OPEN ACCESS

Journal of Statistical Mechanics: Theory and Experiment
An IOP and SISSA journal

REVIEW

Fast unfolding of communities in large networks: 15 years later

Vincent Blondel[1], **Jean-Loup Guillaume**[2] **and Renaud Lambiotte**[3,4,*]

[1] Université catholique de Louvain, Louvain-la-Neuve, Belgium
[2] L3i, La Rochelle Université, La Rochelle, France
[3] University of Oxford, Oxford, United Kingdom
[4] Turing Institute, London, United Kingdom
E-mail: renaud.lambiotte@maths.ox.ac.uk, vincent.blondel@uclouvain.be and jean-loup.guillaume@univ-lr.fr

Received 13 November 2023
Accepted for publication 21 June 2024
Published 28 October 2024

Online at stacks.iop.org/JSTAT/2024/10R001
https://doi.org/10.1088/1742-5468/ad6139

Abstract. The Louvain method was proposed 15 years ago as a heuristic method for the fast detection of communities in large networks. During this period, it has emerged as one of the most popular methods for community detection: the task of partitioning vertices of a network into dense groups, usually called communities or clusters. Here, after a short introduction to the method, we give an overview of the different generalizations, modifications and improvements that have been proposed in the literature, and also survey the quality functions, beyond modularity, for which it has been implemented. Finally, we conclude with a discussion on the limitations of the method and perspectives for future research.

Keywords: networks, community detection, optimization

*Author to whom any correspondence should be addressed.

Contents

1. Introduction 2
2. Basics on networks and modularity 3
3. The Louvain method 7
4. Louvain++, a selection of enhancements 9
4.1. Modification of the algorithm 11
4.2. Modification of the quality function 13
4.3. Louvain for non-simple graphs 15
4.4. Distributed/parallel/GPU solutions 16
5. Perspectives 19
Acknowledgments 20
References 20

1. Introduction

Networks provide a powerful language to model and analyze interacting systems, in a broad range of disciplines [1]. Facebook and food-webs, the brain and the job market are all systems where connectivity is critical, and where elements—here, users, species, neurons and workers—are represented by vertices, while pairwise interactions—here, friendship, predation, synchronization and job mobility—are represented by edges. Over the last two decades, several tools have been developed to extract information from the myriads of interactions composing large networks. Among these, community detection plays a central role. Community detection aims to uncover the community structure of a network: that is, its organization into groups, where vertices in the same group are strongly connected with each other, and weakly connected with vertices outside the group [2, 3]. Community detection is usually considered as an unsupervised clustering problem where the purpose is to find the best—according to a given definition—division into groups together with the most appropriate number of groups. Community detection is a rich and challenging problem that has been considered from a variety of perspectives and principles [4] and finds many applications, from network visualization [5] to vertex classification [6]. In its most popular form, the communities are assumed to form a partition of the network, so that each vertex belongs to one single community. Partitioning methods have the advantage that their outcome is easier to interpret than that of overlapping methods, but also involve less parameters to estimate and have sound mathematical foundations. Even within the family of partitioning methods, several complementary approaches have been developed, leading to different functions to measure the quality of a partition defined in terms of, e.g. the density of links inside

J. Stat. Mech. (2024) 10R001

communities [7], the number of links between communities [8], the confinement of random walkers inside communities [9, 10] or its likelihood of having generated the observed network [11].

A common challenge across methods is the need for efficient algorithms to optimize their respective quality functions: that is, finding the partition of the network with an optimal value. This problem is not new and finds its root in graph partitioning, where different algorithms have been proposed since the 1970s. It is more recent, arguably since the seminal works of Newman [7], that the research community started focusing on the related problem of community detection. It rapidly gained popularity due to the increasing access to large network data, whose analysis required efficient clustering methods. The most popular quality function is the so-called Newman–Girvan modularity, and community detection is often considered—in practice—synonymous to modularity optimization, despite its limitations and the availability of alternative approaches. Until 2007, the most efficient methods for community detection were either spectral, i.e. defining modules based on dominant eigenvectors of matrices associated with the graph, or greedy, with the early works of Clauset *et al* [12] and Wakita and Tsurumi [13]. However, the former are, by construction, limited to networks of the order of 10^4 vertices, orders of magnitude smaller than large-scale social and information networks; whereas the proposed greedy methods did not perform sufficiently well in terms of accuracy, due to the ruggedness of the optimization landscape.

On 4 March 2008, we posted on the arXiv an article entitled 'Fast unfolding of communities in large networks', proposing a greedy method for modularity optimization, and made available an efficient C++ code. A workshop was organized a few days later, on March 8, entitled 'Detection and visualization of communities in large complex networks', where we invited the leading researchers in the field to our research group in Louvain-la-Neuve. The article was based on Etienne Lefebvre's master's thesis and is a fine example of serendipity. His work concerned the generation of random graphs with community structure, and it was by reversing the community generation process that he obtained the foundations of what would become the Louvain method. After a desk rejection from Nature and a swift rejection from PNAS, the paper was submitted to JSTAT, accepted on September 3 and published on October 9 [14]. Fifteen years later, the paper has been cited just over 20 000 times on Google Scholar; this is one new citation every 6 h for more than 15 years. The method had no name in the original article, and was, for a time, named BGLL after the authors' initials. The term 'Louvain method' appears for the first time in an article published in September 2009 [15]. The Louvain method, implemented in standard libraries for graph mining (see table 1), is at the core of the network science ecosystem, with applications in several fields of science (see figure 1). The purpose of this perspective paper is take a step back and to consider the legacy of the paper, by giving an overview of the research surrounding it, from improvements of the original method to its applications to other quality functions.

2. Basics on networks and modularity

A network is represented mathematically by a graph $\mathcal{G} = (V, E)$, made of a set of vertices V of cardinality $n \equiv |V|$, and a set of edges $E = \{\{i, j\} \mid i, j \in V\}$ of cardinality $m \equiv |E|$.

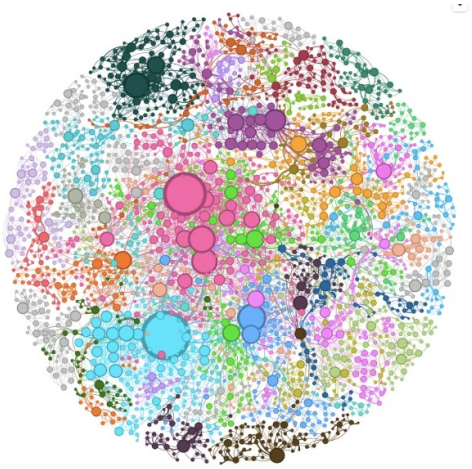
J. Stat. Mech. (2024) 10R001

Table 1. A list of implementations of the Louvain method.

Libraries		
Language	Link	Objective function
Python	https://networkx.org/	mutiresolution Q, directed
Python	https://networkit.github.io/	mutiresolution Q, parallel
Python	https://github.com/vtraag/leidenalg	several functions, Leiden
Python	https://github.com/barahona-research-group/PyGenStability	mutiresolution Q
Julia	https://github.com/afternone/CommunityDetection.jl	several functions
Javascript	https://graphology.github.io/	mutiresolution Q
Matlab	https://github.com/GenLouvain/GenLouvain	several functions, multilayer
Java	https://github.com/jcnguyen/research-2015	several functions
Multiple	https://igraph.org/	mutiresolution Q
Software		
Name	Objective	Link
Cytoscape (plugins)	Analysis and visualisation	https://cytoscape.org/
Gephi	Analysis and visualisation	https://gephi.org/
Tulip	Analysis and visualisation	https://tulip.labri.fr/site/
Pajek	Analysis and visualisation	http://mrvar.fdv.uni-lj.si/pajek/
SAS Viya	Analytics	www.sas.com/en_us/home.html
Neo4j	Database	https://neo4j.com/fr/
Memgraph	Database	https://memgraph.com/

Without loss of generality, we will identify the vertex set V with the set $\{0,\dots,n-1\}$. In this introductory section, we will assume, for simplicity, that the network is undirected (so that edges do not have a direction) and unweighted (so that the graph only encodes the presence or absence of an edge between two vertices). Note that the concepts described in this section and, in particular, expressions for modularity, have been generalized to more general settings, as we discuss in detail in section 4.3. A useful way to encode and investigate the structure of a graph is its adjacency matrix. The adjacency matrix $\boldsymbol{A}$ is an $n \times n$ matrix encoding the presence or absence of an edge between any pair of vertices in the graph. Its entries A_{ij} are equal to 1 if i is adjacent to j, and 0 otherwise. For undirected graphs we have $A_{ij} = A_{ji}$, i.e. the adjacency matrix is symmetric ($\boldsymbol{A} = \boldsymbol{A}^\top$). Several properties of a graph can be readily calculated from the adjacency matrix, including the degree of each vertex i as $k_i = \sum_j A_{ij}$, but above all, this representation makes it possible to exploit concepts and tools from linear algebra.

J. Stat. Mech. (2024) 10R001



Lecture Notes in Computer Science	428
PLoS ONE	263
Scientific Reports	229
Physica A	215
ACM International Conference Proceeding Series	138
Studies in Computational Intelligence	135
IEEE Access	127
Physical Review E	114
Nature Communications	110
NeuroImage	104
Scientometrics	100
Applied Network Science	98
Social Network Analysis and Mining	96
Communications in Computer and Information Science	78
CEUR Workshop Proceedings	69
Sustainability	65
Expert Systems with Applications	60
PLoS Computational Biology	57
Information Sciences	54
PNAS	50

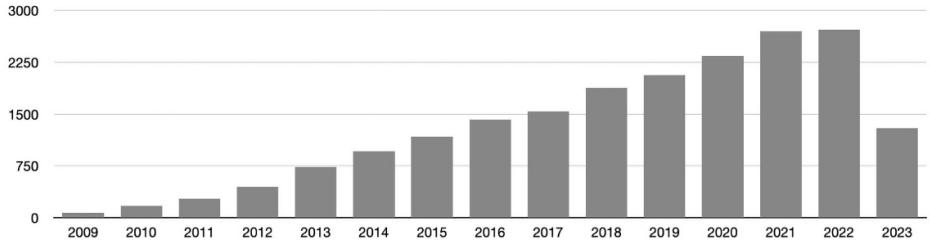


Figure 1. (Top left) The collaboration of researchers having cited the Louvain method. All articles citing [14] have been downloaded from Scopus on 6 April 2023. A bipartite network has been constructed relating author IDs, as identified by Scopus, and papers. The collaboration network was then constructed by first projecting the network according to [16], removing edges with a weight smaller than 0.2 and keeping the largest connected component. The resulting network is composed of 2490 vertices. Running the Louvain method identifies 39 communities, with a modularity of 0.926. The size of the vertices is proportional to their weighted degree. (Top right) Originally published in a Physics journal, the Louvain method has found applications in a variety of scientific domains, as shown by the list of the top 20 journals citing [14]. Further down in the list, one can find journals as diverse as Poetics (nine articles), Journal of Transport Geography (eight articles) and Nature Immunology (five articles). Across those papers, the Louvain method has been mentioned for a variety of purposes, e.g. as a general reference to the field of community detection, or due to its use on specific datasets, or as a baseline for other clustering methods. Several works also proposed generalizations or modifications to the original method, as we will present in detail in this article. (Bottom) A histogram of the total number of citations per year according to Google Scholar (20 054 citations in total on 6 July 2023).

Since the seminal works of Watts and Strogatz [17], it has been recognized that real-world networks exhibit a mix of random and non-random patterns. Important types of structures, found in a wide range of empirical data, include clustering, here, understood in terms of the density of triangles, fat-tailed degree distribution, power-law or not [18],

J. Stat. Mech. (2024) 10R001

and the presence of communities in a network [19], that is, the presence of dense clusters that are weakly connected with one another. Community detection is an algorithmic problem that aims to find the community structure of a network. As such, it can be seen as an unsupervised clustering problem, and some methods actually consist of first embedding the vertices in a space, before applying classical clustering methods, such as k-means [20]. In this section, we describe one of the most popular methods for community detection, despite some limitations described below, based on the notion of Newman–Girvan modularity [7], modularity in short.

Modularity is a measure of the quality of the partition of a network into groups. Let us first consider a group of vertices defined by a set α. The intuitive idea behind modularity is to compare the number of links inside α with an expectation of this number under a random null model. The choice of null model is, in principle, open and should depend on the mechanisms that constrain the formation of edges [21]. The most popular choice is, by far, the soft configuration or Chung–Lu model [22], a random graph model fixing, on expectation, the degrees of each vertex. For this choice, the expected number of edges between two vertices i and j is given by $\frac{k_i k_j}{2m}$, and the contribution of community α to modularity is given by

$$\sum_{i,j\in\alpha}\left(A_{ij}-\frac{k_i k_j}{2m}\right). \tag{1}$$

Note that each term in the sum may either be positive or negative depending on the presence or absence of a link between two vertices. Intuitively, a good, dense community will be one with a high value for this expression. Note also that the null model penalizes more pairs of vertices with a higher degree, which tends to favor configurations where high degree vertices are placed in different communities [10].

The Newman–Girvan modularity does not focus on one single community, but on the partition of the vertices into C communities $\mathcal{P}=\{\alpha_1,\alpha_2,\ldots,\alpha_C\}$. The quality of a partition is obtained from the sum of equation (1) over the communities, leading to the expression

$$Q=\frac{1}{2m}\sum_{\alpha\in\mathcal{P}}\sum_{i,j\in\alpha}\left(A_{ij}-\frac{k_i k_j}{2m}\right), \tag{2}$$

where the additional prefactor $1/(2m)$ ensures that modularity is in the interval $[-1,1]$. Note that modularity can also be interpreted in terms of the assortativity coefficient, a correlation measure of vertex attributes, here, the community labels, at the endpoints of edges in the network [23], in terms of a covariance defined on the graph [24] and in terms of the clustered covariance matrix from the random-walk exploration of the graph [25]. Intuitively, modularity gives a high value to a partition if its communities concentrate an unexpectedly large number of edges inside its communities. In contrast with classical concepts such as the cut size, modularity allows one to compare partitions made of different numbers of communities, and the partition maximizing modularity is usually non-trivial, i.e. neither made of one large community or of n singletons. Modularity has become an essential element of several community detection methods. Modularity

J. Stat. Mech. (2024) 10R001

optimization methods aim to optimize modularity, that is, find the 'best' partition of a network with the highest modularity value. As modularity optimization is NP-hard [26], several heuristics have been proposed for modularity optimization. Popular techniques include spectral methods [27] that exploit the information provided by the dominant eigenvectors of the so-called modularity matrix; however, large networks require faster greedy methods that proceed by agglomerating groups of vertices into larger ones [2].

Before going further, note that the contribution of vertex i, assigned to a certain modularity α, to the modularity of a partition is given by

$$Q_i = \frac{1}{2m}\left(\sum_{j\in\alpha} A_{ij} - \frac{k_i\, k_\alpha}{2m}\right), \tag{3}$$

where k_α is the degree of community α, defined as the sum of the degrees of its vertices. This expression shows that the contribution of vertex i requires only local information, its number of neighbors inside the community, its degree and the degrees inside the community. Locality is essential to design an efficient greedy algorithm for modularity optimization.

Modularity, as a quality function, suffers from some limitations, which are thus inherited by any modularity optimization algorithm. Among those, let us note its tendency to over-fit [28], as it finds communities even in the case of random graphs [29], and to have, implicitly, a characteristic scale, making it search for communities with a size that might not be compatible with the underlying structure [30]. The optimization of modularity is also tricky in practical contexts, as its landscape may be rugged and exhibit several competing local maxima, associated with very different partitions [31]. Generalizations of modularity have been proposed to alleviate these limitations, for instance, by incorporating a resolution parameter allowing one to tune the characteristic size of the communities [25, 32, 33]. Note that this solution opens up a new problem: that of determining the right value of the resolution parameter, which can left as a choice for the user, or identified based on other principles, e.g. based on the robustness of the partitions [34].

3. The Louvain method

The Louvain method is a greedy optimization heuristic originally proposed to optimize modularity; however, its flexibility allows one to optimize other quality functions as well, as described below. The approach consists of two phases that are iteratively repeated, until a maximum of modularity is obtained. The first phase consists of moving vertices from their community to that of their neighbors until a local maximum is reached. The second phase then consists of forming a new graph, whose vertices are obtained by aggregating the vertices attached to the same community in the previous phase. A combination of the two phases is called a 'pass'. Starting from a network composed

J. Stat. Mech. (2024) 10R001

Algorithm 1. Single VertexMover.

Require: $G = (V, E, w)$ a weighted graph
Require: $\mathcal{P}$ a partition of V ($\mathcal{P}[i]$ = community of vertex i)
Ensure: ΔQ, the gain of modularity

begin
  $c_{\text{old}} \leftarrow \mathcal{P}[i]$
  REMOVE(i, c_{old})
  $\mathcal{C} \leftarrow \{\mathcal{P}[j] \mid (i,j) \in E\} \cup \{c_{\text{old}}\}$
  $c_{\text{new}} \leftarrow \arg\max_{c \in \mathcal{C}} \{\text{GAIN}(i, c)\}$
  if GAIN(i, c_{new}) − LOSS$(i, c_{\text{old}}) > 0$ **then**
    INSERT(i, c_{new})
    return GAIN(i, c_{new}) − LOSS(i, c_{old})
  else
    INSERT(i, c_{old})
    return 0

of n vertices, the number of vertices decreases at each pass, until no improvement in modularity can be obtained.

First phase: vertex mover (VM). The first phase begins with an undirected weighted graph with C vertices to which an index between 0 and $C-1$ has been randomly assigned. Here, C is the number of vertices in the first pass, and the number of communities found in previous passes otherwise. One starts by placing each vertex into its own community: that is, we start with a partition made of C communities. We then consider the first vertex, i.e. with index 0, and evaluate the change of modularity by removing 0 from its community and placing it in the community of each of its neighbors. Importantly, for the speed of the method, this step only requires local information—see equation (3)—and its number of operations is proportional to the degree of the vertex. Vertex 0 is then assigned to the community where the increase is maximum, if this maximum increase is positive. Otherwise, vertex 0 is left in its original community—see algorithm 1 (all algorithms are adapted from [35]). This step is repeated sequentially with all the vertices based on their index. When reaching vertex $C-1$, the process restarts at vertex 0, and the iteration continues until no vertex is moved during a complete iteration over the C vertices—see algorithm 2. The phase ends in a local maximum, where greedy moves of the vertices between communities do not allow an increase in the modularity. At this moment, a new partition of the vertices into C' communities has been produced. If $C' \neq C$, one proceeds to the next phase, and updates the value of the number of communities. Otherwise, the algorithm is finished and returns the partition as its estimate for the optimum of modularity.

Second phase: aggregation. The second phase consists of building a new weighted graph, whose vertices are the C communities found during the first phase. The weight of the link between two communities is given by the sum of the weights of the links

Algorithm 2. VertexMoverIteration.

Require: $G = (V, E, w)$ a weighted graph
Require: ϵ a stopping criterion
Ensure: a partition $\mathcal{P}$ of V

begin
  INIT($\mathcal{P}$)
  do
    $increase \leftarrow 0$
    forall vertices i **do**
      $increase \leftarrow increase$+SingleVertexMover(i)
  while $increase > \epsilon$
  return $\mathcal{P}$

between the vertices of these two communities. Moreover, links between the vertices of a same community lead to self-loops in the aggregated network. This operation is performed to ensure that the modularity of a partition in the aggregated graph has the same value as that of the same partition on the original graph. Once the second phase is finished, the first phase is reapplied on the aggregated network.

The output of the algorithm is the partition uncovered in the last pass, as it is the best partition found in terms of modularity—see figure 2 and algorithm 3. The method can also provide intermediary steps: that is, the set of partitions of each pass, which provides a hierarchical representation of the network, each community at a pass being composed of a combination of combinations at the previous pass. Since its inception, the method has consistently been shown to provide a good balance between accuracy and the speed of execution, and to have a complexity close to linear in benchmarks and empirical networks. The speed of the method is ensured by the locality of each operation, as vertices only move to neighboring communities. Its accuracy arises from its flexibility, as vertices may be removed from their community and re-assigned to others in the process, and its multi-scale nature, as the community exploration is first performed locally, and then over longer distances as the vertices are aggregated. Combined with its simplicity, these strengths have made the Louvain method a popular choice to optimize not only modularity, but also other quality functions, including the Map equation [9] and Markov stability [33], as we discuss more in detail below. When applied to modularity, as is usually the case, it is important to remember that the Louvain method inherits the limitations of modularity, including its resolution limit. It is also important to remember that it is a heuristic, which does not provides guarantees on the identification of the global optimum of modularity.

4. Louvain++, a selection of enhancements

Because of its effectiveness and simplicity, the Louvain method has been extensively studied, and numerous modifications have been proposed, whether to improve results,

J. Stat. Mech. (2024) 10R001

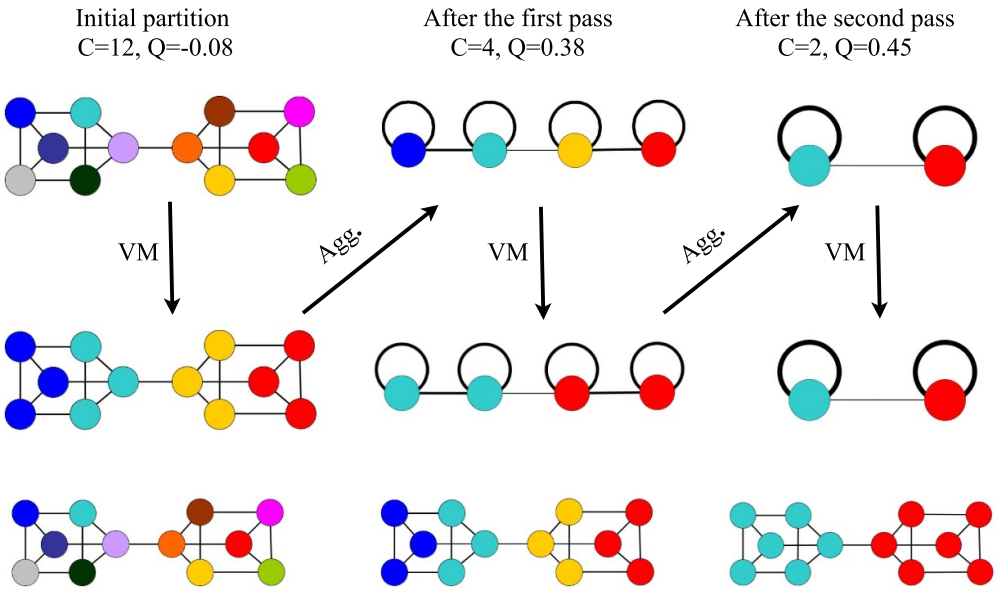


Figure 2. Each pass of the Louvain method is made of two phases. The first phase (VM) consists of sequentially moving vertices to neighboring communities, leading to a maximum increase in modularity. The second phase aggregates vertices and constructs a meta-graph, whose vertices are the communities found after the first phase. The two phases are repeated until no improvement in modularity is observed. In this illustration, the algorithm converges after two passes, uncovering an optimal partition made of two communities, and a value of $Q = 0.45$. In this example, it is easy to show that further aggregating the two communities into one big community would result in a decline in modularity, as $Q = 0$ in that case. [36] John Wiley & Sons. Copyright © ISTE Ltd 2011.

Algorithm 3. The Louvain algorithm.

Require: $G = (V, E, w)$ a weighted graph
Ensure: a partition $\mathcal{P}$ of V

begin
  repeat
    $\mathcal{P} \leftarrow$VertexMoverIteration(G)
    $G \leftarrow$PartitionToGraph$(\mathcal{P}, G)$
  until no improvement is possible

further increase calculation speed or generalize its scope of use. We present these modifications in four different directions, each detailed in its respective subsection:

- the modification of the algorithm's core, including its initialization, stopping criterion, and VM;

- the use of quality functions other than modularity, functions which may be similar to modularity but with other parameters, or functions which may be completely different;
- generalizations of the Louvain method to deal with more than simple graphs. These may include weighted, oriented, spatially constrained and temporal networks;
- and finally the use of parallelism to speed up computation time.

4.1. Modification of the algorithm

The original article and an extended version of it [36] (published in French in 2011 and in English in 2013) already described several improvements that have been proposed independently and studied in greater detail by other authors:

- a partial optimization that consists of stopping the optimization when the modularity gain is below a given threshold (both for iterations or aggregation). This very simple optimization was already implemented in the initial version of the Louvain method;
- removing leaf vertices, which consists of deleting the leaves since they can only be placed in the community of their single neighbor;
- a VM that consists of not considering all vertices at each iteration;
- returning to lower levels to move vertices again;
- a simulated annealing-inspired VM allowing non-optimal moves.

In the following, we describe modifications proposed in the different steps of the algorithm, that is, its initialization, the VM step and the aggregation step.

Initialization: several types of optimization have targeted the initialization phase. The first one reduces the size of the graph before computation by removing a number of vertices that have little or no impact on the results. Based on the observation that a leaf (any vertex with a single neighbor) will inevitably be placed in its neighbor's community, several authors propose deleting these leaves [36, 37] or, more generally, all tree structures, i.e. the initial leaves and all those that will appear when deleting leaves [38]. The resulting graph corresponds to the 2-core of the original graph and the leaves are added back once the communities are found. A generalization has been proposed in [39] where a k-core decomposition is performed and communities are only computed on the k-core. All the removed vertices are then added back and assigned to a community. The authors propose to use label propagation or any modularity maximization algorithm for this task.

Other approaches have modified the initial partition by not starting with communities containing a single vertex. The authors of [40] identify cliques containing at least three vertices that serve as initial communities. However, the way cliques are identified and the management of overlaps between these cliques are not made explicit.

Vertex mover: in the original Louvain method, the best choice is always preferred during the VM operation. Based on the principle of simulated annealing, it was proposed to choose a non-optimal move, even with a negative gain, with a low probability rather than the optimal one [41].

J. Stat. Mech. (2024) 10R001

Several articles propose to avoid considering all vertices at each iteration. The simplest version is defined in [42]. If a vertex does not move for several iterations (with a fixed parameter), then it will not be considered anymore. In [38], the authors show that if two communities A and B have not changed in the last iteration, then no vertex can move from A to B or from B to A in the current iteration. A similar idea is used in [43], where the authors identify vertices that are more likely to move in the next iteration and only consider these vertices. They propose four situations: the simplest of which considers that if a vertex i has just moved from the community A to the community B, then all i's neighbors who are not in B are likely to join i at the next iteration. The other three situations are a little more complex and are not used in practice. Limiting oneself to considering only a few vertices at the next iteration could lead to iterations being stopped prematurely and thus to a loss of quality; however, in practice, the impact is very slight.

In a different manner, [37] uses a concept of seed vertices, which are vertices more central than the average (for a given centrality measure defined in the paper). Non-seed vertices are then moved in priority to the neighboring community that contains a seed vertex and maximizes the (positive) gain of modularity. If there is no such neighboring community, or if no gain is positive, then the classical VM is used.

In [44], instead of considering all neighbor communities and picking the best, one neighbor is picked at random and only its community is considered—other random selection procedures have also been considered, e.g. a community is selected proportionally to its size. Even though random selection will not always make the best choice, if a vertex has many neighbors in a given community, then this community is more likely to be picked. The author notes that this operation accelerates convergence. This principle has been reused in the Leiden method [45] with other elements, ensuring that the communities found by the algorithm are connected and making the Leiden method a popular alternative to the Louvain method in practice.

In most implementations of the Louvain method, the order in which vertices are considered is random. Although this can be seen as a problem, it does allow us to explore more possible solutions. If the aim is to obtain an identical partition, any fixed order can be used. In [46], the authors treat the vertices by decreasing centrality. This choice appears to improve results slightly, but above all, it speeds up the algorithm.

Other improvements have been proposed to simplify the VM in a distributed computing context. We describe them in section 4.4.

Aggregation and refinement: a key element of the Louvain method is that the VM operation is not greedy since any given vertex can be moved several times until convergence is reached. As this step is not irreversible, it is less likely to be penalized by poor initial choices. However, the aggregation phases are irreversible, and the original method does not allow one to go back once the graph has been aggregated. For this reason, a number of papers have been proposed to challenge this principle and allow one to move lower-level vertices, even after aggregation. This procedure is called partition refinement.

In the simpler versions [47], a single refinement of the partition is performed on the first level, i.e. the original graph where all vertices are subject to the VM procedure

J. Stat. Mech. (2024) 10R001

until convergence. In [48, 49], a multi-level refinement is used where, starting from the top level of the partition, the graph is unaggregated once and vertices are allowed to move. This process is repeated by moving down the hierarchy until they are back on the initial graph. A different refinement is proposed in [50, 51], where the subgraph induced by each community is partitioned using one level or the complete Louvain method. All these parts of each community are used to build the aggregated network. A more complex scheme is used in the Leiden method [45], where the VM during refinement is limited and a move is accepted if it increases the modularity, even though it is not the best move. Finally, in [52], if small (less than four vertices) or weak (that score low using their quality function) communities are found, these communities are broken into individual vertices and a VM step is applied once.

The Louvain method and its evolutions do not impose *a priori* constraints on the number or size of communities. If the number of communities is known to be k, then [53] propose to stop the aggregation in the Louvain method at the highest level that contains more than k communities (i.e. the next aggregation would result in fewer than k communities). They then use a greedy method to merge the communities that maximize the modularity gain to obtain exactly k communities. Finally, VM iterations are performed until convergence.

4.2. Modification of the quality function

The Louvain method is often understood as synonymous to modularity optimization. Yet, this heuristic has been used to optimize other quality functions, often considered in view of the known shortcomings of modularity. Several alternative quality functions are minor modifications of modularity. But others are based on entirely different principles. In general, the quality functions consider a positive contribution for intra-community edges and a negative contribution for inter-community edges or in the absence of edges. In the cases described in this section, the whole structure of the Louvain algorithm remains unchanged, and the only difference arises when estimating the local changes in the quality function when moving nodes between communities.

- Several variations of modularity use a resolution parameter, and can be shown to be equivalent, up to a trivial map, to the so-called Potts modularity [32]

$$Q_\gamma = \frac{1}{2m}\sum_{\alpha\in\mathcal{P}}\sum_{i,j\in\alpha}\left(A_{ij} - \gamma\frac{k_i k_j}{2m}\right), \tag{4}$$

  where the resolution parameter γ allows one to give more or less importance to the negative term, and hence to modulate the size of the communities. This expression has the same optimum as the so-called Markov stability, where the quality of a partition is estimated by its tendency to capture random walkers for long times inside communities [25, 33, 54] and where time plays the role of the resolution parameter. Most implementations of the Louvain method have been generalized for one (or both) of these equivalent variations.

- The balanced modularity [35, 55] uses a symmetric version that also considers the absence of links

$$\frac{1}{2m}\sum_{i,j\in V}\left[\left(a_{ij}-\frac{k_ik_j}{2m}\right)x_{ij}-\left(\bar{a}_{ij}-\frac{(n-k_i).(n-k_j)}{n^2-2m}\right)\bar{x}_{ij}\right].$$

- The authors of [41] use a function that considers both edges within communities and edges between communities. The linear version of their function is

$$\frac{1}{2m}\sum_{i,j\in V}\left[\left(a_{ij}-\frac{k_ik_j}{2m}\right)x_{ij}-\beta\left(a_{ij}-\frac{k_ik_j}{2m}\right)^{\alpha}\bar{x}_{ij}\right],$$

  where α and β are resolution parameters that can be tuned to adjust the size of communities.
- In [56], the authors propose the Jaccard Cosine Share Measure. For two vertices i and j, the main term combines Jaccard (the size of the intersection of neighborhoods of i and j divided by the size of their union), Cosine (the normalized dot product of rows i and j of the adjacency matrix) and classical modularity ($a_{ij}-k_ik_j/2m$).
- Other functions mostly based on the number of connections within or between communities have been proposed. For instance, [52] propose a new function defined as the sum for each community of

$$F2\left(C_i\right)=\frac{\left[d_{\text{in}}\left(C_i\right)\right]^2}{\left[d_{\text{in}}\left(C_i\right)+d_{\text{out}}\left(C_i\right)\right]^2}.$$

- Based on principles other than modularity, the Map equation introduced in [9] uses the length of the description of random walks in a network. The equation uses a two-level description to encode both the modules and the vertices. Vertices from different modules can use the same code (the authors use the analogy of street names, which may be similar in different cities). The optimization of the Map equation closely follows the steps of the Louvain method. Note that the Map equation has also been generalized to uncover a hierarchy of communities [51]. The Map equation is a solid alternative to modularity, even if it is also known to have its own limitations, including a field-of-view limit leading to undesirable overpartitioning into communities [57].

More generally, works have explored the conditions that need to be satisfied by a quality function so that it can be optimized by the Louvain method. In that direction, the authors of [35] have used relational coding to describe several quality functions using the same formalism. A quality function F is said to be linear if $F(X,A)=\sum_{i,j\in V}\phi(a_{ij})x_{ij}+K$, where $a_{ij}=1$ if i and j are connected (0 otherwise), $x_{ij}=1$ if i and j are in the same community (0 otherwise), ϕ and K are, respectively, a function and a constant that only depend on the original data. For example, modularity can be expressed as $Q=\sum_{i,j\in V}\left(a_{ij}-k_ik_j/2m\right)x_{ij}$. If a quality function is linear, then the gain obtained using a VM can be computed locally: henceforth, without loss

J. Stat. Mech. (2024) 10R001

of efficiency. The authors also study separable quality functions, i.e. that can be written as $F(X,A)=\sum_{i,j\in V}\phi(a_{ij})\psi(x_{ij})+K$, where ψ only depends on X. If the function ψ only depends on the community of interest, then the corresponding quality function can also be used efficiently in the Louvain method. The authors study five non-linear but separable functions and clarify why only some can be optimized efficiently. A different formulation is given in [58]. The Louvain method can be used to optimize any quality function of the form trace $H^\top[F-ab^\top]H$, where H is the partition indicator matrix, F is a general matrix derived from the network and a and b are two n dimensional vectors.

4.3. Louvain for non-simple graphs

Modularity was originally defined for undirected unweighted networks. Since then, several generalizations have been proposed to allow more complex graph structures. Compared to the expression equation (3), the first term often remains unchanged, to count the total number, or total weight, of edges inside communities; but the second term is modified to take into account the additional constraints induced by the graph structure. For instance, in directed networks, the null model is now chosen according to the directed configuration model, to properly estimate the expected number of edges between two vertices, given their respective in-degree and out-degree [59]. The Louvain method can then be directly generalized with an adapted modularity gain during VM [60]. Other examples include the optimization of modularity for spatial networks where, again, the null model of the quality function can be modified to account for spatial constraints [21]. Alternative approaches keep standard expressions for modularity but instead modify the Louvain method to impose a spatial contiguity of the clusters in the uncovered partition [61].

Community detection is also an active field of study for temporal networks [62, 63]. Compared to its static counterpart, communities are now dynamical objects that may encounter different types of change, such as growth or splitting, and algorithms have the additional task of characterizeing these events. Two different types of approach build on community optimization. First, communities can be uncovered in different snapshots. In this case, some constraints on the communities in adjacent snapshots may be incorporated to ensure their continuity in time, such as in modularity maximization under estrangement constraint [64]. After the communities have been found in each snapshot, additional methods determine, with a certain statistical accuracy, when and how a reorganization takes place [65]. A second family of methods aims to find a decomposition of the network in an extended network in which snapshots are concatenated, e.g. by connecting the same vertex in consecutive snapshots to favor continuity of communities across time. Here, instead of searching communities in each time window, the community detection is performed in one single operation, which can be formulated as a modularity maximization, on a larger multilayer network [66] and again optimized by the Louvain method [67].

Modularity has also been generalized to quantify the quality of the partition of signed networks. Here, the intuition places together vertices such that positive edges are concentrated within the modules and negative edges are between the modules. In

J. Stat. Mech. (2024) 10R001

its simplest setting, the signed modularity consists of dividing the signed graph into two graphs, one defined by the positive interactions and another one defined by the negative interactions, and searching to maximize the difference in the modularities of the positive and negative graphs [68]. The Louvain method can be directly generalized for this quality function, and the only step that needs to be considered with care is the VM. Consider a vertex i. One expects to find other vertices of the community of i in the neighborhood of i only for the positive edges; however, its negative neighbors should instead be placed in other communities. Ways to address this complication are either to consider all the communities in the network in the VM step—and not only the neighboring ones—but this involves a massive slow down of the algorithm, or to consider second neighbors, i.e. neighbors of neighbors, in the case of negative edges [69].

Finally, some works have focused on attributed networks where nodes carry labels. The aim is to create well-connected communities that are also homogeneous with regard to labels, to meet both structural and homophilic criteria. Three families of approaches are proposed in [70]: early fusion, where a new graph is built to encode both topology and attributes; simultaneous fusion, where the clustering is performed simultaneously on both domains; and late fusion, where both clusterings are computed independently before being joined. Among these, SAC2 [71] is an early fusion method where each node retains only its k-nearest neighbors based on a weighted combination of topology and similarity between attributes. The Louvain method is then used on this construction. Simultaneous fusion generally involves a composite function of modularity for the structural part and a similarity function for the attributes. This is the case in [72], using purity for attributes. Relatedly, SAC1 [71] uses different similarities for attributes, and [73] uses inertia. In all cases, the Louvain method is used to optimize these functions. A late fusion is proposed in [74], where structural communities are found with the Louvain method and attributed ones with k-means. The structural communities are only used if they are sufficiently good, otherwise the attributed ones are used. More generally, any consensus-based solution can be used to merge several partitions.

4.4. Distributed/parallel/GPU solutions

Many efforts have been made to parallelize the Louvain method, the main challenge being to perform the VM step on several CPUs simultaneously. There are two main paradigms: shared-memory implementations, where processes share the same memory and must therefore synchronize for writing; and distributed-memory implementations, where processors communicate via messages to maintain a consistent view. The former is limited by the number of physical processing units that can be used, and the latter by the quantity of messages sent. In most distributed implementations, if not all, the input network is partitioned so that vertices and edges are distributed evenly among the processing units. Evaluating the correctness of a parallel implementation can be complex. Because of the impact of the order in which vertices are processed, one does not expect to obtain the same results from one run to the next, or when compared with a sequential run.

Shared-memory implementations: in shared-memory implementations, several vertices simultaneously and independently evaluate the best VM. They all base their choice

J. Stat. Mech. (2024) 10R001

on a structure that may no longer be valid at the time of the decision. The assignment of vertices to communities is protected and can only happen in a sequential way so that no inconsistency can occur but a vertex i may have been assigned between the time a vertex j made its choice and the time it validated it. Hence, convergence might be slower or even not attainable. Several approaches have been considered.

- To keep the VM sequential, [75] propose an implementation by identifying the loops in the code that can be executed in parallel. This includes initialization of the algorithm, identifying all the neighboring communities of a vertex and finally identifying the neighboring community with the best gain. Community merging can also be partially parallelized.
- To avoid conflicts that may result in a slower convergence, one solution would be to find vertices that do not conflict with each other and parallelize them. While the extra cost is generally considered too high [76], some authors propose to identify isolated sets of vertices, i.e. vertices that are not neighbors of each other, and to process them in parallel [77]. The authors of [76] propose to neglect this risk of non-convergence but limit the number of iterations. They also compare asynchronous moves with synchronous moves, where all parallelly processed vertices are moved simultaneously.
- The authors of [78] propose to adapt the number of processes dynamically, depending on the number of neighbors of a vertex and available resources.
- In [79], the authors use techniques already presented to avoid considering all vertices at every iteration. In the first iterations they use a pull method, which is the classic method where neighboring communities are recalculated at each iteration for every vertex. In the later iterations there are fewer vertex moves; they switch to a push method, where neighboring communities are simply updated (pushed) with each vertex move. This optimization is combined with classic code parallelization, using locks for updates of neighboring communities.

Distributed-memory implementations: in distributed-memory implementations, vertices are distributed among the various processes. Each process will therefore store a given number of vertices. A number of problems arise that require algorithmic adaptation of the Louvain method. First, vertices or edges must be distributed evenly between processes to improve the overall calculation time. In complex networks, the presence of very large degree vertices (or hubs) makes vertices/edges distribution more complex. Second, if the ends of an edge are not on the same process, messages will have to be sent to synchronize them and the total number of messages must be kept as low as possible. Finally, since calculations are made independently, it is necessary to be able to question local choices. Several works have looked at, and addressed, these issues.

- In [80], the authors use three levels of parallelism: (1) the network is divided in subgraphs and edges between these subgraphs are ignored. The Louvain method is applied to each subgraph. Then, the individual results are joined, reintroducing the missing edges, and the Louvain method is reapplied. (2) vertices of each subnetwork are considered in batches and vertices of a given batch are processed in parallel. Potential conflicts if two vertices of a batch are neighbors are not considered but the updates

of the vertex's community are still performed sequentially to avoid inconsistency. (3) Modularity gain for neighboring communities is computed in parallel.

- Processes can store knowledge of the neighbors of the vertices they own even though such vertices are stored on another device. Such vertices are called 'ghost vertices'. Similarly, a process can store information about 'ghost' communities adjacent to the ones it owns. Ghost vertices and communities are used in [81]. Initially, a contiguous block of vertices are distributed, trying to balance the number of edges received by each process. At each iteration, some messages are exchanged between processes that own ghost vertices to update the information about community assignment. Graph coarsening is also performed in a distributed way to determine new global ids of communities and to exchange partial edge lists of neighboring communities so that each process can have a complete edge list. A similar procedure adapted for GPUs is detailed in [82].
- The authors of [83] use two hash tables to store the graph and also propose to identify the vertices that will provide the largest improvement in modularity and only consider such vertices to accelerate the VM step. They show that the number of such vertices decays exponentially fast so that only the first few iterations are costly.
- The initial distribution of vertices can have an impact on the workload and on the number of ghost vertices and, therefore, on the quantity of messages to be transmitted. This is particularly the case for hubs; therefore, it is worth choosing the initial partition carefully. In [80], the authors distribute vertices evenly based on their id. This is studied in more detail in [84], where the authors show that the degree distribution of such a partition is close to the degree distribution of the whole network, which ensures a fair workload. Then they propose an enhancement that also considers the degree of ghost vertices to improve workload fairness.
- To account for hubs it is also possible to duplicate them. Each duplicate (called a delegate) retains the neighbors belonging to its partition by default; however, this can be modified to ensure a better balance of edges between partitions [85]. Then, delegates communicate with each other to ensure synchronization. This general principal is used in [86] as a preprocessing step for a distributed Louvain implementation. The authors also take delegates into account when calculating the modularity gain.
- The authors of [87] use PMETIS, a parallel implementation of METIS [88] to obtain a partition of the network and distribute theses parts. Each one is then processed independently; then aggregation is performed and the rest of the algorithm is executed sequentially.
- Independent calculations on each process can result in incorrect communities if vertices are incorrectly assigned to a process. This problem is studied in [89] with the identification of doubtful vertices. The relative commitment for a vertex is calculated by computing the ratio between its internal degree and the maximum internal degree in its community, and a vertex's affinity for a community is calculated by taking into account the relative commitment of the vertex and its neighbors. Vertices with low commitment are considered doubtful and can be moved to another device if they have many connections with vertices on that device.

J. Stat. Mech. (2024) 10R001

5. Perspectives

In this article, we have overviewed the different ways in which the Louvain method has been used, generalized and improved over the last 15 years. Over this period of time, the field of community detection has radically evolved, with important contributions to the statistical foundations of the problem [90, 91], including theoretical characterizations of its detectability limit and the design of efficient spectral methods for sparse networks [92, 93], and also on connections with the theory of Markov chains, leading to the design of algorithms and quality functions based on random-walk processes [9, 33]. The limitations of modularity are now well understood and documented [28]; yet, due to its simplicity, it has remained a central component of the network science toolbox. Researchers have also argued that there is no general-purpose algorithm for community detection [94], and that algorithms based on different ingredients and principles, e.g. stochastic block models versus modularity, should be chosen depending on the context and the purpose of the user [4].

As we have shown, a wide variety of works have built on the original design of the Louvain method, exploring how to use it for other quality functions than just modularity, most notably the Map equation and Markov stability, and adapting its different steps for faster, more accurate, more general and/or distributed solutions. A great deal is known about the strengths and weaknesses of the method; yet it remains an active source of inspiration as well as a standard competitive method to cluster large-scale networks in practice. Most importantly, due to its generality, the heuristic can be deployed for a variety of quality functions, beyond modularity, and hence allows one to explore the community structure of networks from complementary perspectives. Despite its many successes, and the many subsequent works described in this article, several questions remain open, especially on the theoretical foundations of the Louvain method. First, its algorithmic complexity has been shown numerically to be close to linear, at least in networks that are not 'too random' [36], so that the underlying structure of the graph can be exploited to accelerate the convergence; however, a more thorough theoretical analysis would be welcome. In addition, it is important to remember that the Louvain method is a heuristic, and that there is no guarantee that it will converge to the global maximum of the quality function. Interesting future research includes the determination of bounds on the accuracy of the method, and how they depend on the structure of the graph and the presence of noisy edges.

Never did we imagine the impact of the method when it was originally released, and it is equally difficult for us to foresee now what will become of it in the future. The method has accompanied the different trends that have emerged in network science, from multiplex networks to embeddings and temporal networks. For this reason, it is likely that the Louvain method will not only find applications and generalizations in the increasingly popular field of higher-order networks [95–98], but also in continuous generalizations of networks like graphons [99] and in extensions of graphs to non-real weights [100, 101]. But we are even more sure that we will be surprised, and we look forward to sitting down in 15 years, discussing together, discovering in awe the many ways the research community has modified our original idea and, who knows, sharing it with you [102].

J. Stat. Mech. (2024) 10R001

Acknowledgments

Our special thanks go to Etienne Lefebvre, without whom the Louvain method would never have existed. Only one publication as a master's student, but what a success! We also thank John Pougué Biyong for his help with the analysis of the Scopus data, and the many colleagues with whom we have collaborated on this topic over the year; in particular, Thomas Aynaud, Mauricio Barahona, Romain Campigotto, Patricia Conde-Céspedes, Jean-Charles Delvenne, Mason Porter, Martin Rosvall and Michael Schaub. Finally, many thanks to Sébastien Amoury and Shazia Babul for carefully reading this manuscript.

The work of R L was supported by the EPSRC Grants EP/V013068/1 and EP/V03474X/1. The work of J L G was supported by the ANR MITIK project, French National Research Agency (ANR), PRC AAPG2019.

References

[1] Newman M 2018 *Networks* (Oxford University Press)
[2] Fortunato S 2010 *Phys. Rep.* **486** 75–174
[3] Fortunato S and Newman M E 2022 *Nat. Phys.* **18** 848–50
[4] Schaub M T, Delvenne J-C, Rosvall M and Lambiotte R 2017 *Appl. Netw. Sci.* **2** 1–13
[5] Garza S E and Schaeffer S E 2019 *Physica* A **534** 122058
[6] Devooght R, Mantrach A, Kivimäki I, Bersini H, Jaimes A and Saerens M 2014 Random walks based modularity: application to semi-supervised learning *Proc. 23rd Int. Conf. on World Wide Web* pp 213–24
[7] Newman M E and Girvan M 2004 *Phys. Rev.* E **69** 026113
[8] Von Luxburg U 2007 *Stat. Comput.* **17** 395–416
[9] Rosvall M and Bergstrom C 2008 *Proc. Natl Acad. Sci. USA* **105** 1118–23
[10] Lambiotte R, Delvenne J-C and Barahona M 2014 *IEEE Trans. Netw. Sci. Eng.* **1** 76–90
[11] Peixoto T P 2013 *Phys. Rev. Lett.* **110** 148701
[12] Clauset A, Newman M E and Moore C 2004 *Phys. Rev.* E **70** 066111
[13] Wakita K and Tsurumi T 2007 Finding community structure in mega-scale social networks *Proc. 16th Int. Conf. on World Wide Web* pp 1275–6
[14] Blondel V D, Guillaume J-L, Lambiotte R and Lefebvre E 2008 *J. Stat. Mech.* 10008
[15] Lambiotte R and Panzarasa P 2009 *J. Inf.* **3** 180–90
[16] Newman M E 2001 *Phys. Rev.* E **64** 016132
[17] Watts D J and Strogatz S H 1998 *Nature* **393** 440–2
[18] Broido A D and Clauset A 2019 *Nat. Commun.* **10** 1017
[19] Simon H A 1977 *Models of Discovery: And Other Topics in the Methods of Science* (D. Reidel) pp 245–61
[20] MacQueen J B 1967 Some methods for classification and analysis of multivariate observations *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability* vol 1, ed L M L Cam and J Neyman (University of California Press) pp 281–97
[21] Expert P, Evans T S, Blondel V D and Lambiotte R 2011 *Proc. Natl Acad. Sci.* **108** 7663–8
[22] Chung F and Lu L 2002 *Ann. Comb.* **6** 125–45
[23] Peel L, Delvenne J-C and Lambiotte R 2018 *Proc. Natl Acad. Sci.* **115** 4057–62
[24] Devriendt K, Martin-Gutierrez S and Lambiotte R 2022 *SIAM Rev.* **64** 343–59
[25] Delvenne J-C, Yaliraki S N and Barahona M 2010 *Proc. Natl Acad. Sci.* **107** 12755–60
[26] Brandes U, Delling D, Gaertler M, Gorke R, Hoefer M, Nikoloski Z and Wagner D 2007 *IEEE Trans. Knowl. Data Eng.* **20** 172–88
[27] Newman M E J 2013 *Phys. Rev.* E **88** 042822
[28] Ghasemian A, Hosseinmardi H and Clauset A 2019 *IEEE Trans. Knowl. Data Eng.* **32** 1722–35
[29] Guimera R, Sales-Pardo M and Amaral L A N 2004 *Phys. Rev.* E **70** 025101
[30] Fortunato S and Barthelemy M 2007 *Proc. Natl Acad. Sci.* **104** 36–41
[31] Good B H, De Montjoye Y-A and Clauset A 2010 *Phys. Rev.* E **81** 046106

J. Stat. Mech. (2024) 10R001

[32] Reichardt J and Bornholdt S 2006 *Phys. Rev.* E **74** 016110
[33] Lambiotte R, Delvenne J C and Barahona M 2008 Laplacian dynamics and multiscale modular structure in networks (arXiv:0812.1770)
[34] Delvenne J C, Schaub M T, Yaliraki S N and Barahona M 2013 *Dynamics On and of Complex Networks* (*Applications to Time-Varying Dynamical Systems* vol 2) (Springer) pp 221–42
[35] Campigotto R, Conde Céspedes P and Guillaume J L 2014 (arXiv:1406.2518)
[36] Aynaud T, Blondel V D, Guillaume J L and Lambiotte R 2013 *Multilevel Local Optimization of Modularity* (Wiley) ch 13, pp 315–45
[37] Lui D, Huang K, Zhang C, Wu D and Wu S 2021 *Sci. Program.* **2021** 3234280
[38] Zhang J, Fei J, Song X and Feng J 2021 *Math. Problems Eng.* **2021** 1–14
[39] Peng C, Kolda T G and Pinar A 2014 Accelerating community detection by using k-core subgraphs (arXiv:1403.2226)
[40] Abbas E A and Nawaf H N 2020 Improving louvain algorithm by leveraging cliques for community detection *2020 Int. Conf. on Computer Science and Software Engineering (CSASE)* pp 244–8
[41] Zhou Z, Wang W and Wang L 2012 Community detection based on an improved modularity *Pattern Recognition* ed C L Liu, C Zhang and L Wang (Springer) pp 638–45
[42] Ryu S and Kim D 2016 Quick community detection of big graph data using modified louvain algorithm *2016 IEEE 18th Int. Conf. on High Performance Computing and Communications; IEEE 14th Int. Conf. on Smart City; IEEE 2nd Int. Conf. on Data Science and Systems (HPCC/SmartCity/DSS)* pp 1442–5
[43] Ozaki N, Tezuka H and Inaba M 2016 *Int. J. Comput. Electr. Eng.* **8** 207–18
[44] Traag V A 2015 *Phys. Rev.* E **92** 032801
[45] Traag V A, Waltman L and van Eck N J 2019 *Sci. Rep.* **9** 5233
[46] Aldabobi A, Sharieh A and Riad J 2022 An improved Louvain algorithm based on Node importance for Community detection *J. Theor. Appl. Inf. Technol.* **100** 1–14 (available at: www.jatit.org/volumes/Vol100No23/21Vol100No23.pdf)
[47] Du W and He X 2016 A common strategy to improve community detection performance based on the nodes' property *Bio-Inspired Computing – Theories and Applications* ed M Gong, L Pan, T Song and G Zhang (Springer) pp 355–61
[48] Gach O and Hao J K 2014 Improving the Louvain algorithm for community detection with modularity maximization *Artificial Evolution: 11th Int. Conf., Evolution Artificielle, EA 2013, (Bordeaux, France, 21 October–23 October 2013* (Springer) pp 145–56
[49] Rotta R and Noack A 2011 *ACM J. Exp. Algorithmics* **16** 376
[50] Waltman L and van Eck N J 2013 *Eur. Phys. J.* B **86** 471
[51] Rosvall M and Bergstrom C T 2011 *PLoS One* **6** e18209
[52] Yao B, Zhu J, Ma P, Gao K and Ren X 2023 *Appl. Sci.* **13** 4045
[53] Darst R K, Nussinov Z and Fortunato S 2014 *Phys. Rev.* E **89** 032809
[54] Arnaudon A, Schindler D J, Peach R L, Gosztolai A, Hodges M, Schaub M T and Barahona M 2024 *ACM Trans. Math. Softw.* **50** 15
[55] Céspedes P C and Marcotorchino J F 2013 Comparing different modularization criteria using relational metric *Geometric Science of Information* ed F Nielsen and F Barbaresco (Springer) pp 180–7
[56] Chaudhary L and Singh B 2019 Community detection using an enhanced Louvain method in complex networks *Distributed Computing and Internet Technology* ed G Fahrnberger, S Gopinathan and L Parida (Springer International Publishing) pp 243–50
[57] Schaub M T, Lambiotte R and Barahona M 2012 *Phys. Rev.* E **86** 026112
[58] Schaub M, Delvenne J, Lambiotte R and Barahona M 2019 *Phys. Rev.* E **99** 062308
[59] Leicht E A and Newman M E 2008 *Phys. Rev. Lett.* **100** 118703
[60] Li L, He X and Yan G 2018 Improved Louvain method for directed networks *Intelligent Information Processing IX: 10th IFIP TC 12 Int. Conf., IIP 2018, (Nanning, China, 19 October–22 October 2018), Proc.* vol *10* (Springer) pp 192–203
[61] Wang C, Wang F and Onega T 2021 *Trans. GIS* **25** 1065–81
[62] Cazabet R and Rossetti G 2019 *Temporal Network Theory* (Springer) pp 181–97
[63] Masuda N and Lambiotte R 2016 *A Guide to Temporal Networks* (World Scientific)
[64] Kawadia V and Sreenivasan S 2012 *Sci. Rep.* **2** 794
[65] Greene D, Doyle D and Cunningham P 2010 Tracking the evolution of communities in dynamic social networks *2010 Int. Conf. on Advances in Social Networks Analysis and Mining* (IEEE) pp 176–83
[66] Mucha P J, Richardson T, Macon K, Porter M A and Onnela J-P 2010 *Science* **328** 876–8

J. Stat. Mech. (2024) 10R001

[67] Jeub L G S, Bazzi M, Jutla I S and Mucha P J 2019 A generalized Louvain method for community detection implemented in MATLAB, Ver. 2.2 (available at: https://github.com/GenLouvain/GenLouvain)
[68] Traag V A and Bruggeman J 2009 *Phys. Rev.* E **80** 036115
[69] Pougué-Biyong J and Lambiotte R 2024 SignedLouvain: Louvain for signed networks (arXiv:2407.19288) (in preparation)
[70] Chunaev P 2020 *Comput. Sci. Rev.* **37** 100286
[71] Dang T and Viennet E 2012 Community detection based on structural and attribute similarities *Int. Conf. on Digital Society* pp 7–12
[72] Citraro S and Rossetti G 2020 *Appl. Netw. Sci.* **5** 1–20
[73] Combe D, Largeron C, Géry M and Egyed-Zsigmond E 2015 I-Louvain: an attributed graph clustering method *Intelligent Data Analysis* (LaHC, University of Saint-Etienne)
[74] Elhadi H and Agam G 2013 Structure and attributes community detection: comparative analysis of composite, ensemble and selection methods *Proc. 7th Workshop on Social Network Mining and Analysis SNAKDD'13* (Association for Computing Machinery)
[75] Bhowmick S and Srinivasan S 2013 *A Template for Parallelizing the Louvain Method for Modularity Maximization* (Springer) pp 111–24
[76] Shi J, Dhulipala L, Eisenstat D, Łăcki J and Mirrokni V 2021 *Proc. VLDB Endow.* **14** 2305–13
[77] Qie H, Li S, Dou Y, Xu J, Xiong Y and Gao Z 2022 *Sci. Rep.* **12** 8248
[78] Fazlali M, Moradi E and Tabatabaee Malazi H 2017 *Microprocess. Microsyst.* **54** 26–34
[79] Tithi J J, Stasiak A, Aananthakrishnan S and Petrini F 2020 Prune the unnecessary: parallel pull-push Louvain algorithms with automatic edge pruning *Proc. 49th Int. Conf. on Parallel Processing ICPP'20* (Association for Computing Machinery)
[80] Cheong C Y, Huynh H P, Lo D and Goh R S M 2013 Hierarchical parallel algorithm for modularity-based community detection using gpus *Proc. 19th Int. Conf. on Parallel Processing Euro-Par'13* (Springer) pp 775–87
[81] Ghosh S, Halappanavar M, Tumeo A, Kalyanaraman A, Lu H, Chavarriá-Miranda D, Khan A and Gebremedhin A 2018 Distributed Louvain algorithm for graph community detection *2018 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)* pp 885–95
[82] Gawande N, Ghosh S, Halappanavar M, Tumeo A and Kalyanaraman A 2022 *Parallel Comput.* **111** 102898
[83] Que X, Checconi F, Petrini F and Gunnels J A 2015 Scalable community detection with the Louvain algorithm *2015 IEEE Int. Parallel and Distributed Processing Symp.* pp 28–37
[84] Zeng J and Yu H 2015 Parallel modularity-based community detection on large-scale graphs *Proc. 2015 IEEE Int. Conf. on Cluster Computing CLUSTER'15* (IEEE Computer Society) pp 1–10
[85] Pearce R, Gokhale M and Amato N M 2014 Faster parallel traversal of scale free graphs at extreme scale with vertex delegates SC'14 *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis* pp 549–59
[86] Zeng J and Yu H 2018 A scalable distributed Louvain algorithm for large-scale graph community detection *2018 IEEE Int. Conf. on Cluster Computing (CLUSTER)* pp 268–78
[87] Wickramaarachchi C, Frincu M, Small P E and Prasanna V K 2014 *2014 IEEE High Performance Extreme Computing Conf. (HPEC)* pp 1–6
[88] Karypis G and Kumar V 1998 *SIAM J. Sci. Comput.* **20** 359–92
[89] Bhowmick A, Vadhiyar S and Varun P V 2022 *Concurr. Comput.* **34** e6987
[90] Abbe E 2018 *J. Mach. Learn. Res.* **18** 1–86
[91] Moore C 2017 arXiv:1702.00467
[92] Decelle A, Krzakala F, Moore C and Zdeborová L 2011 *Phys. Rev. Lett.* **107** 065701
[93] Krzakala F, Moore C, Mossel E, Neeman J, Sly A, Zdeborová L and Zhang P 2013 *Proc. Natl Acad. Sci.* **110** 20935–40
[94] Peel L, Larremore D B and Clauset A 2017 *Sci. Adv.* **3** e1602548
[95] Lambiotte R, Rosvall M and Scholtes I 2019 *Nat. Phys.* **15** 313–20
[96] Battiston F, Cencetti G, Iacopini I, Latora V, Lucas M, Patania A, Young J G and Petri G 2020 *Phys. Rep.* **874** 1–92
[97] Bianconi G 2021 *Higher-Order Networks* (Cambridge University Press)
[98] Bick C, Gross E, Harrington H A and Schaub M T 2023 *SIAM Rev.* **65** 686–731
[99] Caron F and Fox E B 2017 *J. R. Stat. Soc.* B **79** 1295–366
[100] Böttcher L and Porter M A 2022 arXiv:2212.06257
[101] Tian Y and Lambiotte R 2023 arXiv:2307.01813
[102] Blondel V, Guillaume J L and Lambiotte R 2038 (in progress)

J. Stat. Mech. (2024) 10R001