Rossetti *et al. Applied Network Science* (2019) 4:52
https://doi.org/10.1007/s41109-019-0165-9

Applied Network Science

RESEARCH Open Access

CDLIB: a python library to extract, compare and evaluate communities from complex networks

Giulio Rossetti[1*], Letizia Milli[1,2] and Rémy Cazabet[3]

*Correspondence:
giulio.rossetti@isti.cnr.it
[1]KDD Lab. ISTI-CNR, via G. Moruzzi, 1, Pisa, Italy
Full list of author information is available at the end of the article

Abstract

Community Discovery is among the most studied problems in complex network analysis. During the last decade, many algorithms have been proposed to address such task; however, only a few of them have been integrated into a common framework, making it hard to use and compare different solutions. To support developers, researchers and practitioners, in this paper we introduce a python library - namely CDLIB - designed to serve this need. The aim of CDLIB is to allow easy and standardized access to a wide variety of network clustering algorithms, to evaluate and compare the results they provide, and to visualize them. It notably provides the largest available collection of community detection implementations, with a total of 39 algorithms.

Keywords: Social network analysis, Community discovery library, Community detection framework

Introduction

In the last decades, the analysis of complex networks has received increasing attention from several, heterogeneous fields of research. This popularity comes from the flexibility offered by such an approach: networks can be used to model countless phenomena with a common analytical framework whose basic bricks are nodes and their relations.

Social relationships, trading, transportation and communication infrastructures, even the brain can be modeled as networks and, as such, analyzed using what is now called *network science*. Undoubtedly, such pervasiveness has produced an amplification in the visibility of network analysis studies, thus making this complex and interesting field widespread among higher education centers, universities and academics. Given the exponential diffusion reached by network science, several tools were developed to make it approachable to the wider audience possible. Network analysis programming libraries are nowadays available to computer scientists, physicists as well as mathematicians; moreover, graphical tools were developed for social scientists, biologists as well as for educational purposes.

One of the hottest topics in network science is the Community Discovery, the task of clustering network entities belonging to topological dense regions of a graph.

Although many methods and algorithms have been proposed to cope with this problem, and related issues such as their evaluation and comparison, few of them are integrated

into a common software framework, making hard and time-consuming to use, study and compare them. Only a handful of the most famous methods are available in generic libraries such as NetworkX and Igraph, and running any other method requires to:

1 Find a reliable implementation
2 Learn how to use it
3 Transform the graph to study in the requested format
4 Export and transform the resulting clustering in a format suitable to the user needs.

This laborious process is probably the cause of two strong weaknesses of the Community Discovery field:

- Despite the large number of algorithms published every year, most of the newly proposed ones are compared only to a few classic methods.
- Practitioners barely ever try different methods on their data, while it is well known in the field that different methods often provide widely different solutions.

To cope with these issues - as previously done also in the network diffusion context with NDLIB (Rossetti et al. 2018) - we introduce a novel library designed to easily select/apply community discovery methods on network datasets, evaluate/compare the obtained clustering and visualize the results.

CDLIB represents a comprehensive, easy to use solution for network clustering. This paper aims to introduce CDLIB, describing its main features, and placing it among other tools already available to social network analysis practitioners.

The paper is organized as follows. First we briefly introduce some concepts needed to frame the context CDLIB is designed to analyze. Moving from the Community Discovery problem definition, we introduce CDLIB: there we describe how the library is designed, its rationale and main characteristics. Once made clear the key features of CDLIB, we identify and discuss the available competitors of the library. Finally we conclude the paper, underlining the advantages of CDLIB w.r.t. its competitors and providing insights on the future evolution of our framework. An appendix to this paper briefly describes the models made available by CDLIB v0.1.0.

Community discovery

Community discovery (henceforth CD), the task of decomposing a complex network topology into meaningful node clusters, is one of the oldest and most discussed problems in complex network analysis (Coscia et al. 2011; Fortunato 2010). One of the main reasons behind the attention such task has received during the last decades lies in its intrinsic complexity, strongly tied to its overall ill-posedness. Indeed, one of the few universally accepted axioms characterizing this research field regards the impossibility of providing a single shared definition of what community should look like. Usually, every CD approach is designed to provide a different point of view on how to partition a graph: in this scenario, the solutions proposed by different authors were often proven to perform well when specific assumptions can be made on the analyzed topology. Nonetheless, decomposing a complex structure in a set of meaningful components represents *per se* a step required by several analytical tasks – a need that has transformed what usually is considered a problem definition weakness, the existence of multiple partition criteria, into one of its major strength. Such peculiarity has lead to the definition of several "meta" community definitions, often tied to specific analytical needs. For instance, classic works intuitively

describe communities as sets of nodes closer among them than with the rest of the network, while others, looking at the same problem from another angle, only define such topologies as dense network subgraphs.

A general, high-level, formulation of the Community Discovery problem definition is as follows:

Definition 1 (Community Discovery (CD)) *Given a network G, a community C is defined as a set of distinct nodes:* $C = \{v_1, v_2, \ldots, v_n\}$. *The community discovery problem aims to identify the set* $\mathcal{C}$ *of all the communities in G.*

The absence of a unique, well-posed, definition of what a community in a complex network should represent is only one of the issues to face when approaching network clustering.

Such an ambiguous problem definition is nonetheless one of the causes for which researchers continuously propose novel approaches with the aim of solving well-defined instantiation of this task - often aiming at solving context-specific applications.

Due to the massive literature available in this field, over the years several attempts were made to organize and cluster methods identifying some common grounds. Among others, the surveys of Fortunato (Fortunato 2010; Fortunato and Hric 2016) and Coscia (Coscia et al. 2011) propose complete, detailed and extensive taxonomies for classic algorithms. However, due to the peculiar problem definition, more thematic surveys emerged, focusing for instance on overlapping (Xie et al. 2013), directed (Malliaros and Vazirgiannis 2013), node-centric (Rossetti et al. 2017) as well as dynamic community discovery (Cazabet et al. 2018; Rossetti and Cazabet 2018).

Another category of research contributions aims at comparing empirically existing methods on real or synthetic networks, as has been done for instance in Lancichinetti and Fortunato (2009); Leskovec et al. (2010); Harenberg et al. (2014). Indeed, since most methods do not share a common definition of communities to find, the only way to compare the quality of discovered partitions is to compare their results with a known ground truth, or to evaluate their quality using quality functions. With no standard library implementing various algorithm, each of these works require to re-implement methods and/or to adapt each existing implementation to a single common format. As a consequence, those empirical evaluations are only able to compare a limited number of well-known methods, and compare them with one or two evaluation metrics.

CDlib: community discovery library

We designed CDLIB - *"(C)ommunity (D)iscovery Library"* - to simplify the definition/execution/evaluation of community discovery analysis. CDLIB is a Python package built upon the network facilities offered by NetworkX[1] and Igraph[2]. The library, available for Python 3.x, is currently hosted on GitHub[3], on pypi[4] and has its online documentation on ReadTheDocs[5]. Moreover, CDLIB is also made available through the SoBigData.eu catalogue[6].

[1] NetworkX: https://goo.gl/PHXdnL
[2] Igraph: https://goo.gl/Hi5Srf
[3] CDLIB GitHub: https://goo.gl/Gu3VSV
[4] CDLIB pypi: https://goo.gl/FPtHHU
[5] CDLIB docs: https://goo.gl/ggGbUz
[6] SoBigData: http://www.sobigdata.eu

A complete list of the diffusion models implemented in CDLIB v0.1.0, along with their short descriptions, is reported in the Appendix. At the date of publication, CDLIB provides **39** implementations of CD algorithms, including **14** overlapping, **1** fuzzy, and **2** edge partitions methods.

The main features of the library are as follows.

- Implementation of a wide range of algorithms for community detection, including overlapping, fuzzy and edge clusterings.
- Standardized representation for both graphs and clusterings.
- Tools to efficiently compare methods when varying their parameters, or methods between themselves.
- Implementations of a variety of scores and quality functions to evaluate the quality of individual communities and whole clusterings.
- Visualization tools to compare and analyze clusterings obtained by one or several methods.

In the next sections, we present in details each of those capabilities.

Library rationale

The library provides several community detection algorithms (Refer to the Appendix for a complete list), implemented such as (i) they take as input a unified graph topology representation (ii) they return a clustering using a unified representation.

- The graph topology is implemented by borrowing the entities exposed by both the NetworkX and Igraph libraries. CDLIB allows to call all its CD algorithms equivalently on graph instances belonging to those libraries: it performs, if needed, all the required data type conversions under the hood without requiring the user to be aware of them.
- Apart from the graph object and method parameters needed to instantiate a specific CD algorithm, all the exposed models are designed to return objects having as supertype a common class, namely *Clustering*. Such abstract type exposes several facilities that will be discussed further on.

The standardization of clustering representation - and the decoupling of input/output w.r.t. algorithmic implementations - makes easy to extend CDLIB with novel algorithms. Any CD method written in python can be included in our library just wrapping it into an ad-hoc input/output harmonization process that:

- ensures the conversion, if needed, of a NetworkX/Igraph object into the graph representation used by the algorithm, and
- reshapes the algorithm results in a concrete class derived by `Clustering`.

The following code shows an example of experiment definition, configuration and execution.

Listing 1 Example of the execution of a CD algorithm on a NetworkX Graph object.

```
1 from cdlib import algorithms
2 import networkx as nx
3
4 # Network topology
5 g = nx.karate_club_graph()
6
7 # Model execution
8 coms = algorithms.demon(g, epsilon=0.25)
```

In lines 1–2 are imported all the required modules; in line 5 the Zachary's Karate Club (Zachary 1977) `g` is built using NetworkX; in line 8 the *Demon* (Coscia et al. 2012; 2014)

`algorithm` is executed. As previously discussed, we can replace the NetworkX graph object with the equivalent Igraph one while preserving the final result.

Listing 2 Example of the execution of a CD algorithm on an Igraph Graph object.

```
from cdlib import algorithms
import igraph as ig

# Network topology
g = ig.Nexus.get("karate")

# Model execution
coms = algorithms.demon(g, epsilon=0.25)
```

Once computed the desired network clustering, CDLIB allows its users to:

- evaluate it using several fitness scores
- compare it with alternative partitions
- visualize it using predefined and standard graphic facilities

Network Clustering

As already discussed, the results of each CD algorithm in CDLIB is an object that inherits from the abstract class *Clustering*. This choice is due to the fact that our library has been designed to handle heterogenous clustering types. In particular, in its current release, CDLIB allows executing approaches that partition a graph over the node set (*NodeClustering*) as well as over the edge set (*EdgeClustering*). Moreover, our library explicitly handles three different sub-types of clustering:

- *Partitions* (Crisp clustering): each node (edge) belongs to a unique cluster (community);
- *Overlapping*: each node (edge) is allowed to belong to more than one community at the same time;
- *Fuzzy*: each node (edge) belongs to multiple communities with different level of involvement in each one of them.

This differentiation among clustering types allows to fulfill two main goals: (i) enable clustering specific evaluation methods/representation for each specific concrete class; (ii) make the CDLIB framework easily extendible to support, in future releases, even more specific clustering types (e.g., multiplex node/edge clustering, dynamic node/edge clustering).

For the sake of simplicity, let us consider as an example the structure of the object instance produced by a generic node clustering algorithm of our library. The `NodeClustering` object contains the following information: (i) the list of communities obtained; (ii) a reference to the original graph object; (iii) all the metadata regarding the community extraction process (i.e., community discovery algorithm name and parameter configuration); (iv) a flag that specifies if the clustering is overlapping or not; (v) the percentage of nodes that are involved into the clustering. `FuzzyNodeClustering` extends such information with a node-community allocation probability matrix to keep track of the probabilistic component of the final non-overlapping partition. `Clustering` objects make use of such information to enable specific fitness measures and community comparison scores that will be briefly discussed in the Section "Clustering Evaluation and Comparison".

Ensemble clustering

Usually, given a network to analyze, it is a good practice to test several community discovery algorithms (or variations w.r.t. their parameter values) to identify the best partition for the specific analytical goal/fitness measure.

CDLIB offers a range of built-in facilities to simplify this task. The `cdlib.ensemble` submodule allows automating the execution of multiple instances of community detection algorithm(s) with different sets of parameters. In particular, two different families of meta-functions are defined:

- *Pooling*: automate the pooled execution of multiple CD algorithms over the same graph;
- *Optimization*: automate the parameter space search for a given CD algorithm so to maximize (minimize) a given partition fitness score.

Indeed, pooling and optimization facilities can be combined together to design more complex behaviors (i.e., pooling different algorithms with the aim of comparing their best partitions w.r.t. a given quality score).

Listing 3 Example of multiple execution of two CD algorithm.

```
1  from cdlib import algorithms as al
2  from cdlib import ensemble as en
3  from cdlib import evaluation as ev
4  import networkx as nx
5
6  # Network topology
7  g = nx.karate_club_graph()
8
9  # Louvain
10 res = en.Parameter(name="resolution",start=0.1, end=1,step=0.1)
11 rnd = en.BoolParameter(name="randomize")
12 louvain_conf = [res,rnd]
13
14 # Angel
15 eps = en.Parameter(name="epsilon",start=0.1,end=1, step=0.1)
16 demon_conf = [eps]
17
18 methods = [al.louvain,al.demon]
19
20 # Models pooled execution
21 coms = en.pool(g,methods, [louvain_conf,demon_conf])
22
23 # Optimal Search Louvain
24 coms, mod = en.grid_search(graph=g, method=al.louvain,
25     parameters=[res,rnd],quality_score=ev.erdos_renyi_modularity,
26     aggregate=max)
```

In Example 3 is shown how to perform both pooling and optimal partition search using two facilities offered by `cdlib.ensemble`.

As a first step, lines 9–18 defines the parameter ranges of the two algorithms selected to partition the NetworkX graph (Louvain (Blondel et al. 2008) and Demon). *Parameters* and *BoolParameters* are named tuples defined in `cdlib.ensemble` that allow the submodule functions to generate parameter ranges for CD methods. Line 21 the function `ensemble.pool` executes the specified methods (leveraging all the parameter combinations previously expressed) on the input graph `g`.

In some scenarios, one's objective could be to search the (potentially complex) parameter space of an algorithm in order to discover automatically the clustering optimizing a given quality function (parameter-tuning). CDLIB allows doing so using several strategies, among them the function `grid_search`. An example of how to use such functionality is shown in line 24 where, for the Louvain algorithm, is returned the partition

having maximum modularity while varying the *resolution* and *randomize* parameter values.

In the current version of the library, `cdlib.ensemble` exposes the facilities reported in Table 1.

Clustering evaluation and comparison

As already mentioned, CDLIB allows not only to compute network clusterings applying several algorithmic approaches but also enables the analyst to characterize and compare the obtained results.

Clustering evaluation and comparison facilities are delegated to the `cdlib.evaluation` submodule (also referred by the `Clustering` objects). In Table 2 we reported the description of the symbols used in the formulas for fitness and comparison functions. The submodule provides several fitness scores, listed in Table 3, as well as clustering comparison measures, reported in Table 4. The former set can be used to get insights on the compactness and topological consistency of communities, while the latter allows measuring the level of similarity among partitions obtained by applying different algorithms or w.r.t. a golden ground truth. Indeed, evaluating communities considering solely the score they are able to achieve in a given fitness function has a major drawback: it favors methods that are designed to maximize it. Even though this type of strategy can be used fruitfully to compare methods that explicitly optimize a specific measure, its application to approaches that search for communities with a different definition may produce misleading or inconclusive/irrelevant comparisons. To cope with such an issue it is good practice to evaluate the similarity of different partitions in order to get insights on the different *families* of clusters different algorithms are able to unveil.

Listing 4 Example of community discovery algorithms comparison.

```
from cdlib import algorithms
import networkx as nx

# Network topology
g = nx.karate_club_graph()

# Models execution
louvain_coms = algorithms.louvain(g)
leiden_coms = algorithms.leiden(g)

# Modularity evaluation
louvain_mod = louvain_coms.erdos_renyi_modularity()
leiden_mod = leiden_coms.erdos_renyi_modularity()

# Clustering comparisons
nmi = louvain_coms.normalized_mutual_information(leiden_coms)
```

Table 1 Ensemble functionalities offered by CDLIB

Name	Description
`grid_execution`	Instantiate the specified community discovery method performing a grid search on the parameter set.
`pool`	Execute a pool of community discovery algorithms on the input graph.
`grid_search`	Returns the optimal partition of the specified graph w.r.t. the selected algorithm and quality score performing an exaustive grid search on the parameter space.
`random_search`	Returns the optimal partition of the specified graph w.r.t. the selected algorithm and quality score over a randomized sample of the parameters space.
`pool_grid_filter`	Execute a pool of community discovery algorithms on the input graph and returns the optimal partition for each algorithm given the specified quality function.

Table 2 Description of the symbols used in the formulas for fitness and comparison functions

Symbol	Description
C	partition of the graph
C_i	community ∈ partition C
b_C	number of edges on the community boundary
d_m	internal degree median value
e_C	number of community internal edges in C
k	the total degree
k_{iC}^{int}	the degree of node i within C
k_{iC}^{out}	the degree of node i outside C
l_C	number of edges from nodes in C to nodes outside C
M_{int}	total possible internal edges: $\sum_C \binom{n_C}{2}$
N_c	Total number of communities of all sizes detected by a given algorithm, $\sum_i n_c x_c^i$
n_C	number of community nodes in C
$\Gamma(i)$	degree of the node i
p	density of the graph
p_C	the density of community C. $p_C = m_C/\binom{n_C}{2}$
$\langle q \rangle$	the expected fraction of internal edges
x_c	number of communities having the same number of nodes of c
Coverage	percentage of communities in Y that are matched by at least an object in X $\frac{\lvert Y_{id}\rvert}{\lvert Y\rvert}$ where Y_{id} is the subset of communities in Y matched by community in X
$D(x\|\|y)$	KL divergence
D_c	the sum of the degrees of the vertices in community C
$\delta(c_i, c_j)$	indicator function: it assumes value 1 iff i and j belong to the same community, 0 otherwise
$\delta_1(n_a^i, n_b^j)$	indicator function: it assumes value 1 iff two communities a and b have the same number of nodes, $n_a^i = n_b^j$, 0 otherwise
$Exp(s1, s2)$	the expected agreement between solutions $s1$ and $s2$. $Exp(s1, s2) = \sum_{j=0}^{min(J,K)} N_{j1} N_{j2}/N^2$
$H(X)$	partition entropy of X
$H(X)$	the entropy of the random variable X associated to an algorithm community
$H(Y)$	the entropy of the random variable Y associated to a ground truth community
$H(X, Y)$	the joint entropy
$I(X : Y)$	mutual information $\frac{1}{2}[H(X) - H(X\|Y) + H(Y) - H(Y\|X)]$
$MI(X, Y)$	mutual information of X and Y
$Obs(s1, s2)$	the observed agreement between solutions $s1$ and $s2$. $Obs(s1, s2) = \sum_{j=0}^{min(J,K)} A_j/N$
Redundancy	percentage of communities in Y that are matched by at least an object in X $\frac{\lvert X\rvert}{\lvert Y_{id}\rvert}$ where Y_{id} is the subset of communities in Y matched by community in X

Example 5, shows how given two alternative partitions of a given graph it is possible, using CDLIB, to compare them both structurally - applying, for instance, the Normalized Mutual Information score (Lancichinetti et al. 2008) - and in terms of a specific fitness function - leveraging in our scenario the modularity index (Erdös and Rényi 1959).

To facilitate clustering comparisons, CDLIB also implements aggregate ranking solutions (e.g., TOPSIS - as proposed in Jebabli et al. (2018)). Such functionalities allow to, once obtained multiple fitness/comparison scores for a set of CD algorithms to generate an aggregate ranking to better summarize the results. Moreover, CDLIB also implements non-parametric statistical significance test (Friedman with Bonferroni-Dunn post-hoc (Demšar 2006)) to assess the reliability of produced rankings.

It must be noted that all implemented metrics cannot be applied to all the available CD algorithms: to address this issue, before executing any function defined in `cdlib.evaluation`, CDLIB takes care of checking its consistency w.r.t. the computed network clustering type.

Table 3 Fitness functions implemented in CDLIB

Name	Description	Formula	Reference
`average_internal_degree`	The average internal degree of the community set.	$\frac{2e_C}{n_C}$	(Radicchi et al. 2004)
`avg_distance`	The average path length across all possible pair of nodes composing the community set.		(Orman et al. 2012)
`avg_embeddedness`	The average embeddedness of nodes within the community.	$\frac{1}{n_C}\sum_{i\in C}\frac{k_{iC}^{int}}{\Gamma(i)}$	(Orman et al. 2012)
`avg_odf`	Average fraction of edges of a node of a community that point outside the community itself.	$\frac{1}{n_C}\sum_{u\in C}\frac{\lvert(u,v)\in E:v\notin C\rvert}{\Gamma(u)}$	(Flake et al. 2000)
`avg_transitivity`	The average clustering coefficient of its nodes w.r.t. their connection within the community itself.		(Orman et al. 2012)
`conductance`	Fraction of total edge volume that points outside the algorithms.	$\frac{n_C}{2e_C+n_C}$	(Shi and Malik 2000)
`cut_ratio`	Fraction of existing edges (out of all possible edges) leaving the community.	$\frac{n_C}{b_C(n-b_C)}$	(Fortunato 2010)
`edges_inside`	Number of edges internal to the community.	e_C	(Radicchi et al. 2004)
`expansion`	Number of edges per community node that point outside the cluster.	$\frac{n_C}{b_C}$	(Radicchi et al. 2004)
`flake_odf`	Fraction of nodes of the clustering that have fewer edges pointing inside than to the outside of their communities.	$\frac{u:u\in C,\lvert(u,v)\in E:v\in C\rvert<\lvert\Gamma(u)/2\rvert}{n_C}$	(Flake et al. 2000)
`fraction_over_median_degree`	raction of community nodes having internal degree higher than the median degree value.	$\frac{u:u\in C,\lvert(u,v):v\in C\rvert>\lvert d_m\rvert}{n_C}$	(Yang and Leskovec 2015)
`hub_dominance`	The ratio of the degree of its most connected node w.r.t. the theoretically maximal degree within the community.	$\frac{max_{i\in C}(k_{iC}^{int})}{n_C-1}$	(Orman et al. 2012)
`internal_edge_density`	The internal density of the community set.	$\frac{e_C}{n_C(n_C-1)/2}$	(Radicchi et al. 2004)
`max_odf`	Maximum fraction of edges of a node of a community that point outside the community itself.	$max_{u\in C}\frac{\lvert(u,v)\in E:v\notin C\rvert}{\Gamma(u)}$	(Flake et al. 2000)
`normalized_cut`	Normalized variant of the Cut-Ratio.	$\frac{n_C}{2e_C+n_C}+\frac{n_C}{2(\lvert E\rvert-e_C)+n_C}$	(Shi and Malik 2000)
`scaled_density`	The ratio of the community density w.r.t. the complete graph density.	$\frac{2\lvert E\rvert}{n-1}$	(Orman et al. 2012)
`significance`	Estimate the likelihood that the identified partition appears in a random graph.	$\sum_c\binom{n_c}{2}D(p_C\lvert\lvert p)$	(Traag et al. 2015)
`size`	Number of community nodes.		
`surprise`	Statistical approach that assumes that edges emerge randomly according to a hyper-geometric distribution: the higher the surprise, the less likely the clustering is resulted from a random realization.	$-log\sum_{i=m_C}^{min(\lvert E\rvert,M_{int})}\binom{\lvert E\rvert}{i}\langle q\rangle^i(1-\langle q\rangle)^{\lvert E\rvert-i}$	(Traag et al. 2015)
`triangle_participation_ratio`	Fraction of community nodes that belong to a triad.	$\frac{\lvert u:u\in C,(v,w):v,w\in C,(u,v)\in E,(u,w)\in E,(v,w)\in E\neq\emptyset\rvert}{n_C}$	(Yang and Leskovec 2015)

Table 3 Fitness functions implemented in CDLIB

Name	Description	Formula	Reference
`erdos_renyi_modularity`	Variation of the Newman-Girvan modularity that assumes that nodes in a network connected randomly with a constant probability p.	$\frac{1}{\lvert E\rvert}\sum_{C\in\{C_1,\dots C_k\}}(e_C - \frac{\lvert E\rvert n_C(n_C-1)}{n(n-1)})$	(Erdös and Rényi 1959)
`link_modularity`	Variation of the Girvan-Newman modularity for directed graphs with overlapping communities.	$\frac{1}{\lvert E\rvert}\sum_{i,j\in C}[A_{ij}\delta(c_i,c_j) - \frac{k_i C^{out} k_j C^{in}}{m}\delta(c_i,c_j)]$	(Nicosia et al. 2009)
`modularity_density`	Variation of the Erdos-Renyi modularity that includes information about community sizes into the expected density coefficient so to avoid the negligence of small and dense communities.	$\sum_{C\in\{C_1,\dots C_k\}}\frac{1}{n_C}(\sum_{i\in C}k_{iC}^{int} - \sum_{i\in C}k_{iC}^{out})$	(Li et al. 2016)
`newman_girvan_modularity`	Difference of the fraction of intra community edges of a clustering with the expected number of such edges if distributed according to a null model.	$\frac{1}{\lvert E\rvert}\sum_{C\in\{C_1,\dots C_k\}}(e_C - \frac{(2e_C+l_C)^2}{4\lvert E\rvert})$	(Newman and Girvan 2004)
`z_modularity`	Variant of the standard modularity proposed to avoid the resolution limit.	$\frac{\sum_{C\in\{C_1,\dots C_k\}}\frac{m_C}{\lvert E\rvert} - \sum_{C\in\{C_1,\dots C_k\}}\left(\frac{D_C}{2\lvert E\rvert}\right)^2}{\sqrt{\sum_{C\in\{C_1,\dots C_k\}}\left(\frac{D_C}{2\lvert E\rvert}\right)^2\left(1-\sum_{C\in\{C_1,\dots C_k\}}\left(\frac{D_C}{2\lvert E\rvert}\right)^2\right)^2}}$	(Miyauchi and Kawase 2016)

The upper part of the table groups all the community-wise fitness scores for which the library allows to compute the overall distribution as well as standard statistical indexes. The lower part of the table groups the implemented modularity-based quality scores

Table 4 Clustering comparison functions implemented in CDLIB

Name	Description	Formula	Reference
`ami`	Adjusted Mutual Information is an adjustment of the Mutual Information score to account for chance.	$\frac{(X,Y)-E(MI(X,Y))}{max(H(X),H(Y))-E(MI(X,Y))}$	(Vinh et al. 2010)
`ari`	The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings.	$\frac{RI-Expected_{RI}}{max(RI-Expected_{RI})}$	(Hubert and Arabie 1985)
`closeness`	Closeness of community size distributions.	$\frac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{s}\min\left\{\frac{x_a(n_a^i)}{N^a},\frac{x_b\left(x_b^j\right)}{N_b}\right\}\delta_1\left(n_a^i,n_b^j\right)$	(Dao et al. 2018)
`f1`	Average F1 score (harmonic mean of Precision and Recall) of the optimal matches among the partitions in input. clustering.	$2\times\frac{precision\times recall}{precision+recall}$	(Rossetti et al. 2016)
`nf1`	Normalized version of F1 that corrects the resemblance score taking into account degree of node overlap and clutering coverage.	$\frac{F1\times Coverage}{Redundancy}$	(Rossetti 2017),(Rossetti et al. 2016)
`nmi`	Normalized Mutual Information (NMI) is an normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation)	$\frac{H(X)+H(Y)-H(X,Y)}{(H(X)+H(Y))/2}$	(Lancichinetti et al. 2009)
`onmi-LFK`	Original extension of the Normalized Mutual Information (NMI) score to cope with overlapping partitions.	$1-\frac{1}{2}\left(\frac{H(X\|Y)}{H(X)}+\frac{H(Y\|X)}{H(Y)}\right)$	(Lancichinetti et al. 2009)
`onmi-MGH`	Extension of the Normalized Mutual Information (NMI) score to cope with overlapping partitions, based on max normalization.	$\frac{I(X:Y)}{max(H(X),H(Y))}$	(McDaid et al. 2011)
`omega`	Resemblance index defined for overlapping, complete coverage, clusterings.	$\frac{Obs(s1,s2)-Exp(s1,s2)}{1-Exp(s1,s2)}$	(Murray et al. 2012)
`vi`	Variation of Information among two nodes partitions.	$H(X)+H(Y)-2MI(X,Y)$	(Meilă 2007)

Visualization Facilities

To allow the final user visualising clustering results, CDLIB exposes a set of predefined visual facilities using Matplotlib[7]. These facilities are exposed through the visualization submodule `cdlib.viz`. Such submodule offers two different classes of visualization:

Network Visualization, that allows plotting a graph with node color coding for communities (Fig. 1).

Listing 5 Example of community network visualization (Results shown in Fig. 1)

```
from cdlib import algorithms, viz
import networkx as nx

g = nx.karate_club_graph()
coms = algorithms.louvain(g)
pos = nx.spring_layout(g)
viz.plot_network_clusters(g, coms, pos)
viz.plot_community_graph(g, coms)
```

Analytics plots, where community evaluation outputs can be easily used to generate a visual representation of the main partition characteristics (Fig. 2).

All the plots are generated taking as inputs `Clustering` objects that contain all the required functionalities and metadata to customize the final result. The example reported in Fig. 2 cover only a small subset of the possible analytical views that can be generated leveraging all the fitness/comparisons measures offered by CDLIB.

CDLIB-REST: web service

The facilities offered by CDLIB are specifically designed for those users that want to run experiments on their local machine. However, in some scenarios, e.g., due to limited computational resources or to the rising of other needs, it may be convenient to separate the machine on which the definition of the experiment is made from the one that actually executes the simulation. In order to satisfy such needs - as already done with NDLIB (Rossetti et al. 2018) for the simulation of diffusive phenomena - we developed a RESTfull service, CDLIB-REST[8], that builds upon CDLIB an experiment server queryable through API calls.

CDlib-REST rationale. The web service is designed around the concept of the *experiment*. Every experiment is identified by a unique identifier and it is composed of two entities: (i) a network and (ii) one (or more) CD algorithm(s).

In particular, in order to perform an experiment, a user must:

1. Request a token, which univocally identifies the experiment;
2. Load or select a network resource;
3. Select one, or more, CD algorithm(s);
4. Set the parameters for the CD algorithm(s);
5. Execute the CD algorithm(s);
6. Evaluate and/or compare CD algorithm(s);
7. (optional) Reset the experiment status, modify the algorithms/network;
8. Destroy the experiment.

Figure 3 shows an example of the workflow for CDLIB-REST. The last action, involving the destruction of the experiment, is designed to clean the serialization made by

[7] Matplotlib: https://goo.gl/EY96HV
[8] CDLIB-REST: https://github.com/GiulioRossetti/cdlib_rest

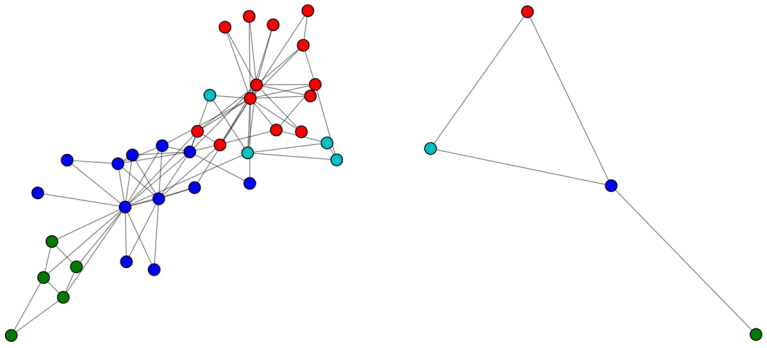

Fig. 1 Visual Analitycs. `Network Visualization` for communities obtained with Louvain method on the Zachary's Karate Club graph. CDLIB allows to: (left) visualize communities on the original graph by identifying them using the same color palette for the nodes; (right) collapse each community in a single node and visualize the community connection graph

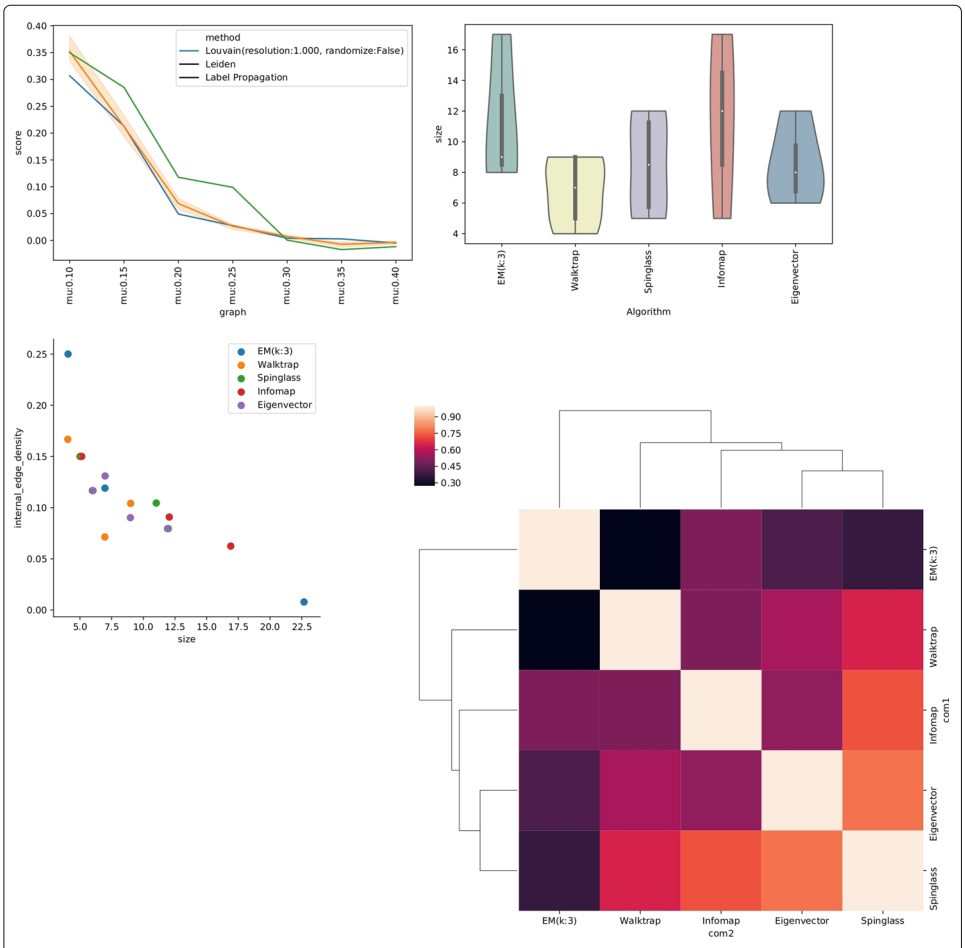

Fig. 2 Visual Analitycs. `Analytics plots` for CD algorithm. First line: (left) is the plot of the scores (in this case obtained with `adjusted_mutual_information`) obtained by a list of methods on a list of graphs (on LFR benchmark graphs (Lancichinetti et al. 2008)); (right) is the plot of the distribution of a property (e.g. `size`) among all communities for a clustering, or a list of clusterings on the Zachary's Karate Club graph. Second line: (left) is the plot of the relation between the two functions `size` and `internal_edge_density` of CD algorithms on the Zachary's Karate Club graph; (right) is the plot of the similarity matrix between a list of clusterings, using the provided scoring function (in this case obtained with `adjusted_mutual_information`) on the Zachary's Karate Club graph

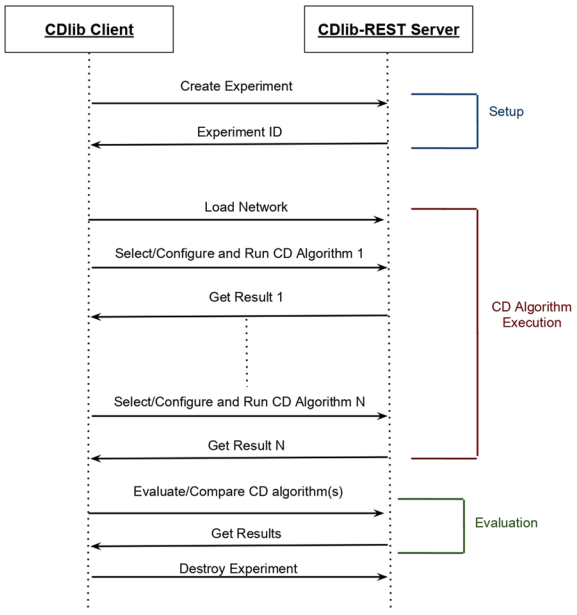

Fig. 3 CDLIB-REST example of workflow of CDLIB-REST

the service. If an experiment is not explicitly destroyed its data is removed, and the associated token invalidated, after a temporal window that can be configured by the service administrator. CDLIB-REST is built using aiohttp[9], that offers asynchronous request, and gives a standard online documentation page (shown in Fig. 4) that can also be directly used to test the exposed endpoints both configuring and running experiments.

Python API wrapper. In order to provide a simplified interface to query the CDLIB-REST service, we defined a Python wrapper that organizes and exposes all the implemented API. Such API wrapper, shipped along with the web service, allows to define and run remote experiments as shown in the example below:

Listing 6 Example of definiton and execution of remote CD algorithms

```
import networkx as nx
from cdlib_rest import CDlib_API

# Connecting the simulation service
with CDlib_API("http://0.0.0.0", 8081) as api:
    # Create the network
    g = nx.karate_club_graph()
    api.load_network(g)

    # Run the CD algorithm
    coms = api.demon(epsilon=0.25)
    # Evaluate the result
    stats = api.fitness_scores([coms], summary=False)
```

CDlib competitors

To the best of our knowledge, until now, no existing library includes large numbers of CD algorithms and corresponding measures to compare their results. Users that want to compare CD algorithms encounter several difficulties: (i) different programming languages, (ii) different input format for the graph file, (iii) different output format of the results.

[9]aiohttp: https://aiohttp.readthedocs.io/en/stable/

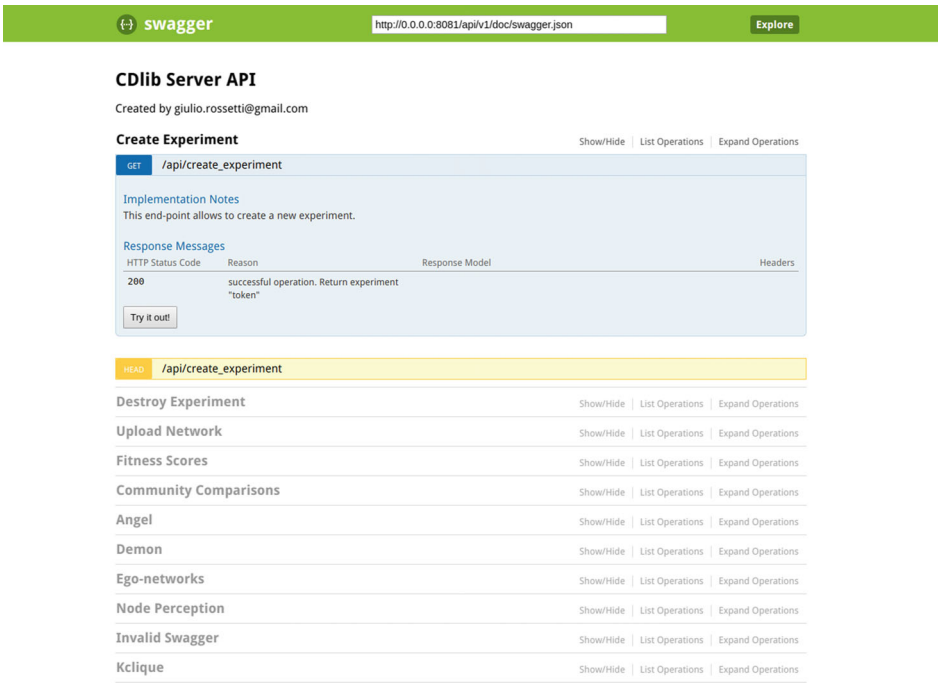

Fig. 4 CDlib-REST webpage of the documentation

Two widespread libraries, NetworkX and Igraph, have implemented a handful of the most famous CD algorithms and performance measures, but they represent only a minor fraction of existing algorithms and measures. CDLIB is the first library to provide a standardized input/output for a wide range of CD algorithms, encompassing not only those available through those two libraries, but also a large spectrum of methods they do not provide.

We can separate CDLIB alternatives in two main categories: (i) Collections of implementations (ii) Standalone libraries.

Collections of implementations

The most simple collections of implementations are repositories, listing of methods that can be implemented in different languages, with different input and outputs.

While those collections are helpful to discover existing implementation, the problem we stated in the introduction is still present: one needs to satisfy all the required implementation dependencies, learn how to instantiate each method, and format inputs and outputs to suit its needs.

Among such repositories, we can cite:

- RapidsAtHKUST[10]: a repository collecting some overlapping CD algorithms written in C++, Java and Python.

In other projects, wrappers are built over existing methods, so that they can be run more easily, notably in CoDACom [11]. The result is not a standalone library, but a collection of scripts allowing to run methods more conveniently.

[10] https://goo.gl/ADmvrQ
[11] https://codacom.greyc.fr/use.php

Standalone libraries

We already cited NetworX and Igraph, well-known network science libraries. Both of them contains several algorithms (5 for NetworkX, 9 for Igraph) and some quality functions to evaluate them.

A few other resources exist:

- CommunityDetection[12]: A python library, with no English documentation, this git-hub repository includes implementations for 7 well-known methods. It has not been maintained in the last three years.
- Circulo[13]: a Community Detection Evaluation Framework, is probably the closest competitor to CDLIB; it is a project started in 2014. The framework is written in Python, based on Igraph, and includes functionalities of data extract transform load (ETL) and performance metrics. The metrics include internal statistical measures of a community (i.e. density), external measurements (i.e. expansion), and network wide metrics (ground truth comparisons). The project has seen no activity in the last three years, and offer no documentation of available functionalities. It provided 17 algorithms; although some of them require the installation of additional tools and the compilation of additional code, due to the calling of external methods.
- SNAP[14]: Stanford Network Analysis Platform is a general purpose network analysis and graph mining library. The core is written in C++ and there is also a python version built as a wrapper around the C++ one. The project is active, provides two crisp algorithms (the Girvan-Newman and the Clauset-Newman-Moore method) and an overlapping one Affiliation Graph Model fitting (AGMfit): it requires the installation of additional tools.
- CommunityALG[15]: is a set of Matlab functions and algorithms for the community detection in networks that expands the BrainConnectivity toolbox[16]. This toolbox is widely used by brain-imaging researchers, and has been ported to many projects.
- CDTB[17]: the Community Detection ToolBox is a MATLAB toolbox which can be used to perform community detection. The CDTB contains several functions and includes graph generators, clustering algorithms (12 CD methods) and finally clustering evaluation functions. Furthermore, CDTB is designed in a parametric manner so that the user can add his own functions and extensions. The project has not been maintained in the last five years.
- Leidenalg[18]: this package implements the Leiden algorithm in C++ and exposes it to python. It scales well, and can be run on graphs of millions of nodes (as long as they can fit in memory). It works with direct, weighted and multiplex graphs and provides some support for community detection on bipartite graphs. It provides implementations for six different methods derived from Leiden algorithm; the methods currently implemented are: modularity, Reichardt and Bornholdt's model using the configuration null model and the Erdös-Rényi null model, the constant Potts model (CPM), Significance, and Surprise.

[12] goo.gl/btdzSS
[13] goo.gl/BybtKi
[14] http://snap.stanford.edu/
[15] https://github.com/carlonicolini/communityalg
[16] https://sites.google.com/site/bctnet/
[17] goo.gl/JCpPpF
[18] https://github.com/vtraag/leidenalg

- GraphTool[19]: is a Python module for manipulation and statistical analysis of networks. The core data structures and algorithms are implemented in C++; this confers it a level of performance that is comparable to that of a pure C/C++ library, both in memory usage and computation time. Many algorithms are implemented in parallel which provides excellent performance on multi-core architectures. In term of community detection, it supports only methods based on bayesian inference of stochastic block models. CDLIB include these methods by handling conversion to/from this library.

In order to compare the selected libraries we identify the following set of features:

- Programming Language. The choice of a programming language
- CD algorithms. The set of CD algorithm implemented in the library, with the distinction from crisp and overlapping models.
- Project Status. Whether the project is currently developed or if its support ceased.

Table 5 reports a characteristization of the selected libraries. We can observe that half of the analyzed libraries are not actively maintained at the moment of our survey. Moving to library related characteristics, we observe a clear pattern: all libraries offer out of the box support for crisp CD algorithms while few libraries, natively support overlapping ones.

Conclusions and future works

One of the main issues in current research is results reproducibility. In some specific contexts, like Community Discovery, we can observe how (i) the lack of (easily findable) algorithm implementations, and (ii) the absence of standard input/output formats represent the main causes for a significant stagnation of comparative analysis and replication studies. Such limitations can be witnessed in a common pattern: authors of CD algorithms usually compare their approaches only against a few classic methods whose implementation are easily findable online. We think that a way to address such issue is to provide a task dedicated library that: (i) allows researcher access to the widest possible set of models and algorithms, and (ii) acts as starting point for the integration of novel methods.

With this aim, we introduced CDLIB, a python library for community discovery that takes care of supporting its users from the model definition to the analysis/comparison and visualization of results. CDLIB primary effort focuses on providing a simple abstraction layer over the data model and in designing input/output specifications shared by both methods and evaluation functions.

CDLIB is an ongoing open project: we plan to further extend it by integrating novel algorithms (contributions are welcome), by supporting alternative clustering definitions (i.e., multiplex, evolving,...) and by integrating evaluation/visualization facilities.

Appendix A: Community discovery methods implemented in CDLIB

CDLIB exposes several community discovery algorithms, covering both node partition approaches and edge partition approaches. In particular, the actual release of the library (v0.1.0) implements the following algorithms:

[19] https://graph-tool.skewed.de/

Table 5 Number of methods available in the main competitors of CDLIB

Library	Language	Active	# algorithms		
			Total	Crisp	Overlapping
CDLIB	Python	✓	39	24	14
NetworkX	Python	✓	5	4	1
Igraph	C	✓	9	9	0
snap	Python/C++	✓	2	1	1
CDTB	Matlab		12	11	1
Circulo	Python		17	11	6
Leidenalg	C++		6	6	0
GraphTool	Python/C++	✓	4	2	2

Appendix B: Node partition.

Crisp communities

Agglomerative Clustering on a Directed Graph (AGDL): this method is a graph-based agglomerative algorithm, for clustering high-dimensional data introduced in Zhang et al. (2012). The algorithm uses the indegree and outdegree to characterize the affinity between two clusters.

Fluid this model was introduced in Parés et al. (2017) and is based on the simple idea of fluids (i.e., communities) interacting in an environment (i.e., a non-complete graph), expanding and contracting. It is a propagation-based algorithm and it allows to specify the number of desired communities (k) and it is asynchronous, where each vertex update is computed using the latest partial state of the graph.

Constant Potts Model (CPM) : this algorithm is a Leiden model (Traag et al. 2011) where the quality function to optimize is: $Q = \sum_{ij} \left(A_{ij} - \gamma\right) \delta(\sigma_i, \sigma_j)$ where A is the adjacency matrix, σ_i denotes the community of node i, $\delta(\sigma_i, \sigma_j) = 1$ if $\sigma_i = \sigma_j$ and 0 otherwise, and, finally γ is a resolution parameter. The internal density of communities $p_c = \frac{m_c}{\binom{n_c}{2}} \geq \gamma$ is higher than γ, while the external density $p_{cd} = \frac{m_{cd}}{n_c n_d} \leq \gamma$ is lower than γ. In other words, choosing a particular γ corresponds to choosing to find communities of a particular density, and as such defines communities. Finally, the definition of a community is in a sense independent of the actual graph, which is not the case for any of the other methods.

Diffusion Entropy Reducer (DER) is a clustering algorithm introduced in Kozdoba and Mannor (2015). The algorithm uses random walks to embed the graph in a space of measures, after which a modification of k-means in that space is applied. It creates the walks, creates an initialization, runs the algorithm, and finally extracts the communities.

Eigenvector : it is the Newman's leading eigenvector method for detecting community structure based on modularity (Newman 2006). This is the proper internal of the recursive, divisive algorithm: each split is done by maximizing the modularity regarding the original network.

Expectation–Maximization (EM) : this method is based on based on a mixture model (Newman and Leicht 2007). The algorithm uses the expectation–maximization algorithm to detect structure in networks.

Gdmp2 : it is a method for identifying a set of dense subgraphs of a given sparse graph (Chen and Saad 2012). It is inspired by an effective technique designed for a similar problem—matrix blocking, from a different discipline (solving linear systems).

Girvan–Newman : this algorithm detects communities by progressively removing edges from the original graph (Girvan and Newman 2002). The algorithm removes the "most valuable" edge, traditionally the edge with the highest betweenness centrality, at each step. As the graph breaks down into pieces, the tightly knit community structure is exposed and the result can be depicted as a dendrogram.

Greedy modularity : this algorithm uses the modularity to find the communities structures (Clauset et al. 2004). At every step of the algorithm two communities that contribute maximum positive value to global modularity are merged.

Infomap : it is based on ideas of information theory (Rosvall and Bergstrom 2008). The algorithm uses the probability flow of random walks on a network as a proxy for information flows in the real system and it decomposes the network into modules by compressing a description of the probability flow. This method is not implemented internally but we use the external implementation[20].

Label Propagation Algorithm (LPA) : this algorithm detects communities using network structure alone (Raghavan et al. 2007). The algorithm doesn't require a pre-defined objective function or prior information about the communities. It works as follows: (i) every node is initialized with a unique label (an identifier) (ii) these labels propagate through the network (iii) at every iteration of propagation, each node updates its label to the one that the maximum numbers of its neighbors belong to. Ties are broken uniformly and randomly. (iv) LPA reaches convergence when each node has the majority label of its neighbors.

Leiden : this algorithm (Traag et al. 2018) is an improvement of the Louvain algorithm. The Leiden algorithm consists of three phases: (1) local moving of nodes, (2) refinement of the partition (3) aggregation of the network based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network. This method is not implemented internally but we use the external implementation[21].

Louvain : this method maximizes a modularity score for each community (Blondel et al. 2008). The algorithm optimizes the modularity in two elementary phases: (1) local moving of nodes; (2) aggregation of the network. In the local moving phase, individual nodes are moved to the community that yields the largest increase in the quality function. In the aggregation phase, an aggregate network is created based on the partition obtained in the local moving phase. Each community in this partition becomes a node in the aggregate network. The two phases are repeated until the quality function cannot be increased further. This method is not implemented internally but we use the external implementation[22].

[20] https://pypi.org/project/infomap/
[21] https://github.com/vtraag/leidenalg
[22] https://github.com/taynaud/python-louvain

Rber pots : it is a Leiden model where the quality function to optimize is: $Q = \sum_{ij} \left(A_{ij} - \gamma p\right) \delta(\sigma_i, \sigma_j)$ where A is the adjacency matrix, $p = \frac{m}{\binom{n}{2}}$ is the overall density of the graph,σ_i denotes the community of node$i$, $\delta(\sigma_i, \sigma_j) = 1$ if $\sigma_i = \sigma_j$ and 0 otherwise, and, finallyγ is a resolution parameter. The method was introduced in Reichardt and Bornholdt (2006). This method is not implemented internally but we use the external implementation[23].

Rb pots is a Leiden model where the quality function to optimize is: $Q = \sum_{ij} \left(A_{ij} - \gamma \frac{k_i k_j}{2m}\right) \delta(\sigma_i, \sigma_j)$ where A is the adjacency matrix, k_i is the (weighted) degree of node i, m is the total number of edges (or total edge weight), σ_i denotes the community of node i and $\delta(\sigma_i, \sigma_j) = 1$ if $\sigma_i = \sigma_j$ and 0 otherwise. For directed graphs a slightly different formulation is used, as proposed by Leicht and Newman : $Q = \sum_{ij} \left(A_{ij} - \gamma \frac{k_i^{\text{out}} k_j^{\text{in}}}{m}\right) \delta(\sigma_i, \sigma_j)$, where k_i^{out} and k_i^{in} refers to respectively the outdegree and indegree of node i , and A_{ij} refers to an edge from i to j. Note that this is the same of Leiden algorithm when setting $\gamma = 1$ and normalising by $2m$, or m for directed graphs. The method was introduced in Reichardt and Bornholdt (2006) and Leicht and Newman (2008). This method is not implemented internally but we use the external implementation[24].

Structural Clustering Algorithm for Networks (SCAN) : is an algorithm which detects clusters, hubs and outliers in networks (Xu et al. 2007). It clusters vertices based on a structural similarity measure. The method uses the neighborhood of the vertices as clustering criteria instead of only their direct connections. Vertices are grouped into the clusters by how they share neighbors.

Significance communities : it is a Leiden model where the quality function to optimize is: $Q = \sum_c \binom{n_c}{2} D(p_c \parallel p)$ where n_c is the number of nodes in community c, $p_c = \frac{m_c}{\binom{n_c}{2}}$, is the density of community c, $p = \frac{m}{\binom{n}{2}}$ is the overall density of the graph, and finally $D(x \parallel y) = x \ln \frac{x}{y} + (1 - x) \ln \frac{1-x}{1-y}$ is the binary Kullback-Leibler divergence. For directed graphs simply multiply the binomials by 2. The expected Significance in Erdos-Renyi graphs behaves roughly as $\frac{1}{2} n \ln n$ for both directed and undirected graphs in this formulation. It was introduced in Traag et al. (2013).

Spinglass : this method relies on an analogy between a very popular statistical mechanic model called Potts spin glass, and the community structure (Reichardt and Bornholdt 2006). It applies the simulated annealing optimization technique on this model to optimize the modularity.

SBM inference : this method is based on the inference of a stochastic block model. It corresponds to the version using the minimum description length principle to automatically discover the number of communities, introduced in Peixoto (2014a). It can fit both degree-corrected or non-degree-corrected SBM.

[23] https://github.com/vtraag/leidenalg
[24] https://github.com/vtraag/leidenalg

SBM inference with nested block models : this method is based on the inference of a nested stochastic block model. The goal of the nested block model is to avoid a resolution limit problem forbidding to discover small communities in large networks. This method was introduced in Peixoto (2014b). It can fit both degree-corrected or non-degree-corrected SBM.

Surprise Communities : it is a Leiden model where the quality function to optimize is: $Q = mD(q \parallel \langle q \rangle)$ where m is the number of edges, $q = \frac{\sum_c m_c}{m}$, is the fraction of internal edges, $\langle q \rangle = \frac{\sum_c \binom{n_c}{2}}{\binom{n}{2}}$ is the expected fraction of internal edges, and finally $D(x \parallel y) = x \ln \frac{x}{y} + (1-x) \ln \frac{1-x}{1-y}$ is the binary Kullback-Leibler divergence. For directed graphs, we can multiply the binomials by 2, and this leaves $\langle q \rangle$ unchanged, so that we can simply use the same formulation. For weighted graphs, we can simply count the total internal weight instead of the total number of edges for q , while $\langle q \rangle$ remains unchanged. It was introduced in Traag et al. (2015).

Walktrap : it is an approach based on random walks (Pons and Latapy 2005). The general idea is that if you perform random walks on the graph, then the walks are more likely to stay within the same community because there are only a few edges that lead outside a given community. Walktrap runs short random walks and uses the results of these random walks to merge separate communities in a bottom-up manner.

Overlapping Communities

Angel : it is a node-centric bottom-up community discovery algorithm. It leverages ego-network structures and overlapping label propagation to identify micro-scale communities that are subsequently merged in mesoscale ones. Angel is the, faster, successor of Demon.

BigClam : it is an overlapping community detection method that scales to large networks. The model was introduced in Yang and Leskovec (2013) and it has three main ingredients: 1) The node community memberships are represented with a bipartite affiliation network that links nodes of the social network to communities that they belong to. 2) People tend to be involved in communities to various degrees. Therefore, each affiliation edge in the bipartite affiliation network has a nonnegative weight. The higher the node's weight of the affiliation to the community the more likely is the node to be connected to other members in the community. 3)When people share multiple community affiliations, the links between them stem for one dominant reason. This means that for each community a pair of nodes shares we get an independent chance of connecting the nodes. Thus, naturally, the more communities a pair of nodes shares, the higher the probability of being connected.

Cluster-Overlap Newman Girvan Algorithm (CONGA) : is an algorithm for discovering overlapping communities (Gregory 2007). It extends the Girvan and Newman's algorithm with a specific method of deciding when and how to split vertices. The algorithm is as follows: 1. Calculate edge betweenness of all edges in the network. 2. Calculate vertex betweenness of vertices, from edge betweennesses. 3. Find the candidate set of vertices: those whose vertex betweenness is greater than the maximum edge betweenness.

4. If candidate set is non-empty, calculate pair betweennesses of candidate vertices, and then calculate split betweenness of candidate vertices. 5. Remove edge with maximum edge betweenness or split vertex with maximum split betweenness (if greater). 6. Recalculate edge betweenness for all remaining edges in same component(s) as removed edge or split vertex. 7. Repeat from step 2 until no edges remain.

Cluster-Overlap Newman Girvan Algorithm Optimized (CONGO) : it is an optimization of the CONGA algortithm (Gregory 2008). The CONGO algorithm is the same as CONGA but using local betweenness. The complete CONGO algorithm is as follows: 1. Calculate edge betweenness of edges and split betweenness of vertices. 2. Find edge with maximum edge betweenness or vertex with maximum split betweenness, if greater. 3. Recalculate edge betweenness and split betweenness: a) Subtract betweenness of h-region centered on the removed edge or split vertex. b) Remove the edge or split the vertex. c) Add betweenness for the same region. 4. Repeat from step 2 until no edges remain.

Demon : it is a node-centric bottom-up overlapping community discovery algorithm (Coscia et al. 2012; 2014). It leverages ego-network structures and overlapping label propagation to identify micro-scale communities that are subsequently merged in mesoscale ones.

Ego-networks : this method returns overlapping communities centered at each node within a given radius.

Kclique : this method finds k-clique communities in graph using the percolation method (Palla et al. 2005). A k-clique community is the union of all cliques of size k that can be reached through adjacent (sharing k-1 nodes) k-cliques.

Link Aggregate Algorithm and Iterative Scan Algorithm (LAIS2) : it is an overlapping community discovery algorithm based on the density function (Baumes et al. 2005). In the algorithm considers the density of a group is defined as the average density of the communication exchanges between the actors of the group. LAIS2 is composed of two procedures LA (Link Aggregate Algorithm) and IS2 (Iterative Scan Algorithm).

Lemon : it is a large scale overlapping community detection method based on local expansion via a minimum one norm (Li et al. 2015). The algorithm adopts a local expansion method in order to identify the community members from a few exemplary seed members. The algorithm finds the community by seeking a sparse vector in the span of the local spectra such that the seeds are in its support. Lemon can achieve the highest detection accuracy among state-of-the-art proposals. The running time depends on the size of the community rather than that of the entire graph.

Local optimization Funtion Model (LFM) : it is based on the local optimization of a fitness function (Lancichinetti et al. 2009). It finds both overlapping communities and the hierarchical structure.

MultiCom : it is an algorithm for detecting multiple local communities, possibly overlapping, by expanding the initial seed set (Hollocou et al. 2017). This algorithm uses

local scoring metrics to define an embedding of the graph around the seed set. Based on this embedding, it picks new seeds in the neighborhood of the original seed set, and uses these new seeds to recover multiple communities.

Node perception : it is based on the idea of joining together small sets of nodes (Soundarajan and Hopcroft 2015). The algorithm first identifies sub-communities corresponding to each node's perception of the network around it. To perform this step, it considers each node individually, and partition that node's neighbors into communities using some existing community detection method. Next, it creates a new network in which every node corresponds to a sub-community, and two nodes are linked if their associated sub-communities overlap by at least some threshold amount. Finally, the algorithm identifies overlapping communities in this new network, and for every such community, merge together the associated sub-communities to identify communities in the original network.

Overlapping Seed Set Expansion (OSSE) : this is an overlapping community detection algorithm optimizing the conductance community score (Whang et al. 2013). The algorithm uses a seed set expansion approach; the key idea is to find good seeds, and then expand these seed sets using the personalized PageRank clustering procedure.

Speaker-listener Label Propagation Algorithm (SLPA) : it is an overlapping community discovery that extends the LPA (Xie et al. 2011). SLPA consists of the following three stages: 1) the initialization 2) the evolution 3) the post-processing

Fuzzy Communities

Fuzzy-Rough Community Detection on Fuzzy Granular model of Social Network (FRC-FGSN) : the method assigns nodes to communities specifying the probability of each association (Kundu and Pal 2015). The flattened partition ensures that each node is associated with the community that maximizes such association probability. FRC-FGSN may generate orphan nodes (i.e., nodes not assigned to any community).

Appendix C: Edge partition.

Hierarchical Link Clustering (HLC) : it is a method to classify links into topologically related groups (Ahn et al. 2010). The algorithm uses similarity between links to build a dendrogram where each leaf is a link from the original network and branches represent link communities. At each level of the link dendrogram is calculated the partition density function, based on link density inside communities, to pick the best level to cut.

Markov CLustering (MCL) : this algorithm is based on a simulation of (stochastic) flow in graphs (Enright et al. 2002). The MCL algorithm finds cluster structure in graphs by a mathematical bootstrapping procedure. The process deterministically computes (the probabilities of) random walks through the graph, and uses two operators transforming one set of probabilities into another. It does so using the language of stochastic matrices (also called Markov matrices) which capture the mathematical concept of random walks on a graph. The MCL algorithm simulates random walks within a graph by the alternation of two operators called expansion and inflation.

Abbreviations
AGDL: Agglomerative clustering on a directed graph; AGMFit: Affiliation graph model fitting; CD: Community discovery; CDlib: Community discovery library; CDTB: Community detection toolBox; Circulo: A community detection evaluation framework; CONGA: Cluster-overlap Newman Girvan algorithm; CONGO: Cluster-overlap Newman Girvan algorithm optimized; CPM: Constant Potts model; DER: Diffusion entropy reducer; EM: Expectation maximization; FRC-FGSN: Fuzzy-rough community detection on fuzzy granular model of social network; HLC: Hierarchical link clustering; LAIS2: Link aggregate algorithm and iterative scan algorithm; LFM: Local optimization funtion model; LPA: Label propagation algorithm; MCL: Markov cLustering; NDlib: Network diffusion library; OSSE: Overlapping seed set expansion; SCAN: Structural clustering algorithm for networks; SLPA: Speaker-listener label propagation algorithm; SNAP: Stanford network analysis platform

Acknowledgements
This work is supported by the European Community's H2020 Program under the scheme "INFRAIA-1-2014-2015: Research Infrastructures", grant agreement #654024 "SoBigData: Social Mining & Big Data Ecosystem"[25].

Authors' contributions
GR planned the library Design; LM and GR wrote the documentation; all authors performed community models integration. All authors wrote the manuscript, reviewed and approved the manuscript.

Funding
This work is supported by the European Community's H2020 Program under the scheme "INFRAIA-1-2014-2015: Research Infrastructures", grant agreement #654024 "SoBigData: Social Mining & Big Data Ecosystem"[26].

Availability of data and materials
All the described Community Discovery algorithms, have been implemented and made available within the python library CDlib and can be found at the addess https://github.com/GiulioRossetti/cdlib. Not applicable for data.

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

Author details
[1]KDD Lab. ISTI-CNR, via G. Moruzzi, 1, Pisa, Italy. [2]University of Pisa, Largo Bruno Pontecorvo,2 , Pisa, Italy. [3] Université de Lyon, Lyon, France.

Received: 17 April 2019 Accepted: 1 July 2019
Published online: 29 July 2019

References
Ahn Y.-Y., Bagrow JP, Lehmann S (2010) Link communities reveal multiscale complexity in networks. Nature 466(7307):761
Baumes J, Goldberg M, Magdon-Ismail M (2005) Efficient identification of overlapping communities. In: International Conference on Intelligence and Security Informatics. Springer. pp 27–36
Blondel VD, Guillaume J.-L., Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech Theory Exp 2008(10):10008
Cazabet R, Rossetti G, Amblard F (2018) Dynamic community detection. Encyclopedia of Social Network Analysis and Mining. Springer, New York. pp 1–10
Chen J, Saad Y (2012) Dense subgraph extraction with application to community detection. IEEE Trans Knowl Data Eng 24(7):1216–1230
Clauset A, Newman ME, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70(6):066111
Coscia M, Giannotti F, Pedreschi D (2011) A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining: The ASA Data Science Journal 4(5):512–546
Coscia M, Rossetti G, Giannotti F, Pedreschi D (2012) 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. pp 615–623
Coscia M, Rossetti G, Giannotti F, Pedreschi D (2014) Uncovering hierarchical and overlapping communities with a local-first approach. ACM Trans Knowl Discov Data (TKDD) 9(1):6
Dao V.-L., Bothorel C, Lenca P (2018) Estimating the similarity of community detection methods based on cluster size distribution. In: International Conference on Complex Networks and Their Applications. Springer. pp 183–194
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7(Jan):1–30
Enright AJ, Van Dongen S, Ouzounis CA (2002) An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res 30(7):1575–1584
Erdös P, Rényi A (1959) On random graphs i. Publ Math Debr 6:290
Flake GW, Lawrence S, Giles CL, et al. (2000) Efficient identification of web communities. In: KDD Vol. 2000. pp 150–160
Fortunato S (2010) Community detection in graphs. Phys Rep 486(3-5):75–174
Fortunato S, Hric D (2016) Community detection in networks: A user guide. Phys Rep 659:1–44
Girvan M, Newman ME (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826
Gregory S (2007) An algorithm to find overlapping community structure in networks. In: European Conference on Principles of Data Mining and Knowledge Discovery. Springer. pp 91–102

[25]SoBigData:http://www.sobigdata.eu
[26]SoBigData:http://www.sobigdata.eu

Gregory S (2008) A fast algorithm to find overlapping communities in networks. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer. pp 408–423
Harenberg S, Bello G, Gjeltema L, Ranshous S, Harlalka J, Seay R, Padmanabhan K, Samatova N (2014) Community detection in large-scale networks: a survey and empirical evaluation. Wiley Interdiscip Rev Comput Stat 6(6):426–439
Hollocou A, Bonald T, Lelarge M (2017) Multiple local community detection. SIGMETRICS Perform Eval Rev 45(3):76–83. https://doi.org/10.1145/3199524.3199537
Hubert L, Arabie P (1985) Comparing partitions. J Classif 2(1):193–218
Jebabli M, Cherifi H, Cherifi C, Hamouda A (2018) Community detection algorithm evaluation with ground-truth data. Phys A Stat Mech Appl 492:651–706
Kozdoba M, Mannor S (2015) Community detection via measure space embedding. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. pp 2890–2898
Kundu S, Pal SK (2015) Fuzzy-rough community in social networks. Pattern Recognit Lett 67:145–152
Lancichinetti A, Fortunato S (2009) Community detection algorithms: a comparative analysis. Phys Rev E 80(5):056117
Lancichinetti A, Fortunato S, Kertész J (2009) Detecting the overlapping and hierarchical community structure in complex networks. New J Phys 11(3):033015
Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110
Leicht EA, Newman ME (2008) Community structure in directed networks. Phys Rev Lett 100(11):118703
Leskovec J, Lang KJ, Mahoney M (2010) Empirical comparison of algorithms for network community detection. In: Proceedings of the 19th International Conference on World Wide Web. ACM. pp 631–640
Li Y, He K, Bindel D, Hopcroft JE (2015) Uncovering the small community structure in large networks: A local spectral approach. In: Proceedings of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee. pp 658–668
Li Z, Wang R.-S., Zhang S, Zhang X.-S. (2016) Quantitative function and algorithm for community detection in bipartite networks. Inf Sci 367:874–889
Malliaros FD, Vazirgiannis M (2013) Clustering and community detection in directed networks: A survey. Phys Rep 533(4):95–142
McDaid AF, Greene D, Hurley N (2011) Normalized mutual information to evaluate overlapping community finding algorithms. arXiv preprint. arXiv:1110.2515
Meilă M (2007) Comparing clusterings—an information based distance. J Multivar Anal 98(5):873–895
Miyauchi A, Kawase Y (2016) Z-score-based modularity for community detection in networks. PloS One 11(1):0147805
Murray G, Carenini G, Ng R (2012) Using the omega index for evaluating abstractive community detection. In: Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization. Association for Computational Linguistics. pp 10–18
Newman ME (2006) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74(3):036104
Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113
Newman ME, Leicht EA (2007) Mixture models and exploratory analysis in networks. Proc Natl Acad Sci 104(23):9564–9569
Nicosia V, Mangioni G, Carchiolo V, Malgeri M (2009) Extending the definition of modularity to directed graphs with overlapping communities. J Stat Mech Theory Exp 2009(03):03024
Orman GK, Labatut V, Cherifi H (2012) Comparative evaluation of community detection algorithms: a topological approach. J Stat Mech Theory Exp 2012(08):08001
Palla G, Derényi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814
Parés F, Gasulla DG, Vilalta A, Moreno J, Ayguadé E, Labarta J, Cortés U, Suzumura T (2017) Fluid communities: A competitive, scalable and diverse community detection algorithm. In: International Workshop on Complex Networks and Their Applications. Springer. pp 229–240
Peixoto TP (2014) Efficient monte carlo and greedy heuristic for the inference of stochastic block models. Phys Rev E 89(1):012804
Peixoto TP (2014) Hierarchical block structures and high-resolution model selection in large networks. Phys Rev X 4(1):011047
Pons P, Latapy M (2005) Computing communities in large networks using random walks. In: International Symposium on Computer and Information Sciences. Springer. pp 284–293
Radicchi F, Castellano C, Cecconi F, Loreto V, Parisi D (2004) Defining and identifying communities in networks. Proc Natl Acad Sci 101(9):2658–2663
Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106
Reichardt J, Bornholdt S (2006) Statistical mechanics of community detection. Phys Rev E 74(1):016110
Rossetti G (2017) Rdyn: graph benchmark handling community dynamics. J Compl Netw 5(6):893–912
Rossetti G, Cazabet R (2018) Community discovery in dynamic networks: A survey. ACM Comput Surv (CSUR) 51(2):35
Rossetti G, Milli L, Rinzivillo S, Sîrbu A, Pedreschi D, Giannotti F (2018) Ndlib: a python library to model and analyze diffusion processes over complex networks. Int J Data Sci Analytics 5(1):61–79
Rossetti G, Pappalardo L, Rinzivillo S (2016) A novel approach to evaluate community detection algorithms on ground truth. In: Complex Networks VII: Proceedings of the 7th Workshop on Complex Networks CompleNet 2016. Springer International Publishing. pp 133–144
Rossetti G, Pedreschi D, Giannotti F (2017) Node-centric community discovery: From static to dynamic social network analysis. Online Soc Netw Media 3:32–48
Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105(4):1118–1123
Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22:888–905
Soundarajan S, Hopcroft JE (2015) Use of local group information to identify communities in networks. ACM Trans Knowl Discov Data (TKDD) 9(3):21

Traag VA, Aldecoa R, Delvenne J.-C. (2015) Detecting communities using asymptotical surprise. Phys Rev E 92(2):022816
Traag VA, Krings G, Van Dooren P (2013) Significant scales in community structure. Sci Rep 3:2930
Traag VA, Van Dooren P, Nesterov Y (2011) Narrow scope for resolution-limit-free community detection. Phys Rev E 84(1):016114
Traag V, Waltman L, van Eck NJ (2018) From louvain to leiden: guaranteeing well-connected communities. arXiv preprint. arXiv:1810.08473
Vinh NX, Epps J, Bailey J (2010) Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. J Mach Learn Res 11(Oct):2837–2854
Whang JJ, Gleich DF, Dhillon IS (2013) Overlapping community detection using seed set expansion. In: Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management. ACM. pp 2099–2108
Xie J, Kelley S, Szymanski BK (2013) Overlapping community detection in networks: The state-of-the-art and comparative study. ACM Comput Surv (csur) 45(4):43
Xie J, Szymanski BK, Liu X (2011) Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference On. IEEE. pp 344–349
Xu X, Yuruk N, Feng Z, Schweiger TA (2007) Scan: a structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. pp 824–833
Yang J, Leskovec J (2013) Overlapping community detection at scale: a nonnegative matrix factorization approach. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. ACM. pp 587–596
Yang J, Leskovec J (2015) Defining and evaluating network communities based on ground-truth. Knowl Inf Syst 42(1):181–213
Zachary WW (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33(4):452–473
Zhang W, Wang X, Zhao D, Tang X (2012) Graph degree linkage: Agglomerative clustering on a directed graph. In: European Conference on Computer Vision. Springer. pp 428–441

Publisher's Note