficient of each individual label also as a change in the labeling — further experiments with various coefficients that allow for efficient estimation of how drastic the label changes have been would be of interest, as well as combining modularity and stability considerations into a more complex stopping condition, as the modularity condition was able to reduce the number of iterations needed.

Synchronous computation seems to yield better results overall, although for the GR graphs there was no difference. As these graphs are not bipartite, using two phases or a single phase seems to have no clear effect. On the BA graphs, having more edges than the others, many variants struggled with the five-second time limit and seem to have diverged in the overlapping setup with 2-hop neighborhoods. The overlapping community type was only a benefit in the caveman graphs, where the endpoints of the edges "bridging" between caves are naturally inclined to belong to their own cave

Table 4
Results of the LPA variants on small graphs with clear communities. The first column # shows the number of acceptable community structures. The column $\langle n \rangle$ shows, only for the caveman graphs, the average order of the input graphs as it varies more. The column (ms) indicates the average wall-clock run time in millisecond of those executions. For the barbell graphs, the column ↗ indicates how many executions with this option were interrupted for exceeding the time limit (not applicable to the caveman graphs and hence omitted). The highest inclusion frequency and the lowest run time are highlighted for each of the five types of graphs; when several values are very close to being the best, both are highlighted. We also highlight the ones in which more than a hundred executions exceeded the time limit.

	Caveman graphs			Barbell graphs											
				ER			GR			WS			BA		
Option	#	$\langle n \rangle$	ms	#	ms	↗	#	ms	↗	#	ms	↗	#	ms	↗
Initialization															
unique	428	35.4	10.46	395	76.48	78	415	58.22	64	485	41.69	32	307	120.14	65
rand. $k = 1$	49	30.3	7.12	356	62.89	0	402	57.82	0	328	45.21	0	375	92.90	0
rand. $k = 2$	797	29.1	44.59	383	67.28	0	431	54.00	0	410	45.03	0	403	102.89	32
rough cores	2,248	33.4	45.12	460	68.15	32	400	57.88	0	814	35.53	0	682	90.23	64
Stopping condition															
fixed	1,256	33.2	29.32	523	55.77	14	654	53.14	0	618	33.20	0	617	83.86	64
stable	1,088	32.5	69.85	545	85.24	64	633	61.22	32	667	42.54	0	462	217.49	96
threshold	1,232	32.3	23.98	526	64.82	0	361	56.28	0	752	44.69	0	587	80.59	1
Phasing															
two-phase	1,959	33.0	37.09	701	65.14	40	807	54.65	16	988	42.25	0	821	98.14	81
single	1,617	32.2	43.11	893	71.73	38	841	59.13	16	1,049	38.79	0	946	99.52	80
Ordering															
changes	903	32.6	40.28	364	67.54	20	402	58.20	16	473	39.95	0	415	93.28	40
same	901	32.6	39.04	391	66.02	20	431	56.47	8	501	40.61	0	439	99.67	40
increasing	884	32.6	39.07	369	68.91	20	399	57.31	8	562	41.34	0	477	101.53	40
decreasing	886	32.9	40.87	470	72.11	18	416	55.83	8	501	39.83	0	436	100.53	41
Neighborhood															
0-1 hop	1,465	32.7	30.50	592	50.83	0	574	50.30	0	760	38.43	0	756	67.25	0
1 hop	911	32.5	27.19	573	48.66	0	563	52.36	0	876	41.58	0	694	58.58	0
2 hop	1,200	32.8	60.76	429	120.62	78	511	69.42	32	401	41.88	0	317	262.53	161
Preference															
unitary	3,365	32.6	41.50	542	29.62	78	435	23.70	32	606	18.06	0	491	47.13	161
clustering	105	34.0	10.14	520	69.09	0	493	49.95	0	717	34.63	0	491	105.24	0
eigenvector	106	35.2	15.63	532	108.53	0	720	81.80	0	714	65.34	0	510	170.48	0
Frequency															
absolute	495	35.1	10.33	484	27.42	0	1,360	50.04	0	1,303	37.84	0	1,236	57.65	0
attenuated	3,081	32.3	44.55	1,110	86.89	78	288	89.48	32	734	45.13	0	531	194.85	161
Selection															
current	1,784	32.8	39.91	793	69.85	71	941	56.30	16	1,124	38.50	0	872	104.54	81
uniform	1,792	32.6	39.72	801	67.82	71	707	57.77	32	913	42.88	0	895	93.37	80
Mode															
synchr.	2,042	33.1	34.88	952	67.77	32	824	57.79	16	1,217	40.63	0	937	100.507	81
asynchr.	1,534	32.1	46.38	642	70.41	46	824	56.06	16	829	40.22	0	830	97.54	80
Comm. type															
flat	21	36.2	9.50	1,5431	74.39	0	1,126	61.06	0	1,785	43.34	0	1,541	107.42	0
overlapping	3,555	32.7	39.99	163	20.03	78	522	48.04	32	252	20.08	0	226	40.64	161

and, also, in a lower degree to the neighboring cave — for all of the barbell graphs, requiring a flat community structure worked better.

The order in which the nodes are processed does not seem to be very relevant for these graphs, and neither does the tie-break rule for when there are several maximal labels. What comes to node preference, the structural measures of clustering coefficient and eigenvector centrality are rather useless in the caveman graphs that have a largely symmetric structure, with all nodes playing a similar role in the structure. For the noisy barbell graphs, there is either little difference between using the clustering coefficients or the eigenvector centralities, or the latter are a bit better.

In conclusion, most of the options appear to be beneficial in at least some cases, although there is clearly room for improvement so as to avoid the traps of the giant community and the opposite error of the scattering the nodes into small fragmented communities.

10. Open problems and future directions

Even though LPA has been widely enhanced and extended, there is still work to be done. A comprehensive analysis of the impact and behavior of proposed enhancements and their combinations has not yet been published. Instead of the

present state of fragmented enhancements, an updated version of LPA combining several proposed improvements would be a welcome addition to the state of the art, along with a formal analysis of such an integrated variant.

Formal analysis on LPA as such is scarce; the issues with convergence and other major problems have been discovered empirically. It would useful to have formal proofs for such behaviors to correctly evaluate the strengths and weaknesses of the algorithm and its variants. Some examples of possible formal analysis include stating the relationship (or equivalence) of LPA with other community-detection algorithms, establishing whether LPA suffers from the resolution limit, formalizing the trade-off between time and quality, and determining under what conditions is convergence guaranteed. Similarly, some of the desirable properties of LPA have only been stated empirically — such as having 95% of the nodes clustered after five iterations. A formal study of the circumstances under which this holds and how the proportion of converged nodes behaves as a function of the number of iterations would allow for a better foundation to further analysis.

Also, hierarchical and fuzzy variants deserve further attention. Hierarchical communities are relevant in many application domains where communities are naturally present (topic mining in hyperlinked information repositories such as Wikipedia, for example). The leap from overlapping variants to a fuzzy formulation is still to be made — to the best of our knowledge, no strictly fuzzy LPA variant has been published. This may be partially due to the need for more evaluation metrics and benchmarks for fuzzy community detection. Variants for k-partite graphs as well as and hyper-graphs are also desirable.

LPA has low computational cost and can hence be used to boost other community-detection algorithms to produce powerful hybrid approaches. For instance, LPA could be used as part of a hyper-heuristic or as a constructive phase followed by local search of improvements.

The parallel versions could be extended into distributed algorithms where the computations take place in several, possibly heterogeneous and unreliable machines. Also self-stabilizing algorithms that do not explicitly terminate but instead adjust to changes in the input graph over time are of interest to perform community-detection in dynamic graphs.

11. Conclusions

In this survey, the main enhancements, extensions, and uses of LPA have been discussed. LPA has shown to be a scalable and simple algorithm: it is local, requires no parameters, and is easy to understand and implement. However, several major issues have been identified, including sensitivity to node ordering and randomness in the update rule (resulting in instability), the formation of a trivial solution known as the monster community, the lack or slowness of convergence in several cases, and the limitations of the one-hop horizon. We described several variants that have been proposed to overcome such issues, mainly via modifying the update rule and attempting to eliminate randomness in the algorithm. The original algorithm has also been extended for different kinds of networks (such as weighted or bipartite) and modified to allow for different community detection schemes (overlapping, for example). Other uses of the LPA include partitioning and structure compression of massive networks. Hybrid algorithms based on LPA include evolutionary approaches that combine LPA in their structure to achieve better solutions with little overhead.

One of the main challenges for LPA variants (both for enhancements and extensions) is to improve the algorithm without affecting its scalability and simpleness. For the time being, the simplest version is the original formulation, and it is very efficient in a considerable number of ways. The variants proposed, however, overcome some known issue with the results or the behavior of the algorithm thus surpassing the original version. The variants, nevertheless, come with a price — which could be affordable or not. We expect that future variants will seek to identify the smallest possible price for the proposed changes that yields the greatest gain in terms of reducing the problematic aspects or avoiding them altogether, ideally with a formal proof of how the cost and the benefit are related.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.physa.2019.122058.

References

[1] Thiago H. Silva, Aline Carneiro Viana, Fabrício Benevenuto, Leandro Villas, Juliana Salles, Antonio Loureiro, Daniele Quercia, Urban computing leveraging location-based social network data: A survey, ACM Comput. Surv. (ISSN: 0360-0300) 52 (1) (2019) 17:1–17:39, http://dx.doi.org/10.1145/3301284, URL http://doi.acm.org/101145/3301284.

[2] Jooho Kim, Makarand Hastak, Social network analysis: Characteristics of online social networks after a disaster, Int. J. Inf. Manage. (ISSN: 0268-4012) 38 (1) (2018) 86–96, http://dx.doi.org/10.1016/j.ijinfomgt.2017.08.003, URL http://www.sciencedirect.com/science/article/pii/S026840121730525X.

[3] Stanley Wasserman, Katherine Faust, Social Network Analysis: Methods and Applications, Vol. 8, Cambridge University Press, Cambdridge, UK, 1994.

[4] Mark E.J. Newman, Networks: An Introduction, Oxford University Press, New York, NY, 2010.

[5] L. Safak Yilmaz, Albertha J.M. Walhout, Walhout metabolic network modeling with model organisms, Curr. Opin. Chem. Biol. (ISSN: 1367-5931) 36 (2017) 32–39, http://dx.doi.org/10.1016/j.cbpa.2016.12.025, URL http://www.sciencedirect.com/science/article/pii/S1367593116302095.

[6] Jeong-Mo Choi, Amy I. Gilson, Eugene I. Shakhnovich, Graph's topology and free energy of a spin model on the graph, Phys. Rev. Lett. 118 (2017) 088302, http://dx.doi.org/10.1103/PhysRevLett.118.088302, URL https://link.aps.org/doi/101103/PhysRevLett.118088302.

[7] Emmanuel Martinez-Ledesma, Roeland G.W. Verhaak, Victor Treviño, Identification of a multi-cancer gene expression biomarker for cancer clinical outcomes using a network-based algorithm, Sci. Rep. 5 (2015) 11966.

[8] B. Liu, Z. Li, X. Chen, Y. Huang, X. Liu, Recognition and vulnerability analysis of key nodes in power grid based on complex network centrality, IEEE Trans. Circuits Syst. II (ISSN: 1549-7747) 65 (3) (2018) 346–350, http://dx.doi.org/10.1109/TCSII.2017.2705482.

[9] Giuliano Andrea Pagani, Marco Aiello, The power gridas a complex network: A survey, Physica A (ISSN: 0378-4371) 392 (11) (2013) 2688–2700, http://dx.doi.org/10.1016/j.physa.2013.01.023, URL http://www.sciencedirect.com/science/article/pii/S0378437113000575.

[10] Diego F. Rueda, Eusebi Calle, Jose L. Marzo, Robustness comparison of 15 real telecommunication networks: Structural and centrality measurements, J. Netw. Syst. Manage. (ISSN: 1573-7705) 25 (2) (2017) 269–289, http://dx.doi.org/10.1007/s10922-016-9391-y.

[11] Tao Zhou, Jie Ren, Matúš Medo, Yi-Cheng Zhang, Bipartite network projection and personal recommendation, Phys. Rev. E 76 (2007) 046115, http://dx.doi.org/10.1103/PhysRevE.76.046115, URL https://link.aps.org/doi/101103/PhysRevE.76046115.

[12] Jianming He, Wesley W. Chu, A social network-based recommender system (snrs), in: Nasrullah Memon, Jennifer Jie Xu, David L. Hicks, Hsinchun Chen (Eds.), Data Mining for Social Network Data, Springer, US, Boston, MA, ISBN: 978-1-4419-6287-4, 2010, pp. 47–74, http://dx.doi.org/10.1007/978-1-4419-6287-4_4.

[13] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey, Knowl.-Based Syst. (ISSN: 0950-7051) 46 (2013) 109–132, http://dx.doi.org/10.1016/j.knosys.2013.03.012, URL http://www.sciencedirect.com/science/article/pii/S0950705113001044.

[14] Zhoubao Sun, Lixin Han, Wenliang Huang, Xueting Wang, Xiaoqin Zeng, Min Wang, Hong Yan, Recommender systems based on social networks, J. Syst. Softw. (ISSN: 0164-1212) 99 (2015) 109–119, http://dx.doi.org/10.1016/j.jss.2014.09.019, URL http://www.sciencedirect.com/science/article/pii/S0164121214002064.

[15] Pablo Chamoso, Alberto Rivas, Sara Rodrguez, Javier Bajo, Relationship recommender system in a business and employment-oriented social network, Inform. Sci. (ISSN: 0020-0255) 433–434 (2018) 204–220, http://dx.doi.org/10.1016/j.ins.2017.12.050, URL http://www.sciencedirect.com/science/article/pii/S0020025517311660.

[16] Julian Mcauley, Jure Jeskovec, Discovering social circles in ego networks, ACM Trans. Knowl. Discov. Data (ISSN: 1556-4681) 8 (1) (2014) 4:1–4:28, http://dx.doi.org/10.1145/2556612, URL http://doi.acm.org/101145/2556612.

[17] Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, Sunita Verma, Ego-net community mining applied to friend suggestion, Proc. VLDB Endow. (ISSN: 2150-8097) 9 (4) (2015) 324–335, http://dx.doi.org/10.14778/2856318.2856327.

[18] Ullas Gargi, Wenjun Lu, Vahab Mirrokni, Sangho Yoon, Large-scale community detection on youtube for topic discovery and exploration, in: Fifth International AAAI Conference on Weblogs and Social Media, 2011.

[19] S. Huang, H. Chen, Exploring the online underground marketplaces through topic-based social network and clustering, in: 2016 IEEE Conference on Intelligence and Security Informatics (ISI), 2016, pp. 145–150, http://dx.doi.org/10.1109/ISI.2016.7745458.

[20] Hui Li, Dingming Wu, Wenbin Tang, Nikos Mamoulis, Overlapping community regularization for rating prediction in social recommender systems, in: Proceedings of the 9th ACM Conference on Recommender Systems, RecSys '15, ACM, New York, NY, USA, ISBN: 978-1-4503-3692-5, 2015, pp. 27–34, http://dx.doi.org/10.1145/2792838.2800171, URL http://doi.acm.org/101145/27928382800171.

[21] Jianxing Zheng, Yanjie Wang, Personalized recommendations based on sentimental interest community detection, Sci. Program. 2018 (2018).

[22] Jianxing Zheng, Suge Wang, Deyu Li, Bofeng Zhang, Personalized recommendation based on hierarchical interest overlapping community, Inform. Sci. (ISSN: 0020-0255) 479 (2019) 55–75, http://dx.doi.org/10.1016/j.ins.2018.11.054, URL http://www.sciencedirect.com/science/article/pii/S002002551830940X.

[23] Lilian Weng, Filippo Menczer, Yong-yeol Ahn, Virality prediction and community structure in social networks, Sci. Rep. (Nat. Publisher Group) 3 (2013) 2522, URL https://search.proquest.com/docview/1897440217?accountid=38018.

[24] M. Randles, D. Lamb, A. Taleb-Bendiab, Experiments with honeybee foraging inspired load balancing, in: 2009 Second International Conference on Developments in eSystems Engineering, 2009, pp. 240–247, http://dx.doi.org/10.1109/DeSE.2009.19.

[25] Maria Grineva, Maxim Grinev, Dmitry Lizorkin, Extracting key terms from noisy and multitheme documents, in: Proceedings of the 18th International Conference on World Wide Web, WWW '09, ACM, New York, NY, USA, ISBN: 978-1-60558-487-4, 2009, pp. 661–670, http://dx.doi.org/10.1145/1526709.1526798, URL http://doi.acm.org/101145/1526709.1526798.

[26] Tomasz Hachaj, Marek R. Ogiela, Clustering of trending topics in microblogging posts: A graph-based approach, Future Gener. Comput. Syst. (ISSN: 0167-739X) 67 (2017) 297–304, http://dx.doi.org/10.1016/j.future.2016.04.009, URL http://www.sciencedirect.com/science/article/pii/S0167739X16300863.

[27] Youcef Abdelsadek, Kamel Chelghoum, Francine Herrmann, Imed Kacem, Benot Otjacques, Community extraction and visualization in social networks applied to twitter, Inform. Sci. (ISSN: 0020-0255) 424 (2018) 204–223, http://dx.doi.org/10.1016/j.ins.2017.09.022, URL http://www.sciencedirect.com/science/article/pii/S0020025516307800.

[28] Ludvig Bohlin, Daniel Edler, Andrea Lancichinetti, Martin Rosvall, Community detection and visualization of networks with the map equation framework, in: Ying Ding, Ronald Rousseau, Dietmar Wolfram (Eds.), Measuring Scholarly Impact: Methods and Practice, Springer International Publishing, Cham, ISBN: 978-3-319-10377-8, 2014, pp. 3–34, http://dx.doi.org/10.1007/978-3-319-10377-8_1.

[29] Santo Fortunato, Community detection in graphs, Phys. Rep. 486 (3–5) (2010) 75–174, http://dx.doi.org/10.1016/j.physrep.2009.11.002.

[30] Satu Elisa Schaeffer, Graph clustering, Comput. Sci. Rev. 1 (1) (2007) 27–64, http://dx.doi.org/10.1016/j.cosrev.2007.05.001.

[31] Jaewon Yang, Jure Leskovec, Defining and evaluating network communities based on ground-truth, Knowl. Inf. Syst. 42 (1) (2015) 181–213, http://dx.doi.org/10.1007/s10115-013-0693-z.

[32] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, Ploutarchos Spyridonos, Community detection in social media, Data Min. Knowl. Discov. 24 (3) (2012) 515–554, http://dx.doi.org/10.1007/s10618-011-0224-z.

[33] Gary William Flake, Steve Lawrence, C. Lee Giles, Frans M. Coetzee, Self-organization and identification of web communities, Computer 35 (3) (2002) 66–70.

[34] Michael T. Schaub, Jean-Charles Delvenne, Martin Rosvall, Renaud Lambiotte, The many facets of community detection in complex networks, Appl. Netw. Sci. (ISSN: 2364-8228) 2 (1) (2017) 4, http://dx.doi.org/10.1007/s41109-017-0023-6.

[35] Michelle Girvan, Mark E.J. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. 99 (12) (2002) 7821–7826, http://dx.doi.org/10.1073/pnas.122653799.

[36] Andrew Y. Ng, Michael I. Jordan, Yair Weiss, On spectral clustering: Analysis and an algorithm, in: T.G. Dietterich, S. Becker, Z. Ghahramani (Eds.), Proceedings of the Fourteenth Conference on Advances in Neural Information Processing Systems, Vol. 2, The MIT Press, Cambridge, MA, USA, 2002.

[37] Aaron Clauset, Mark E.J. Newman, Cristopher Moore, Finding community structure in very large networks, Phys. Rev. E 70 (2004) 066111, http://dx.doi.org/10.1103/PhysRevE.70.066111.

[38] Mark E.J. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. 103 (23) (2006) 8577–8582, http://dx.doi.org/10.1073/pnas.0601602103.

[39] Santo Fortunato, Marc Barthélemy, Resolution limit in community detection, Proc. Natl. Acad. Sci. 104 (1) (2007) 36–41, http://dx.doi.org/10.1073/pnas.0605965104.

[40] Zhenping Li, Shihua Zhang, Rui-Sheng Wang, Xiang-Sun Zhang, Luonan Chen, Quantitative function for community detection, Phys. Rev. E 77 (3) (2008) 036109, http://dx.doi.org/10.1103/PhysRevE.77.036109.

[41] Gergely Palla, Imre Derényi, Illés Farkas, Tamás Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (2005) 814–818, http://dx.doi.org/10.1038/nature03607.

[42] Andrea Lancichinetti, Santo Fortunato, János Kertész, Detecting the overlapping and hierarchical community structure in complex networks, New J. Phys. 11 (3) (2009) 033015, http://dx.doi.org/10.1088/1367-2630/11/3/033015.
[43] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.
[44] Jierui Xie, Stephen Kelley, Boleslaw K. Szymanski, Overlapping community detection in networks: The state-of-the-art and comparative study, ACM Comput. Surv. 45 (4) (2013) 43:143:35, http://dx.doi.org/10.1145/2501654.2501657.
[45] Santo Fortunato, Darko Hric, Community detection in networks: A user guide, Phys. Rep. 659 (Supplement C) (2016) 1–44, http://dx.doi.org/10.1016/j.physrep.2016.09.002.
[46] Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, Santo Fortunato, Finding statistically significant communities in networks, PLoS One 6 (4) (2011) e18961, http://dx.doi.org/10.1371/journal.pone.0018961.
[47] Andrea Lancichinetti, Santo Fortunato, Consensus clustering in complex networks, Sci. Rep. 2 (2012) 336, http://dx.doi.org/10.1038/srep00336.
[48] Usha Nandini Raghavan, Réka Albert, Soundar Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (2007) 036106, http://dx.doi.org/10.1103/PhysRevE.76.036106.
[49] Johan Ugander, Lars Backstrom, Balanced label propagation for partitioning massive graphs, in: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, ACM, New York, NY, USA, 2013, pp. 507–516, http://dx.doi.org/10.1145/2433396.2433461.
[50] Lu Wang, Yanghua Xiao, Bin Shao, Haixun Wang, How to partition a billion-node graph, in: Data Engineering, IEEE, 2014, pp. 568–579, http://dx.doi.org/10.1109/ICDE.2014.6816682.
[51] Salvatore Catanese, Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, Alessandro Provetti, Extraction and analysis of facebook friendship relations, in: Ajith Abraham (Ed.), Computational Social Networks: Mining and Visualization, Springer, London, UK, 2012, pp. 291–324, http://dx.doi.org/10.1007/978-1-4471-4054-2_12.
[52] Alejandro Baldominos, Javier Calle, Dolores Cuadra, Beyond social graphs: mining patterns underlying social interactions, Pattern Anal. Appl. 20 (2017) 269–285, http://dx.doi.org/10.1007/s10044-016-0550-2.
[53] Hyoungshick Kim, John Tang, Ross Anderson, Social authentication: Harder than it looks, in: Financial Cryptography and Data Security, in: Lecture Notes in Computer Science, vol. 7397, Springer, Berlin / Heidelberg, Germany, 2012, p. 15, http://dx.doi.org/10.1007/978-3-642-32946-3_1.
[54] Sounman Hong, Sun Hyoung Kim, Political polarization on twitter: Implications for the use of social media in digital governments, Gov. Inf. Q. 33 (2016) http://dx.doi.org/10.1016/j.giq.2016.04.007.
[55] Michael D. Conover, Bruno Goncalves, Jacob Ratkiewicz, Alessandro Flammini, Filippo Menczer, Predicting the political alignment of twitter users, in: IEEE Third International Conference on Privacy, Security, Risk and Trust and IEEE Third International Conference on Social Computing, IEEE, 2011, http://dx.doi.org/10.1109/PASSAT/SocialCom.2011.34.
[56] Linyuan Lü, Duanbing Chen, Xiao-Long Ren, Qian-Ming Zhang, Yi-Cheng Zhang, Tao Zho, Vital nodes identification in complex networks, Phys. Rep. 650 (2016) 63, http://dx.doi.org/10.1016/j.physrep.2016.06.007.
[57] Jian-Guo Liu, Jian-Hong Lin, Qiang Guo, Tao Zhou, Locating influential nodes via dynamics-sensitive centrality, Sci. Rep. 6 (2016) 21380, http://dx.doi.org/10.1038/srep21380.
[58] Duan-Bing Chen, Hui Gao, Linyuan Lü, Tao Zhou, Identifying influential nodes in large-scale directed networks: The role of clustering, PLoS One 8 (e77455) (2013) http://dx.doi.org/10.1371/journal.pone.0077455.
[59] Danielle S. Bassett, Nicholas F. Wymbs, Mason A. Porter, Peter J. Mucha, Jean M. Carlson, Scott T. Grafton, Dynamic reconfiguration of human brain networks during learning, Proc. Natl. Acad. Sci. USA 108 (18) (2011) 7641–7646, http://dx.doi.org/10.1073/pnas.1018985108.
[60] Alex Fornito, Andrew Zalesky, Edward Bullmore, Fundamentals of Brain Network Analysis, Academic Press, London, UK, ISBN: 978-0-12-407908-3, 2016.
[61] Wei Lui, Aiping Wu, Uncover protein complexes in E. coli network, in: IEEE International Conference on Bioinformatics and Biomedicine, IEEE, 2015, http://dx.doi.org/10.1109/BIBM.2015.7359844.
[62] Manuel Guerrero, Consolación Gil, Francisco G. Montoya, Alfredo Alcayde, Raúl Baños, Community detection in power grids by an evolutionary method, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, 2017, pp. 279–289, http://dx.doi.org/10.1145/3067695.3075606.
[63] Yang Ou, Yiming Wang, Efficient methods for spectrum sensing in cognitive radio, in: International Conference on Wireless Communications and Signal Processing, IEEE, 2010, http://dx.doi.org/10.1109/WCSP.2010.5630652.
[64] Tao Wu, Leiting Chen, Linfeng Zhong, Xingping Xian, Enhanced collective influence: A paradigm to optimize network disruption, Physica A 472 (2017) 43–52, http://dx.doi.org/10.1016/j.physa.2016.12.036.
[65] Meng-Cen Qian, Zhi-Qiang Jiang, Wei-Xing Zhou, Universal and nonuniversal allometric scaling behaviors in the visibility graphs of world stock market indices, J. Phys. A 43 (33) (2010) http://dx.doi.org/10.1088/1751-8113/43/33/335002.
[66] Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632, http://dx.doi.org/10.1145/324133.324140.
[67] Ian X.Y. Leung, Pan Hui, Pietro Lió, Jon Crowcroft, Towards real-time community detection in large networks, Phys. Rev. E 79 (6) (2009) 066107, http://dx.doi.org/10.1103/PhysRevE.79.066107.
[68] Steve Gregory, Finding overlapping communities in networks by label propagation, New J. Phys. 12 (10) (2010) 103018, http://dx.doi.org/10.1088/1367-2630/12/10/103018.
[69] Michael J. Barber, John W. Clark, Detecting network communities by propagating labels under constraints, Phys. Rev. E 80 (2009) 026129, http://dx.doi.org/10.1103/PhysRevE.80.026129.
[70] Xin Liu, Tsuyoshi Murata, Advanced modularity-specialized label propagation algorithm for detecting communities in networks, Physica A 389 (7) (2010) 1493–1500, http://dx.doi.org/10.1016/j.physa.2009.12.019.
[71] Lovro Šubelj, Marko Bajec, Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction, Phys. Rev. E 83 (2011) 036103, http://dx.doi.org/10.1103/PhysRevE.83.036103.
[72] Jierui Xie, Boleslaw K. Szymanski, Xiaoming Liu, SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process, in: Data Mining Workshops, IEEE Computer Society, Washington, DC, USA, 2011, pp. 344–349, http://dx.doi.org/10.1109/ICDMW.2011.154.
[73] Gergely Tibély, János Kertész, On the equivalence of the label propagation method of community detection and a Potts model approach, Physica A 387 (19) (2008) 4982–4984, http://dx.doi.org/10.1016/j.physa.2008.04.024.
[74] Jierui Xie, Boleslaw K. Szymanski, LabelRank: A stabilized label propagation algorithm for community detection in networks, in: Network Science Workshop, IEEE, 2013, pp. 138–143.
[75] Lovro Šubelj, Marko Bajec, Robust network community detection using balanced propagation, Eur. Phys. J. B 81 (3) (2011) 353–362, http://dx.doi.org/10.1140/epjb/e2011-10979-2.
[76] Hao Lou, Shenghong Li, Yuxin Zhao, Detecting community structure using label propagation with weighted coherent neighborhood propinquity, Physica A 392 (14) (2013) 3095–3105, http://dx.doi.org/10.1016/j.physa.2013.03.014.
[77] Lovro Šubelj, Label propagation for clustering, in: P. Doreian, V. Batagelj, A. Ferligoj (Eds.), Advances in Network Clustering and Blockmodeling, Wiley, New York, 2018.

[78] Ronghua Shang, Weitong Zhang, Licheng Jiao, Circularly searching core nodes based label propagation algorithm for community detection, Int. J. Pattern Recognit. Artif. Intell. 30 (08) (2016) 1659024, http://dx.doi.org/10.1142/S0218001416590242.

[79] Gerard Salton, Michael J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., New York, NY, 1983.

[80] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, Thomas A.J. Schweiger, SCAN: A structural clustering algorithm for networks, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2007, pp. 824–833, http://dx.doi.org/10.1145/1281192.1281280.

[81] Leonid Barenboim, Michael Elkin, Distributed (δ+1)-coloring in linear (in δ) time, in: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 2009, pp. 111–120, http://dx.doi.org/10.1145/1536414.1536432.

[82] Mark E.J. Newman, Michelle Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113, http://dx.doi.org/10.1103/PhysRevE.69.026113.

[83] Zengqiang Chen, Zheng Xie, Qing Zhang, Community detection based on local topological information and its application in power grid, Neurocomputing 170 (2015) 384–392, http://dx.doi.org/10.1016/j.neucom.2015.04.093.

[84] Junhao Zhang, Tongfei Chen, Junfeng Hu, On the relationship between Gaussian stochastic blockmodels and label propagation algorithms, J. Stat. Mech. Theory Exp. 2015 (3) (2015) P03009, http://dx.doi.org/10.1088/1742-5468/2015/03/P03009.

[85] Gennaro Cordasco, Luisa Gargano, Label propagation algorithm: a semi-synchronous approach, Int. J. Soc. Netw. Min. 1 (1) (2012) 3–26, http://dx.doi.org/10.1504/IJSNM.2012.045103.

[86] Aiping Zhang, Guang Ren, Yejin Lin, Baozhu Jia, Hui Cao, Jundong Zhang, Shubin Zhang, Detecting community structures in networks by label propagation with prediction of percolation transition, Sci. World J. 2014 (2014) 148686, http://dx.doi.org/10.1155/2014/148686.

[87] Jia Hou Chin, Kuru Ratnavelu, A semi-synchronous label propagation algorithm with constraints for community detection in complex networks, Sci. Rep. 7 (2017) 45836.

[88] Xin Liu, Tsuyoshi Murata, How does label propagation algorithm work in bipartite networks? in: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 03, IEEE Computer Society, 2009, pp. 5–8, http://dx.doi.org/10.1109/WI-IAT.2009.217.

[89] Jirí Šíma, Satu Elisa Schaeffer, On the NP-completeness of some graph cluster measures, in: Proceedings of the Thirty-Second International Conference on Current Trends in Theory and Practice of Computer Science, in: Lecture Notes in Computer Science, vol. 3831, Springer Verlag, Berlin/Heidelberg, Germany, 2006, pp. 350–357, http://dx.doi.org/10.1007/11611257_51.

[90] Shichao Liu, Fuxi Zhu, Huajun Liu, Zhiqiang Du, A core leader based label propagation algorithm for community detection, China Commun. 13 (12) (2016) 97–106, http://dx.doi.org/10.1109/CC.2016.7897535.

[91] Wei Liu, Xingpeng Jiang, Matteo Pellegrini, Xiaofan Wang, Discovering communities in complex networks by edge label propagation, Sci. Rep. 6 (2016) 22470, http://dx.doi.org/10.1038/srep22470.

[92] Wei Weng, A. Nian Zhang, Huarong Xu, Community mining method of label propagation based on dense pairs, J. Eng. Sci. Technol. Rev. 7 (3) (2014) 68–73.

[93] Jianbin Huang, Heli Sun, Jiawei Han, Boqin Feng, Density-based shrinkage for revealing hierarchical and overlapping community structure in networks, Physica A 390 (11) (2011) 2160–2171, http://dx.doi.org/10.1016/j.physa.2010.10.040.

[94] Zhi-Hao Wu, You-Fang Lin, Steve Gregory, Huai-Yu Wan, Sheng-Feng Tian, Balanced multi-label propagation for overlapping community detection in social networks, J. Comput. Sci. Tech. 27 (3) (2012) 468–479, http://dx.doi.org/10.1007/s11390-012-1236-x.

[95] Weitong Zhang, Rui Zhang, Ronghua Shang, Licheng Jiao, Weighted compactness function based label propagation algorithm for community detection, Physica A (ISSN: 0378-4371) 492 (2018) 767–780, http://dx.doi.org/10.1016/j.physa.2017.11.006, URL http://www.sciencedirect.com/science/article/pii/S0378437117310786.

[96] Zheng-Hong Deng, Hong-Hai Qiao, Qun Song, Li Gao, A complex network community detection algorithm based on label propagation and fuzzy c-means, Physica A (ISSN: 0378-4371) 519 (2019) 217–226, http://dx.doi.org/10.1016/j.physa.2018.12.024, URL http://www.sciencedirect.com/science/article/pii/S037843711831536X.

[97] Zhen Lin, Xiaolin Zheng, Nan Xin, Deren Chen, CK-LPA: Efficient community detection algorithm based on label propagation with community kernel, Physica A 416 (2014) 386–399, http://dx.doi.org/10.1016/j.physa.2014.09.023.

[98] Qiong Gui, Rui Deng, Pengfei Xue, Xiaohui Cheng, A community discovery algorithm based on boundary nodes and label propagation, Pattern Recognit. Lett. (ISSN: 0167-8655) 109 (2018) 103–109, http://dx.doi.org/10.1016/j.patrec.2017.12.018, URL http://www.sciencedirect.com/science/article/pii/S0167865517304646 Special Issue on Pattern Discovery from Multi-Source Data (PDMSD).

[99] Kuang Zhou, Arnaud Martin, Quan Pan, Zhunga Liu, Selp: Semi-supervised evidential label propagation algorithm for graph data clustering, Internat. J. Approx. Reason. (ISSN: 0888-613X) 92 (2018) 139–154, http://dx.doi.org/10.1016/j.ijar.2017.09.008, URL http://www.sciencedirect.com/science/article/pii/S0888613X17300804.

[100] Dong Liu, Hong-Yu Bai, Hui-Jia Li, Wen-Jun Wang, Semi-supervised community detection using label propagation, Internat. J. Modern Phys. B 28 (29) (2014) 1450208.

[101] J. Meng, D. Fu, T. Yang, Semi-supervised soft label propagation based on mass function for community detection, in: 2018 21st International Conference on Information Fusion (FUSION), 2018, pp. 1163–1170, http://dx.doi.org/10.23919/ICIF.2018.8455696.

[102] Christian L. Staudt, Henning Meyerhenke, Engineering parallel algorithms for community detection in massive networks, IEEE Trans. Parallel Distrib. Syst. 27 (1) (2016) 171–184, http://dx.doi.org/10.1109/TPDS.2015.2390633.

[103] Jyothish Soman, Ankur Narang, Fast community detection algorithm with GPUs and multicore architectures, in: 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS), IEEE, Los Alamitos, CA, USA, 2011, pp. 568–579, http://dx.doi.org/10.1109/IPDPS.2011.61.

[104] Konstantin Kuzmin, Mingming Chen, Boleslaw K. Szymanski, Parallelizing SLPA for scalable overlapping community detection, Sci. Program. 2015 (2015) 461362, http://dx.doi.org/10.1155/2015/461362.

[105] Qishan Zhang, Qirong Qiu, Wenzhong Guo, Kun Guo, Naixue Xiong, A social community detection algorithm based on parallel grey label propagation, Comput. Netw. 107 (Part 1) (2016) 133–143, http://dx.doi.org/10.1016/j.comnet.2016.06.002.

[106] Jierui Xie, Boleslaw K. Szymanski, Community detection using a neighborhood strength driven label propagation algorithm, in: Network Science Workshop, IEEE Computer Society, Washington, DC, USA, 2011, pp. 188–195, http://dx.doi.org/10.1109/NSW.2011.6004645.

[107] Naiyue Chen, Yun Liu, Haiqiang Chen, Junjun Cheng, Detecting communities in social networks using label propagation with information entropy, Physica A 471 (2017) 788–798, http://dx.doi.org/10.1016/j.physa.2016.12.047.

[108] Heli Sun, Jianbin Huang, Xiang Zhong, Ke Liu, Jianhua Zou, Qinbao Song, Label propagation with alpha-degree neighborhood impact for network community detection, Comput. Intell. Neurosci. 2014 (2014) 130689, http://dx.doi.org/10.1155/2014/130689.

[109] Jia Hou Chin, Kuru Ratnavelu, Detecting community structure by using a constrained label propagation algorithm, PLoS One 11 (5) (2016) e0155320.

[110] Ehsan Jokar, Mohammad Mosleh, Community detection in social networks based on improved label propagation algorithm and balanced link density, Phys. Lett. A (ISSN: 0375-9601) 383 (8) (2019) 718–727, http://dx.doi.org/10.1016/j.physleta.2018.11.033, URL http://www.sciencedirect.com/science/article/pii/S0375960118311794.

[111] Heli Sun, Jiao Liu, Jianbin Huang, Guangtao Wang, Zhou Yang, Qinbao Song, Xiaolin Jia, CenLP: A centrality-based label propagation algorithm for community detection in networks, Physica A 436 (2015) 767–780, http://dx.doi.org/10.1016/j.physa.2015.05.080.

[112] Qian Li, Tao Zhou, Linyuan Lü, Duanbing Chen, Identifying influential spreaders by weighted LeaderRank, Physica A 404 (Supplement C) (2014) 47–55, http://dx.doi.org/10.1016/j.physa.2014.02.041.

[113] Yan Xing, Fanrong Meng, Yong Zhou, Mu Zhu, Mengyu Shi, Guibin Sun, A node influence based label propagation algorithm for community detection in networks, Sci. World J. 2014 (2014) 627581, http://dx.doi.org/10.1155/2014/627581.

[114] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, Eran Shir, A model of Internet topology using *k*-shell decomposition, Proc. Natl. Acad. Sci. 104 (27) (2007) 11150–11154, http://dx.doi.org/10.1073/pnas.0701175104.

[115] Jierui Xie, Mingming Chen, Boleslaw K. Szymanski, LabelRankT: Incremental community detection in dynamic networks via label propagation, in: Proceedings of the Workshop on Dynamic Networks Management and Mining, ACM, New York, NY, USA, 2013, pp. 25–32, http://dx.doi.org/10.1145/2489247.2489249.

[116] Shenghong Li, Hao Lou, Wen Jiang, Junhua Tang, Detecting community structure via synchronous label propagation, Neurocomputing 151 (Part 3) (2015) 1063–1075b, http://dx.doi.org/10.1016/j.neucom.2014.04.084.

[117] Liang Zong-Wen, Li Jian-Ping, Yang Fan, Athina Petropulu, Detecting community structure using label propagation with consensus weight in complex network, Chin. Phys. B 23 (9) (2014) 098902, http://dx.doi.org/10.1088/1674-1056/23/9/098902.

[118] Xin Liu, Tsuyoshi Murata, Community detection in large-scale bipartite networks, in: Web Intelligence and Intelligent Agent Technologies, IET, 2009, pp. 50–57, http://dx.doi.org/10.1109/WI-IAT.2009.15.

[119] Martin Rosvall, Carl T. Bergstrom, An information-theoretic framework for resolving community structure in complex networks, Proc. Natl. Acad. Sci. 104 (18) (2007) 7327–7331, http://dx.doi.org/10.1073/pnas.0611034104.

[120] Tao Wu, Yuxiao Guo, Leiting Chen, Yanbing Liu, Integrated structure investigation in complex networks by label propagation, Physica A 448 (2016) 68–80, http://dx.doi.org/10.1016/j.physa.2015.12.073.

[121] Hao Peng, Dandan Zhao, Lin Li, Jianfeng Lu, Jianmin Han, Songyang Wu, An improved label propagation algorithm using average node energy in complex networks, Physica A 460 (2016) 98–104, http://dx.doi.org/10.1016/j.physa.2016.04.042.

[122] Rodrigo Francisquini, Valério Rosset, Mariá C.V. Nascimento, GA-LP: A genetic algorithm based on label propagation to detect communities in directed networks, Expert Syst. Appl. 74 (2017) 127–138, http://dx.doi.org/10.1016/j.eswa.2016.12.039.

[123] Xue Li, Directed lpa: Propagating labels in directed networks, Phys. Lett. A (ISSN: 0375-9601) 383 (8) (2019) 732–737, http://dx.doi.org/10.1016/j.physleta.2018.11.047, URL http://www.sciencedirect.com/science/article/pii/S0375960118311939.

[124] Sahar Kianian, Mohammad Reza Khayyambashi, Naser Movahhedinia, Semantic community detection using label propagation algorithm, J. Inf. Sci. 42 (2016) http://dx.doi.org/10.1177/0165551515592599.

[125] Ke Liu, Jianbin Huang, Heli Sun, Mengjie Wan, Yutao Qi, He Li, Label propagation based evolutionary clustering for detecting overlapping and non-overlapping communities in dynamic networks, Knowl.-Based Syst. 89 (2015) 487–496, http://dx.doi.org/10.1016/j.knosys.2015.08.015.

[126] Mohammad Sattari, Kamran Zamanifar, A cascade information diffusion based label propagation algorithm for community detection in dynamic social networks, J. Comput. Sci. (ISSN: 1877-7503) 25 (2018) 122–133, http://dx.doi.org/10.1016/j.jocs.2018.01.004, URL http://www.sciencedirect.com/science/article/pii/S1877750317303009.

[127] Andrea Clementi, Miriam Di Ianni, Giorgio Gambosi, Emanuele Natale, Riccardo Silvestri, Distributed community detection in dynamic graphs, Theoret. Comput. Sci. 584 (2015) 19–41, http://dx.doi.org/10.1016/j.tcs.2014.11.026.

[128] Chris Gaiteri, Mingming Chen, Boleslaw Szymanski, Konstantin Kuzmin, Jierui Xie, Changkyu Lee, Timothy Blanche, Elias Chaibub Neto, Su-Chun Huang, Thomas Grabowski, Tara Madhyastha, Vitalina Komashko, Identifying robust communities and multi-community nodes by combining top-down and bottom-up approaches to clustering, Sci. Rep. 5 (2015) http://dx.doi.org/10.1038/srep16361.

[129] Sun He-Li, Huang Jian-Bin, Tian Yong-Qiang, Song Qin-Bao, Liu Huai-Liang, Detecting overlapping communities in networks via dominant label propagation, Chin. Phys. B 24 (1) (2015) 018703, http://dx.doi.org/10.1088/1674-1056/24/1/018703.

[130] M. Lu, Z. Zhang, Z. Qu, Y. Kang, Lpanni: Overlapping community detection using label propagation in large-scale complex networks, IEEE Trans. Knowl. Data Eng. (ISSN: 1041-4347) (2018) 1, http://dx.doi.org/10.1109/TKDE.2018.2866424.

[131] Xiaolong Deng, Ying Wen, Yuanhao Chen, Highly efficient epidemic spreading model based LPA threshold community detection method, Neurocomputing 210 (2016) 3–12, http://dx.doi.org/10.1016/j.neucom.2015.10.142.

[132] Mohammad Sattari, Kamran Zamanifar, A spreading activation-based label propagation algorithm for overlapping community detection in dynamic social networks, Data Knowl. Eng. (ISSN: 0169-023X) 113 (2018) 155–170, http://dx.doi.org/10.1016/j.datak.2017.12.003, URL http://www.sciencedirect.com/science/article/pii/S0169023X16303500.

[133] Jihui Han, Wei Li, Weibing Deng, Multi-resolution community detection in massive networks, Sci. Rep. 6 (2016) 38998, http://dx.doi.org/10.1038/srep38998.

[134] Lovro Šubelj, Marko Bajec, Group detection in complex networks: An algorithm and comparison of the state of the art, Physica A 397 (2014) 144–156, http://dx.doi.org/10.1016/j.physa.2013.12.003.

[135] Yuxin Zhao, Shenghong Li, Shilin Wang, Agglomerative clustering based on label propagation for detecting overlapping and hierarchical communities in complex networks, Adv. Complex Syst. 17 (06) (2014) 1450021, http://dx.doi.org/10.1142/S0219525914500210.

[136] Gunter Dueck, New optimization heuristics: The great deluge algorithm and the record-to-record travel, J. Comput. Phys. (ISSN: 0021-9991) 104 (1) (1993) 86–92, http://dx.doi.org/10.1006/jcph.1993.1010, URL http://www.sciencedirect.com/science/article/pii/S0021999183710107.

[137] Ba-Dung Le, Hong Shen, Hung Nguyen, Nickolas Falkner, Improved network community detection using meta-heuristic based label propagation, Appl. Intell. (ISSN: 1573-7497) 49 (4) (2019) 1451–1466, http://dx.doi.org/10.1007/s10489-018-1321-0.

[138] Michele Coscia, Giulio Rossetti, Fosca Giannotti, Dino Pedreschi, DEMON: a local-first discovery method for overlapping communities, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2012, pp. 615–623, http://dx.doi.org/10.1145/2339530.2339630.

[139] Venu Satuluri, Srinivasan Parthasarathy, Scalable graph clustering using stochastic flows: Applications to community discovery, in: Proceedings of the Fifteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2009, pp. 737–746, http://dx.doi.org/10.1145/1557019.1557101.

[140] Alex Rodríguez, Alessandro Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.

[141] Jiajun Ding, Xiongxiong He, Junqing Yuan, Yan Chen, Bo Jiang, Community detection by propagating the label of center, Physica A (ISSN: 0378-4371) 503 (2018) 675–686, http://dx.doi.org/10.1016/j.physa.2018.02.174, URL http://www.sciencedirect.com/science/article/pii/S0378437118302632.

[142] Dongqing Zhou, Xing Wang, A neighborhood-impact based community detection algorithm via discrete PSO, Math. Probl. Eng. 2016 (2016) 3790590, http://dx.doi.org/10.1155/2016/3790590.

[143] Maoguo Gong, Qing Cai, Xiaowei Chen, Lijia Ma, Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition, IEEE Trans. Evol. Comput. 18 (1) (2014) 82–97, http://dx.doi.org/10.1109/TEVC.2013.2260862.

[144] Peng Wu, Li Pan, Multi-objective community detection based on memetic algorithm, PLoS One 10 (5) (2015) e012684, http://dx.doi.org/10.1371/journal.pone.0126845.

[145] Mohamed Guendouz, Abdelmalek Amine, Reda Mohamed Hamou, A discrete modified fireworks algorithm for community detection in complex networks, Appl. Intell. 46 (2) (2017) 373–385, http://dx.doi.org/10.1007/s10489-016-0840-9.

[146] Dongxiao He, Jie Liu, Bo Yang, Yuxiao Huang, Dayou Liu, Di Jin, An ant-based algorithm with local optimization for community detection in large-scale networks, Adv. Complex Syst. 15 (08) (2012) 1250036, http://dx.doi.org/10.1142/S0219525912500361.
[147] Reza Badie, Abolfazl Aleahmad, Masoud Asadpour, Maseud Rahgozar, An efficient agent-based algorithm for overlapping community detection using nodes' closeness, Physica A 392 (20) (2013) 5231–5247, http://dx.doi.org/10.1016/j.physa.2013.06.056.
[148] Paolo Boldi, Marco Rosa, Massimo Santini, Sebastiano Vigna, Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks, in: Proceedings of the 20th International Conference on World Wide Web, ACM, New York, NY, USA, 2011, pp. 587–596, http://dx.doi.org/10.1145/1963405.1963488.
[149] Peter Ronhovde, Zohar Nussinov, Local resolution-limit-free Potts model for community detection, Phys. Rev. E 81 (4) (2010) 046114, http://dx.doi.org/10.1103/PhysRevE.81.046114.
[150] Ling Chen, Qiang Yu, Bolun Chen, Anti-modularity and anti-community detecting in complex networks, Inform. Sci. 275 (2014) 293–313, http://dx.doi.org/10.1016/j.ins.2014.02.040.
[151] Yuxin Zhao, Shenghong Li, Feng Jin, Identification of influential nodes in social networks with community structure based on label propagation, Neurocomputing 210 (2016) 34–44, http://dx.doi.org/10.1016/j.neucom.2015.11.125.
[152] Ronghua Shang, Shuang Luo, Yangyang Li, Licheng Jiao, Rustam Stolkin, Large-scale community detection based on node membership grade and sub-communities integration, Physica A 428 (2015) 279–294, http://dx.doi.org/10.1016/j.physa.2015.02.004.
[153] L. Peng, R. Lin, Fraud phone calls analysis based on label propagation community detection algorithm, in: 2018 IEEE World Congress on Services (SERVICES), 2018, pp. 23–24, http://dx.doi.org/10.1109/SERVICES.2018.00025.
[154] Leto Peel, Daniel B. Larremore, Aaron Clauset, The ground truth about metadata and community detection in networks, Sci. Adv. 3 (5) (2017) http://dx.doi.org/10.1126/sciadv.1602548, URL https://advances.sciencemag.org/content/3/5/e1602548.
[155] Darko Hric, Richard K. Darst, Santo Fortunato, Community detection in networks: Structural communities versus ground truth, Phys. Rev. E 90 (2014) 062805, http://dx.doi.org/10.1103/PhysRevE.90.062805, URL https://link.aps.org/doi/101103/PhysRevE.90062805.
[156] Aric A. Hagberg, Daniel A. Schult, Pieter J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Gäel Varoquaux, Travis Vaught, Jarrod Millman (Eds.), Proceedings of the 7th Python in Science Conference, 2008, pp. 11–15.
[157] Thomas Aynaud, Louvain community detection, December 2018. URL https://github.com/taynaud/python-louvain.
[158] Pál Erdős, Alfréd Rényi, On Random Graphs I, Vol. 2, Akadémiai Kiadó, 1976, pp. 308–315, First publication in Publ. Math. Debrecen 1959.
[159] Pál Erdős, Alfréd Rényi, On the Evolution of Random Graphs, Vol. 2, Akadémiai Kiadó, 1976, pp. 482–525, First publication in MTA Mat. Kut. Int. Közl. 1960.
[160] Mathew D. Penrose, Random Geometric Graphs, Oxford University Press, Oxford, UK, 2003.
[161] Duncan J. Watts, Steven H. Strogatz, Collective dynamics of 'small world' networks, Nature 393 (6684) (1998) 440–444, http://dx.doi.org/10.1038/30918.
[162] Albert-László Barabási, Réka Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512, http://dx.doi.org/10.1126/science.286.5439.509.