

Fast consensus clustering in complex networksAditya Tandon,¹ Aiiad Albeshri,² Vijey Thayananthan,² Wadee Alhalabi,² and Santo Fortunato^{3,1}¹*School of Informatics, Computing and Engineering, Indiana University, Bloomington, Indiana 47408, USA*²*Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Kingdom of Saudi Arabia*³*Indiana University Network Science Institute (IUNI), Bloomington, Indiana 47408, USA*

(Received 5 February 2019; published 1 April 2019)

Algorithms for community detection are usually stochastic, leading to different partitions for different choices of random seeds. Consensus clustering has proven to be an effective technique to derive more stable and accurate partitions than the ones obtained by the direct application of the algorithm. However, the procedure requires the calculation of the consensus matrix, which can be quite dense if (some of) the clusters of the input partitions are large. Consequently, the complexity can get dangerously close to quadratic, which makes the technique inapplicable on large graphs. Here, we present a fast variant of consensus clustering, which calculates the consensus matrix only on the links of the original graph and on a comparable number of additional node pairs, suitably chosen. This brings the complexity down to linear, while the performance remains comparable as the full technique. Therefore, our fast consensus clustering procedure can be applied on networks with millions of nodes and links.

DOI: [10.1103/PhysRevE.99.042301](https://doi.org/10.1103/PhysRevE.99.042301)**I. INTRODUCTION**

Detecting communities in networks is a fundamental part of network analysis [1–3]. It is an unsupervised classification problem and as such it is ill-defined: different techniques typically find different partitions on the same network, unless the latter has a clear-cut community structure, with very dense groups which are very loosely connected to each other.

Another important problem of the field is that community detection algorithms are *noisy*. Since many algorithms require the use of random numbers, for initialization, optimization, tie-breaks, etc., different random seeds may lead to different outcomes, even when the same method is applied on the same network. Consensus clustering is an effective technique to decrease the noise induced by the stochasticity of methods. The procedure consists of “averaging” over a set of input partitions: the result, called *median* or *consensus* partition is usually more robust and accurate than the input partitions [4]. Consensus clustering is now regularly used in the network science community [5–10]. It requires the computation of a *consensus matrix*, expressing how often any pair of nodes is found in the same cluster in the input partitions. The calculation concerns only the pairs of nodes that happen to be co-clustered at least once. This means that all pairs of nodes of any cluster of the input partitions will give non-vanishing entries in the consensus matrix. In particular, if the largest cluster in any input partition has a non-negligible size with respect to the whole network, the consensus matrix can become quite dense and both the space and the time complexity of the calculation could get close to $O(n^2)$, where n is the number of nodes of the graph. Such high complexity limits the applicability of the procedure to networks which are not too large.

In this paper we propose a fast consensus clustering technique, which is applicable to large networks. The basic idea is to compute the consensus matrix for a very small set of properly selected pairs of nodes. This way the matrix will be sparse all along the calculation, which can reach linear complexity, if the clustering technique used to detect the communities has itself linear complexity. If instead we have reasons to prefer to use a slower clustering algorithm its complexity will be dominating compared to the other steps of the fast consensus clustering routine developed here.

II. THE METHOD

Before we get into the details of our technique, we briefly summarize the consensus clustering method exposed in Ref. [4], which will be our reference and will be called *LF consensus* throughout. We start from a graph G , with n nodes and m links, and a clustering algorithm of our choice. We apply the algorithm n_p times on G , with different random seeds, obtaining n_p partitions, which are the input partitions of the method. The *consensus matrix* \mathbf{D} is an $n \times n$ matrix. Each entry D_{ij} corresponds to the fraction of input partitions in which nodes i and j belong to the same cluster. For instance, if nodes 3 and 8 are put in the same cluster in 10 out of 20 input partitions, $D_{38} = D_{83} = 10/20 = 1/2$. By construction, the consensus matrix is then a weighted matrix, and its entries are real numbers between zero and one. Once the consensus matrix has been constructed, the original community detection algorithm is applied on it n_p times. Since all pairs of nodes in the same cluster in any partition yield a contribution, the consensus matrix can be very dense, even if the initial network is sparse. This could be a problem when we apply the clustering algorithm on it, for two reasons: 1) many algorithms have troubles to detect communities on dense matrices; 2)

the computational complexity of the procedure could get very high. Therefore, before the clustering algorithm is applied, \mathbf{D} is filtered, in that all entries below a certain threshold τ are set to zero. Then a new consensus matrix \mathbf{D}' is built from the new partitions, and so on. The process is repeated until all partitions are identical, i.e., the consensus matrix has only entries equal to zero and one.

This procedure can considerably improve the accuracy of the result, with respect to the set of initial partitions, but the construction of the consensus matrix has a worst time and space complexity of $O(n^2)$, irrespective of the sparsity of the original network, which makes the technique prohibitively expensive on large networks.

To save both space and time, the consensus matrix should be computed only for a small subset of all eligible node pairs. The natural candidates are the pairs of nodes which are connected to each other [10]. This choice proves to be a valid one. However, there are cases in which the gain in the accuracy of the consensus partitions is very modest if we consider only the pairs of neighboring nodes. Therefore we added some additional pairs, chosen such that they close triads with the links of the original graph, provided the weights of those links in the consensus matrix are sufficiently large. This leads to the following routine:

(1) **Derive the input partitions.** Apply the clustering algorithm on the network at study n_p times, to obtain the input partitions. In all the tests shown here we took $n_p = 20$.

(2) **Construct the consensus matrix.** Compute only the elements \mathbf{D}_{ij} , where i and j are neighbors in G .

(3) **Thresholding.** Set to zero all elements of \mathbf{D} below a threshold value τ . This removes weak links and speeds up convergence. By doing so, some nodes might get disconnected from the graph corresponding to the matrix. If a node gets disconnected, keep it attached to the rest of the graph by preserving the link with the highest weight. This way the graph is connected at all times.

(4) **Triadic closure.** Select m random nodes. For each node select at random a pair of neighboring nodes j and k . If the entry $\mathbf{D}_{jk} = 0$, then we set it equal to the fraction of partitions in which nodes j and k co-occur in the same cluster.

(5) Apply the clustering algorithm on the new weighted graph \mathbf{D} repeatedly to get n_p new partitions.

(6) Repeat steps 2–5 until convergence.

Convergence is reached when less than 2% of all non-zero entries of \mathbf{D} have weights smaller than one (a weight of one implies that the two nodes co-occur in the same cluster in all input partitions). The output is the matrix \mathbf{D}_{out} . We then apply the community detection algorithm on \mathbf{D}_{out} to get the final set of partitions, which represent the output of our technique. The 2% threshold is suggested by our numerical experiments: for 1%, which was our initial choice, the number of steps required until convergence sensibly increases. However, we have verified that results are stable for thresholds at least up to 10%, so the actual value is not important, provided it is not too low.

The main difference from the procedure of Ref. [4] is the fact that we compute up to $2m$ elements of \mathbf{D} , which is then very sparse. The advantage is that the calculation of \mathbf{D} has space and time complexity $O(m)$, which is much lower than the $O(n^2)$ of the full method when the graph at hand is

sparse ($m \propto n$), as it usually happens for real networks. On the downside, the partitions obtained by running a clustering algorithm on \mathbf{D} are a bit more noisy than in the full procedure and it is unlikely to converge to a set of identical partitions in the end. This is why we accept to stop when \mathbf{D}_{out} has but a modest proportion of entries different from zero and one. However, with our criterion convergence is usually reached within a handful of iterations, and the output partitions turn out to be of superior quality than the set of input partitions, for all community detection methods we have used in our experiments.

For our calculations we used Python implementations of the algorithms from the NetworkX [11] and igraph [12] libraries. The software to perform our fast consensus clustering procedure can be found in Ref. [13].

III. RESULTS

For our tests we used artificial benchmark graphs with a built-in community structure. Specifically, we adopted the Lancichinetti-Fortunato-Radicchi (LFR) benchmark graphs, which have become a standard in the evaluation of the performance of clustering algorithms [14,15]. LFR graphs are characterized by power-law distributions of node degree and community size, features that frequently occur in real world networks. The *mixing parameter* μ is the ratio between the external degree of a node with respect to its community (i.e., the number of links joining the node to its neighbors outside its community) and its total degree. So, when μ is close to zero, the nodes have most of their neighbors within their communities, which are then well separated from each other and easily detectable. The larger μ the fuzzier the communities and the more difficult it gets to identify them. The performance of a method on the LFR networks will be estimated by computing the *normalized mutual information* (NMI) between the built-in partition of the graph and the one detected by the clustering algorithm, as a function of μ . We used the modified version of the NMI introduced by Lancichinetti, Fortunato, and Kertész [16], to make the results comparable with those of Ref. [4]. For each μ -value we created 20 benchmark graphs (unless specified otherwise) and averaged the corresponding NMI scores among them.

We will show the results obtained by integrating our consensus technique with the following three clustering algorithms:

(i) *Fast greedy modularity optimization.* It is a technique developed by Clauset, Newman, and Moore (CNM) [17], that performs a quick maximization of the modularity by Newman and Girvan [18].

(ii) *Louvain method,* by Blondel *et al.* [19]. The goal is still the optimization of modularity, by means of a hierarchical approach. We will be using the first (bottom) level of the hierarchy generated by the method, the one with the smallest communities. This partition gives an excellent performance on LFR benchmark graphs [15]. The actual outcome of the procedure, which corresponds to the top level partition and the largest modularity, is known to be poorly correlated with the built-in partitions of the benchmark [3], mostly because of the resolution limit of modularity [20].

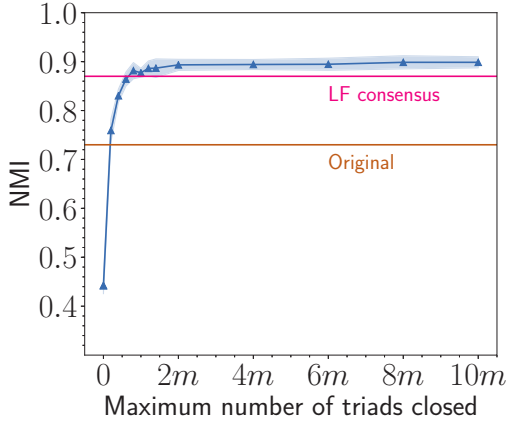


FIG. 1. Importance of triadic closure in the construction of a sparse consensus matrix. The accuracy of the resulting partition rapidly increases as we compute additional elements of the consensus matrix, by randomly closing triads formed by the (weighted) links of the initial matrix, after thresholding. Closing a modest number of triads suffices to outperform the direct application of the clustering method (here the Louvain algorithm), whose accuracy is indicated by the horizontal yellow line. The accuracy quickly climbs up to that of *LF consensus* (magenta line) and reaches about 0.87 for about m closed triads, remaining remarkably stable thereafter. The thickness of the blue line indicates the standard error of the mean. For the LFR benchmark graph used in this test we chose the following parameters: number of nodes $n = 10\,000$, mixing parameter $\mu = 0.75$, degree exponent $\tau_1 = 2$, community size exponent $\tau_2 = 3$, average degree $k_{\text{avg}} = 20$, maximum degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 10$, maximum community size $c_{\text{max}} = 100$.

(iii) *Label propagation method (LPM)* by Raghavan *et al.* [21]. This method simulates the spreading of labels based on the simple rule that at each iteration a given node takes the most frequent label in its neighborhood.

We also run tests with the well-known INFOMAP algorithm [22], but this method has a great performance on LFR benchmark graphs [15], so the consensus procedure can lead just to a modest improvement and we do not show it here. We have however verified that our technique delivers higher quality results than the input partitions for INFOMAP as well.

Before showing the performance of our method we would like to discuss two issues. In Fig. 1 we show that triadic closure helps to improve the accuracy of the results. The test was carried out by using the Louvain algorithm on LFR benchmark graphs with 10 000 nodes. The value of the mixing parameter μ was set to 0.75, which is in the area where the performance starts degrading (see Fig. 3). The other parameters are given in the caption. On the x -axis we report the maximum number of triads that can be closed at each iteration, in multiples of the number of links m . When the number of triads exceeds about m , a plateau is reached, yielding a superior accuracy compared to the input partitions and *LF consensus*.

In Ref. [4] it was shown that consensus clustering leads to more stable partitions compared to the input ones. We want to check if this holds true for our method as well. This analysis is illustrated in Fig. 2, where we computed the average NMI scores between any two input and output partitions,

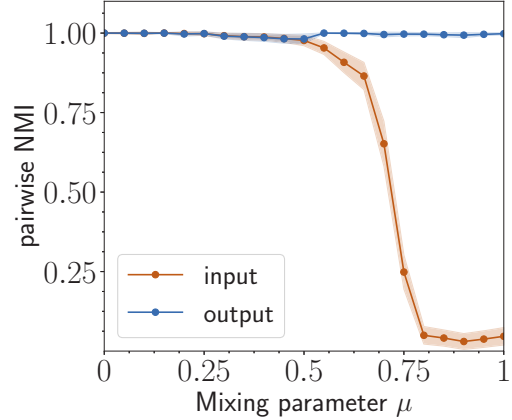


FIG. 2. Stability of consensus clustering partitions. Average value of the NMI for all pairs of partitions returned by our technique, compared with the corresponding average score for the input partitions. The thickness of the lines indicate the standard error of the average. The test uses the Louvain algorithm on LFR benchmarks. When communities are easy to find (low to intermediate μ -values), partitions are essentially identical in both cases and the pairwise NMI is equal to one. Interestingly, for our consensus clustering procedure the resulting partitions are very similar even when communities are harder to find, whereas the input partitions become less and less similar to each other. The parameters used to generate the LFR benchmark graphs are: number of nodes $n = 10\,000$, degree exponent $\tau_1 = 2$, community size exponent $\tau_2 = 3$, average degree $k_{\text{avg}} = 20$, maximum degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 10$, maximum community size $c_{\text{max}} = 100$.

respectively, for the Louvain method on LFR benchmarks. We see that for the consensus partitions the average is very close to one for any value of the mixing parameter μ , including those corresponding to the regime where the clusters are not detected. In contrast, the input partitions become progressively uncorrelated when communities become fuzzy and undetectable. We stress that the result is non-trivial here because we deal with a sparse consensus matrix through the whole calculation, which could introduce a significant amount of noise in the output compared to *LF consensus*, where the consensus matrix is dense and the community structure considerably enhanced.

In Fig. 3 we compare the accuracy of our method with *LF consensus*. For each method we identified the value of the threshold parameter τ that yields the best performance, which is 0.7 for CNM, 0.2 for Louvain, and 0.8 for LPM. We used these thresholds systematically. For Louvain and LPM our method has the same performance as *LF consensus* across all values of μ . For CNM our method does not perform as well as *LF consensus*, but its accuracy is still way above that of the CNM algorithm itself, especially on the larger graphs.

In general, our fast consensus clustering algorithm can be applied to large network sizes, which were hitherto out of reach for *LF consensus* as well as all consensus clustering techniques relying on the calculation of the full consensus matrix. In Fig. 4 we show the performance plot of the Louvain algorithm on LFR graphs with 100 000 nodes. As for the smaller network sizes, our technique outperforms the direct application of the clustering algorithm. The margin is small

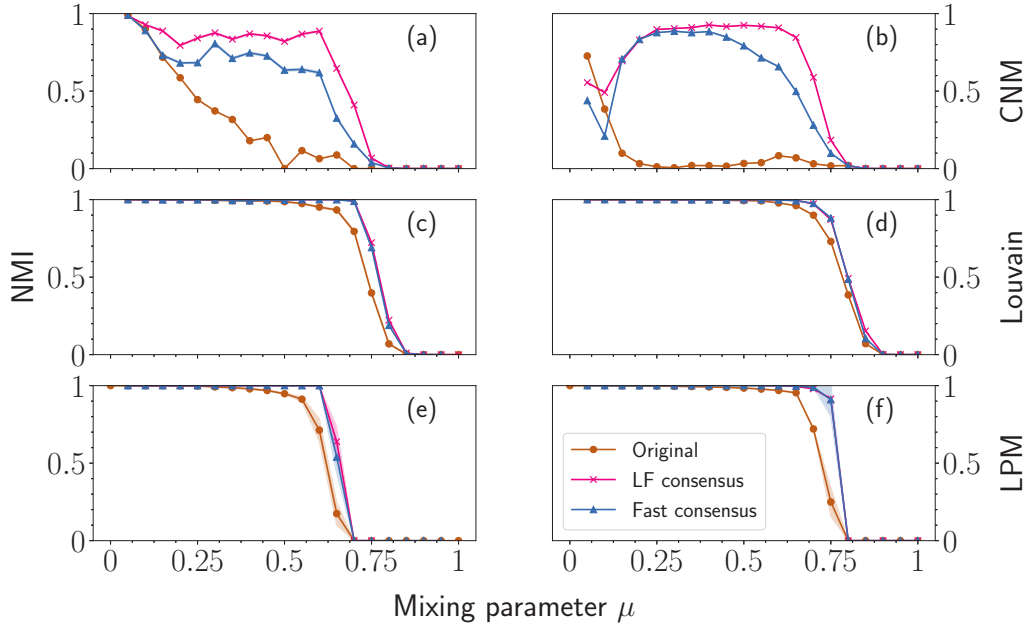


FIG. 3. Performance of fast consensus clustering on LFR benchmark graphs with 1000 (left) and 10 000 (right) nodes. The algorithms are fast greedy modularity maximization by Clauset, Newman, and Moore [CNM, (a) and (b)], Louvain [(c) and (d)], and the label propagation method [LPM, (e) and (f)]. The three curves in each plot correspond to the performance of the community detection method (brown), *LF consensus* (magenta), and fast consensus (blue). The thickness of the lines indicates the standard error of the mean. The parameters used to generate the LPM benchmark graphs are: degree exponent $\tau_1 = 2$, community size exponent $\tau_2 = 3$, average degree $k_{\text{avg}} = 20$, maximum degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 10$, maximum community size $c_{\text{max}} = 50$ for $n = 1000$ nodes and $c_{\text{max}} = 100$ for $n = 10\,000$ nodes. For each μ -value we generated 20 benchmark configurations, for the LPM and $\mu = 0.6$ (1000 nodes) and $\mu = 0.75$ (10 000 nodes) we used 100 configurations to reduce the error.

because the Louvain algorithm (first level communities, see earlier in Sec. III) has already a good performance on the LFR benchmark [15].

Figure 5 compares the time complexity of the consensus clustering procedure presented here with that of Ref. [4]. Calculations were executed on an iMac with a 3.2 GHz Intel

Core i5 processor. We have been able to run our technique on networks with 10^6 nodes in a few hours. Getting consensus partitions from *LF consensus* on networks of this size is impossible because of the high memory and time demands. The complexity of both methods scales as a power of the number of nodes of the network, with exponents 1.6

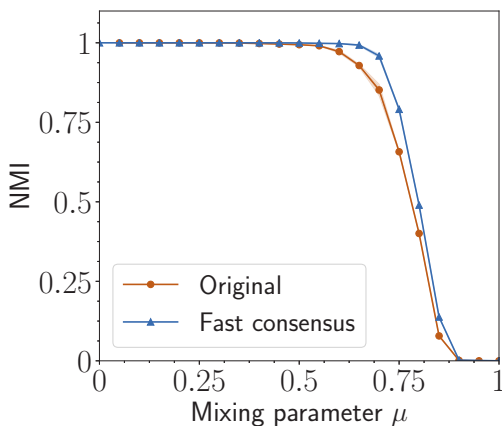


FIG. 4. Fast consensus clustering on large networks. Comparative analysis of the performance of the Louvain algorithm, with and without consensus clustering, on LFR benchmark graphs with 100 000 nodes. The thickness of the lines indicates the standard error of the mean. The other parameters to construct the graphs are: degree exponent $\tau_1 = 2$, community size exponent $\tau_2 = 3$, average degree $k_{\text{avg}} = 20$, maximum degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 10$, maximum community size $c_{\text{max}} = 1000$.

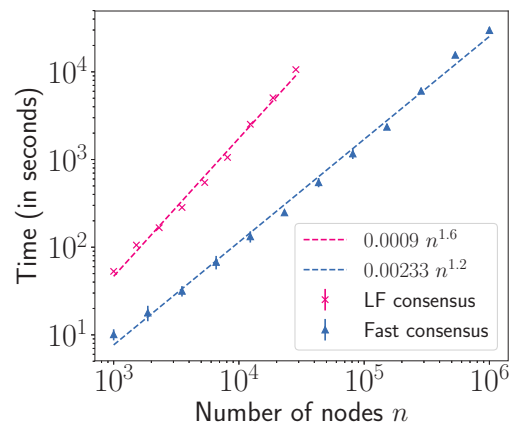


FIG. 5. Time complexity of *LF consensus* and our fast consensus clustering procedure. The clustering algorithm is Louvain, the tests were done on LFR benchmark graphs with the following parameters: degree exponent $\tau_1 = 2$, community size exponent $\tau_2 = 3$, average degree $k_{\text{avg}} = 20$, maximum degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 10$, maximum community size $c_{\text{max}} = n/10$ (n being the number of nodes of the graph). The mixing parameter μ is set to the value 0.3. The error bars indicate the standard deviation of the mean.

(*LF consensus*) and 1.2 (fast), respectively. Also, the prefactor for the complexity of our method is significantly smaller than for *LF consensus*. For our method, the complexity of the calculation of the consensus matrix is exactly linear, as we have seen (proportional to m , where $m \sim n$ if the network is sparse), and the number of iterations required to reach convergence increases very slowly with the network size. So the final complexity matches that of the clustering algorithm. Indeed, we have verified that the `igraph` implementation of the Louvain algorithm we have used has slightly superlinear complexity, in accord with the exponent 1.2. For *LF consensus*, instead, the complexity is dominated by the construction of the consensus matrix and it can get actually close to quadratic if the input partitions have large clusters, which severely constrains its applicability. Naturally, if the chosen clustering algorithm has complexity significantly larger than linear, there is no gain in using a fast consensus approach.

IV. CONCLUSIONS

We have devised an algorithm to implement a fast variant of the consensus clustering routine introduced in Ref. [4]. This procedure consists in sampling the consensus matrix, instead of computing all its elements, which could lead to a

worst-case quadratic space and time complexity. The elements of the consensus matrix which are actually computed are those corresponding to the pairs of neighbors of the network at study, plus at most as many pairs closing triangles with those. The performance remains the same or close to that of the full procedure in all cases we have examined, while the complexity becomes linear, which enables clustering analyses of large networks. The idea of sampling the consensus matrix can be easily integrated in other consensus clustering approaches [7–9]. We expect that also in those cases the accuracy will remain comparable as in the corresponding original methods. In particular deducting a random baseline from the elements of the consensus matrix, as it is done in the techniques of Refs. [7,9], might lead to better results than the simple procedure presented here.

ACKNOWLEDGMENTS

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, Saudi Arabia, under Grant No. RG-1439-311-10. The authors, therefore, acknowledge with thanks DSR for technical and financial support.

-
- [1] M. A. Porter, J.-P. Onnela, and P. J. Mucha, Communities in networks, *Not. Amer. Math. Soc.* **56**, 1082 (2009).
 - [2] S. Fortunato, Community detection in graphs, *Phys. Rep.* **486**, 75 (2010).
 - [3] S. Fortunato and D. Hric, Community detection in networks: A user guide, *Phys. Rep.* **659**, 1 (2016).
 - [4] A. Lancichinetti and S. Fortunato, Consensus clustering in complex networks, *Sci. Rep.* **2**, 336 (2012).
 - [5] P. Ronhovde and Z. Nussinov, Multiresolution community detection for megascale networks by information-based replica correlations, *Phys. Rev. E* **80**, 016109 (2009).
 - [6] O. Sporns, Structure and function of complex brain networks, *Dialogues in Clinical Neuroscience* **15**, 247 (2013).
 - [7] D. S. Bassett, M. A. Porter, N. F. Wymbs, S. T. Grafton, J. M. Carlson, and P. J. Mucha, Robust detection of dynamic community structure in networks, *Chaos* **23**, 013142 (2013).
 - [8] A. Tagarelli, A. Amelio, and F. Gullo, Ensemble-based community detection in multilayer networks, *Data Min. knowl. Discov.* **31**, 1506 (2017).
 - [9] L. GS Jeub, O. Sporns, and S. Fortunato, Multiresolution consensus clustering in networks, *Sci. Rep.* **8**, 3259 (2018).
 - [10] Valérie Poulin and François Théberge, Ensemble clustering for graphs, *International Workshop on Complex Networks and their Applications* (Springer, Berlin, 2018), p. 231.
 - [11] A. A. Hagberg, D. A. Schult, and P. J. Swart, Exploring network structure, dynamics, and function using networkx, in *Proceedings of the 7th Python in Science Conference*, edited by Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Pasadena, CA, USA, 2008), pp. 11–15.
 - [12] G. Csardi and T. Nepusz, The `igraph` software package for complex network research, *InterJournal, Complex Systems*, 1695 (2006).
 - [13] <http://github.com/adityat/fastconsensus>.
 - [14] A. Lancichinetti, S. Fortunato, and F. Radicchi, Benchmark graphs for testing community detection algorithms, *Phys. Rev. E* **78**, 046110 (2008).
 - [15] A. Lancichinetti and S. Fortunato, Community detection algorithms: A comparative analysis, *Phys. Rev. E* **80**, 056117 (2009).
 - [16] A. Lancichinetti, S. Fortunato, and J. Kertesz, Detecting the overlapping and hierarchical community structure in complex networks, *New J. Phys.* **11**, 033015 (2009).
 - [17] A. Clauset, M. E. J. Newman, and C. Moore, Finding community structure in very large networks, *Phys. Rev. E* **70**, 066111 (2004).
 - [18] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, *Phys. Rev. E* **69**, 026113 (2004).
 - [19] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, Fast unfolding of communities in large networks, *J. Stat. Mech.* (2008) P10008.
 - [20] S. Fortunato and M. Barthélemy, Resolution limit in community detection, *Proc. Natl. Acad. Sci. USA* **104**, 36 (2007).
 - [21] U. N. Raghavan, R. Albert, and S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, *Phys. Rev. E* **76**, 036106 (2007).
 - [22] M. Rosvall and C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, *Proc. Natl. Acad. Sci. USA* **105**, 1118 (2008).