Darst, Richard; Nussinov, Z.; Fortunato, Santo

Improving the performance of algorithms to find communities in networks

Published in:
Physical Review E

DOI:
10.1103/PhysRevE.89.032809

Published: 01/01/2014

Please cite the original version:
Improving the performance of algorithms to find communities in networks

Richard K. Darst,1 Zohar Nussinov,2 and Santo Fortunato1

1Department of Biomedical Engineering and Computational Science, Aalto University School of Science, P.O. Box 12200, FI-00076, Finland
2Physics Department, Washington University in St. Louis, CB 1105, One Brookings Drive, St. Louis, Missouri 63130-4899, USA

(Received 15 November 2013; revised manuscript received 20 January 2014; published 20 March 2014)

Most algorithms to detect communities in networks typically work without any information on the cluster structure to be found, as one has no a priori knowledge of it, in general. Not surprisingly, knowing some features of the unknown partition could help its identification, yielding an improvement of the performance of the method. Here we show that, if the number of clusters was known beforehand, standard methods, like modularity optimization, would considerably gain in accuracy, mitigating the severe resolution bias that undermines the reliability of the results of the original unconstrained version. The number of clusters can be inferred from the spectra of the recently introduced nonbacktracking and flow matrices, even in benchmark graphs with realistic community structure. The limit of such a two-step procedure is the overhead of the computation of the spectra.

DOI: 10.1103/PhysRevE.89.032809 PACS number(s): 89.75.Hc

I. INTRODUCTION

Community structure is one of the most important features of complex networks. Communities, or clusters, are subgraphs of a network with an unusually high density of edges, with respect to the average edge density of the network as a whole. Nodes in the same community tend to share the same attributes and/or roles within the network, and their identification might lead to the discovery of unknown features purely from topological inputs. This is why the quest for methods to detect communities in networks has become one of the hottest topics in network science [1,2].

In general, the only preliminary information available to any algorithm is the topology of the network, i.e., which pairs of nodes are connected to each other and which are not. Anything about the cluster structure to be found is unknown and is supposed to be given as output of the procedure. It would be valuable to have some information on the unknown partition of the network, as one could reduce considerably the huge space of possible partitions to explore, and increase the chance of successfully identifying the communities. In particular, if the number of clusters was known and could be given as an input or constraint, methods deliver more accurate partitions.

The problem is that it is difficult to have access to the number of clusters. Fortunately, recent work has shown that it is possible to derive this information from the spectra of the nonbacktracking matrix [3] and the flow matrix [4], at least on the classic version of the planted partition model [5], where clusters have identical size and nodes the same degree (on average). We show that the prediction of the number of clusters remains accurate as well on the Lancichinetti-Fortunato-Radicchi (LFR) benchmark graph [6], which extends the original planted partition model by introducing realistic features of community structure, i.e., heterogeneous distributions of degrees and cluster sizes. The prediction is more reliable the larger the graph size.

Therefore, we propose to improve the performance of community detection algorithms via a two-step approach: first one infers the number of clusters from the spectrum of the nonbacktracking or the flow matrix; then one uses this number as an additional input of the algorithm. We show that popular methods, like the optimization of the modularity by Newman and Girvan [7], become much more accurate this way. This is remarkable, as the direct optimization of modularity is known to have a limited resolution, which may prevent the method from identifying the correct scale of the communities, even when the latter is very pronounced [8,9]. Knowing the number of clusters, $q$, and constraining the optimization of the measure among partitions with fixed $q$, one can mitigate the problem.

The computation of the spectra is unfortunately quite heavy and represents the bottleneck of the procedure, making the analysis of large networks ($\gtrsim1 \times 10^6$ links) basically unfeasible. However, our results are encouraging and might stimulate the development of quicker approximate heuristics than the ones currently available.

In Sec. II we reveal the importance of knowing the number of clusters for the results of popular algorithms. Section III tests the reliability of the prediction of the number of clusters in the standard planted partition model and in the LFR benchmark graph. In Sec. IV we introduce a two-step approach consisting of a fast greedy optimization of the modularity by Newman and Girvan for partitions with a fixed number of clusters and we show the superiority of its performance with respect to the exhaustive modularity optimization via simulated annealing on the LFR benchmark. Our findings are summarized in Sec. V.

II. CONSTRAINED VERSUS UNCONSTRAINED COMMUNITY DETECTION

Stochastic block models are the best known and most exploited class of models of networks with community structure. A graph with $N$ nodes is divided into $q$ groups of equal size $n = N/q$, and the probability of nodes $i$ and $j$ to be linked is given by $p_{rs}$, where $r$ and $s$ are the groups of $i$ and $j$, respectively. The principle is that the linking probability of any two nodes only depends on the memberships of the nodes. Despite its simplicity, this model can generate a wide variety of scenarios. Here we focus on the simplest one, where $p_{rs} = p_{ss}, \forall r = s$ and $p_{rs} = p_{sa}, \forall r \neq s$. Here, if $p_{mm} > p_{sa}$, the expected number of neighbors of a node within its group exceeds the expected number of neighbors of the node in each of the remaining $q - 1$ groups, so the groups are communities...
According to the general intuition. This version coincides with the planted partition model by Condon and Karp [5] and has generated popular benchmark graphs that are regularly used to test the performance of community detection techniques, like the four-groups test [1] and the LFR benchmark [6].

According to the model, clusters are present and should be detectable so long as $p_{in} > p_{out}$. However, recent works [10–14] have pointed out that, in the limit of sparse graphs, i.e., of networks of infinite size but finite average degree, random fluctuations make the detectability of clusters possible only for $p_{in} > p_{out} + \Delta$, where $\Delta$ is a function of the parameters $n$, $q$, $p_{in}$, and $p_{out}$. So it is necessary for the clusters to have an internal edge density higher by some finite amount than the external one(s) to overcome the noise coming from the inherent stochasticity of the network’s construction. By introducing the average internal and external degree of a node in a cluster, $\mu_{in} = (n-1)p_{in}$ and $\mu_{out} = np_{out}$, respectively, the formula of the detectability limit derived in Refs. [11,13] reads

\[
\mu_{in} - \mu_{out} = \sqrt{\mu_{in} + (q-1)\mu_{out}}. \tag{1}
\]

Typically, one fixes the value of the internal average degree $\mu_{in}$ and obtains the limit value of $\mu_{out}$ from Eq. (1), beyond which no method can do better than a random guess, which yields $1/q$, where $q$ is the number of clusters.

In Fig. 1 we show the performance of several popular community detection algorithms on such model, in the simple case of two clusters ($q = 2$). Each panel corresponds to a different value of the cluster size $n$, ranging from 50 to 1000. The average internal degree $\mu_{in}$ is fixed to 10. Given these parameters, Eq. (1) yields a value $\mu_{out}^\text{det} = 6$ for the detectability limit (dashed vertical line), regardless of the cluster size $n$, and we vary $\mu_{out}$ from 0 to $\mu_{out}^\text{det}$. The dot-dashed line indicates the limit performance of spectral modularity optimization, analytically derived in Ref. [13]. Performance is computed as the fraction of correctly detected nodes [13]. If all nodes are correctly assigned to their planted clusters, the fraction of correctly detected nodes is 1. As long as clusters are detectable, it stays above the value corresponding to random assignment (here $1/2$, marked by the horizontal dashed line).

The algorithms we adopt are modularity optimization via simulated annealing (Mod) [15], the absolute Potts model (APM) [16], the order statistics local optimization method (OSLOM) [17], and Infomap [18]. Further information on all methods is available in the Appendix. For Mod and APM we performed two types of runs, one without any indication on the number of clusters, $q$, of the planted partition, the other by constraining the optimization procedure to partitions with $q$ clusters (Mod + $q$ and APM + $q$). Figure 1 shows that in general the performance of all methods is quite far from optimal. However, the limit performance (with slight variations due to finite size effects) is attained when the number of clusters, $q$, of the planted partition is known. In particular, we remark that modularity maximization, in the absence of

![FIG. 1](image1.png)

**FIG. 1.** (Color online) Fraction of correctly detected nodes for the planted partition model with $q = 2$ clusters, average internal degree $\mu_{in} = 10$, and four values for the size of the clusters: (a) $n = 50$, (b) $n = 200$, (c) $n = 500$, and (d) $n = 1000$. Symbols indicate the performance curves of different popular methods of community detection. For modularity optimization (Mod) and the absolute Potts model (APM) we show two curves, referring to the results of the method in the absence of any information on the number of clusters of the planted partition, and when such information is fed into the model as initial input. In both cases, knowing the number of clusters beforehand leads to a much better performance. Modularity’s limit performance (dot-dashed curve), as estimated by Nadakuditi and Newman [13], is attained only when the number of clusters is known. Otherwise modularity optimization performs rather poorly, as expected. The detectability limit is shown as a vertical line at $\mu_{out} = 6$.

![FIG. 2](image2.png)

**FIG. 2.** (Color online) Fraction of correctly detected nodes for the planted partition model with $n = 200$ nodes per cluster, average internal degree $\mu_{in} = 10$, and four values for the number of clusters: (a) $q = 4$, (b) $q = 6$, (c) $q = 8$, and (d) $q = 10$. Symbols indicate the performance curves of different popular methods of community detection. For modularity optimization (Mod) and the absolute Potts model (APM) we show two curves, referring to the results of the method in the absence of any information on the number of clusters of the planted partition, and when such information is fed into the model as initial input. In both cases, knowing the number of clusters beforehand leads to a much better performance. The detectability limit is shown as a vertical line which varies as a function of $q$, according to Eq. (1). The horizontal dashed line indicates the fraction $1/q$ of nodes correctly detected by a random partition of the graph in $q$ equal-sized clusters.
any information on the number of clusters, does a rather poor job. The limit curve and in general the whole analysis by Nadakuditi and Newman assumes that one knows the number of clusters beforehand, which is a very strong (and generally invalid) hypothesis. Furthermore, we observe that, once the number of clusters is fixed to the correct value (in Mod + q and APM + q), both methods return the same results. Practically speaking, the only thing that a method can then do is move nodes to the cluster to which it has the most edges. The result is identical performance for any constant-q method in this simple symmetric case. In Fig. 2, we see that knowledge of q remains important as the number of clusters increases.

III. INFERRING THE NUMBER OF CLUSTERS FROM SPECTRA OF GRAPH MATRICES

From Sec. II we conclude that knowing the number of clusters beforehand could push (some) methods up to the best attainable performance. Indeed, methods mostly fail in that they generally find a partition with a different number of clusters than the planted one. This is shown in Fig. 3, where we show the average number of clusters detected by the methods we used on the graphs of Fig. 1. For illustration purposes, we show the variable \(|\Delta q/q| + 1\), where \(\Delta q\) is the difference between the number of detected clusters and the actual number q. If \(q_{\text{det}} = q\), \(\Delta q = q_{\text{det}} - q = 0\) and \(\Delta q/q + 1 = 1\). Naturally, Mod + q and APM + q yield the correct number of clusters by default, for any \(\mu_{\text{out}}\), since q is given as input. However, if the number of clusters is to be inferred by the method, it may differ from the actual q at \(\mu_{\text{out}}\) far below the detectability limit. In fact, for the largest graph sizes, deviations are large for very small values of \(\mu_{\text{out}}\), when the clusters of the planted partition are connected by very few links and should easily be distinguishable. In particular, we notice that Infomap, which is known to have an excellent performance on the more realistic LFR benchmark [19], breaks the two clusters in many pieces, so it is not a valuable predictor of q for this graph model.

On the other hand, it was recently shown that the correct number of clusters in the planted partition model can be correctly inferred from the spectra of two matrices: the nonbacktracking matrix B [3] and the flow matrix F [4]. Both are \(2m \times 2m\) matrices, where m is the number of links of the network. Each link is considered in both directions, yielding \(2m\) directed links and indicated with the notation \(i \rightarrow j\), meaning that the link goes from node i to node j. Their elements read

\[
B_{i \rightarrow j, k \rightarrow l} = \delta_{ij}(1 - \delta_{jk})
\]

and

\[
F_{i \rightarrow j, k \rightarrow l} = \frac{\delta_{ij}(1 - \delta_{jk})}{d_i - 1}.
\]

In Eq. (3), \(d_i\) indicates the degree of node i. So the elements of F are basically the elements of B, normalized by node degrees. This is done to account for the heterogeneous degree distributions observed in most real networks. Both matrices have nonzero elements only for each pair of links forming a directed path from the first node of one link to the second of the other link. To do that, links have to be incident at one node. As a matter of fact, the nonbacktracking matrix B is just the adjacency matrix of the (directed) links of the graph.

A remarkable property of both matrices is that most eigenvalues, which are generally complex, are enclosed by a circle centered at the origin and that the number of eigenvalues lying outside of the circle is a good proxy of the number of communities of the network. For B the circle’s radius is given by the square root \(\sqrt{c}\) of the leading eigenvalue c; for F it equals \(\sqrt{[d/(d-1)]/[d]}\), which is never greater than 1. In Refs. [3,4] it was shown that the eigenvectors can be used for the detection of the communities, by turning the nodes into points in a \((q-1)\)-dimensional Euclidean space (q being the number of off-circle eigenvalues, i.e., of clusters) and grouping them with partitional clustering techniques, like K means [20].

In Fig. 3 we added the predictions of the number of clusters obtained by both matrices (labeled by NBW and Flow). The exact techniques used to compute the spectra are listed in the Appendix. We find that the prediction is much more accurate than those of all community detection methods we used, and it becomes more precise, the larger the graph size, as expected. There is no difference between the two curves, because the degree of all nodes are essentially identical in the planted partition model, so the two matrices are basically proportional to one another and their spectra approximately coincide.

To see whether one can rely on the prediction of the number of clusters from the spectra of B and F, one should tackle more complex network models with communities. The classic version of the planted partition model, used here as well as in Refs. [3,4], is too homogeneous to resemble any real network.
Those graphs, and this is reflected in Fig. 4, as it gives the best
of Ref. [19]. Infomap was the best-performing method on
the comparative analysis of community detection algorithms
also remark that Mod fails to guess
the degree distribution, but Infomap is clearly superior. We
prediction of the number of clusters. Here the two matrices
The average number of clusters is
⟨μ⟩ = \frac{|Δq|}{q} + 1
The mixing parameter μ is the ratio of the external degree of
each node (with respect to the cluster it belongs to) by the total
degree of the node, so it varies from 0 to 1: values near zero
correspond to well-separated clusters, whereas values near 1
indicate a system with very mixed communities (hence hard
to identify). The other parameters of the LFR benchmark are
average degree, 20; maximum degree, 50; degree distribution
exponent, −2; and community size distribution exponent, −1.
The average number of clusters is \langle q \rangle = 40.534 (1000, S),
\langle q \rangle = 19.901 (1000, B), \langle q \rangle = 203.239 (5000, S), and \langle q \rangle =
100.879 (5000, B). These parameters are the same used for the
comparative analysis of community detection algorithms
of Ref. [19]. Infomap was the best-performing method on
those graphs, and this is reflected in Fig. 4, as it gives the best
detection of the number of clusters. Here the two matrices
yield close but distinct results, due to the inhomogeneity of
the degree distribution, but Infomap is clearly superior. We
also remark that Mod fails to guess q also for very low values of
μ, due to the resolution limit [8]. So it seems that we do not
gain that much by using B and F.

However, it may be that the situation improves on larger
systems. Therefore, we repeated the procedure on two other
sets of LFR graphs, with 10 000 and 20 000 nodes, respectively.
We extended the range of cluster sizes to the interval [10:1000],
so that it spans two orders of magnitude and there is a big
difference between the smallest and the largest community.
All other graph parameters are the same as above. Here the
average number of clusters is \langle q \rangle = 46.60 (10 000), \langle q \rangle =
95.66 (20 000). In Fig. 5 we show that in these larger and
more heterogeneous graphs both the nonbacktracking and the
flow matrices give a better prediction of q than Infomap.

IV. A TWO-STEP APPROACH
The results of Sec. III suggest that one might considerably
improve the performance of community detection algorithms
with a two-step procedure, in that one first infers the number
of clusters through the spectrum of the nonbacktracking or
the flow matrix and then runs the algorithm on the space
of partitions with that given number of clusters. Even if the
inference of the correct number of clusters is not 100% reliable,
especially if the system is not too large, one might still get
much closer to the true partition than by running the method
without any information on the number of clusters.
To show that, we designed such a two-step procedure for
modularity optimization. The number of clusters is deduced
through the flow matrix. Then, modularity is optimized via
a greedy procedure, which exploits the idea of the Louvain
algorithm [21]. The latter works in a hierarchical manner. First,
each node is its own cluster. Then nodes are put in the clusters
of their neighbors such that they yield the largest increase
of modularity. This gives the first hierarchical level. Then
such groups are turned into supernodes and the procedure is
repeated, which means that the clusters of the first hierarchical
level are combined into larger groups, yielding the second
hierarchical level, and so on, until one reaches the partition
with largest modularity. Naturally, the number of clusters
decreases when one moves from one level or partition to the
next. We stop at the level L such that the number of clusters, q_L,
is larger than the target number q but the number of clusters,
q_{L+1}, of the level L + 1 is smaller than q. If by any chance
there is a level L such that q_L = q we move to the refinement
step described below. If there is no level such that q_L \geq q,
the algorithm is restarted with a different seed until such a
partition is found. For the time being, we assume to be in the
more frequent case in which we start with a partition with
number of clusters larger than q.
We then perform the following cleanup procedure in order to correct the number of communities, decreasing it to the correct value $q$:

1. Of all pairs of communities, choose the two which, when merged, give the greatest increase (least decrease) in the network modularity and merge them.

2. Repeat step 1 until the number of communities equals $q$.

3. Iterate through all nodes in random order. For each node, move it to the community which most increases modularity, or leave it in its present community if modularity cannot be increased.

4. Repeat step 3 until there are no further changes to be made.

The last two steps are a refinement procedure aiming at further improving the modularity without changing the number of clusters and are identical to the steps of our initial greedy procedure. In Fig. 6 we show the performance of this procedure on LFR benchmark graphs with the same parameters as the ones in Fig. 4 and those used in the comparative analysis of Ref. [19]. As a reference, we report the performance curve of the exhaustive maximization of modularity via simulated annealing. We used simulated annealing because we wanted to get a very good estimate of the actual modularity maximum. Here the performance is expressed by the normalized mutual information (NMI) [22], which can also compute the similarity of covers, i.e., of partitions of the network into overlapping communities. This version is used consistently throughout the comparative analysis of Ref. [19]. The two-step procedure introduced here outperforms the unconstrained exhaustive optimization of modularity, with the performance boost increasing for larger graphs. In Table I we show that the two-step algorithm has comparable complexity as simulated annealing.

### Table I. Comparison of the time complexity of several algorithms used in this work. Time is indicated in seconds. We took four different values for the graph size $N$ (1000, 5000, 10000, and 20000); all other parameters of the LFR benchmarks are the same used for Fig. 4. The actual computation time depends on the system parameters, not only on the size $N$. Spectral: Time to calculate the largest $q$ eigenvalues of the flow matrix. This time is far larger than the remaining constrained modularity optimization routine, so it is representative of the complexity of the full calculation. Computations were performed using ARPACK, which uses the implicitly restarted Arnoldi method for sparse matrix iterative eigenvalue computation. The complexity scales approximately as $t/s \sim N^{2.3}$. Mod: Time to maximize modularity using simulated annealing with the parameters given in the Appendix. Infomap: Time to run the Infomap algorithm. All computations were performed on an Intel(R) Core(TM)2 Quad CPU (2.83 GHz) processor.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Spectral</th>
<th>Mod</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5 ± 1</td>
<td>30 ± 1</td>
<td>1 ± 0.5</td>
</tr>
<tr>
<td>5000</td>
<td>250 ± 10</td>
<td>250 ± 25</td>
<td>3 ± 1</td>
</tr>
<tr>
<td>10 000</td>
<td>1200 ± 100</td>
<td>1000 ± 100</td>
<td>8 ± 1</td>
</tr>
<tr>
<td>20 000</td>
<td>4700 ± 400</td>
<td>4500 ± 500</td>
<td>18 ± 1</td>
</tr>
</tbody>
</table>

FIG. 6. (Color online) Modularity optimization on the LFR benchmark. The graphs are generated using the same parameters as for those in Figs. 4(a)–4(d), respectively. Hexagons indicate modularity optimization via simulated annealing, without any constraint on the number of clusters. The green triangles show the performance of our two-step procedure, where the number of clusters, $q$, is inferred by the flow matrix and modularity is optimized with a greedy procedure over the set of partitions with $q$ clusters. The improvement of the performance is manifest, especially on the larger graphs.

V. SUMMARY

We have seen that the number of clusters is an important input for community detection algorithms. If it is known beforehand, one can push up the performance of methods. We have shown that the spectra of the nonbacktracking and flow matrices allow a reliable estimate of the number of clusters, both on the classic planted partition model and on the more realistic LFR benchmark, if the networks are not too small. Therefore, a two-step procedure, when one first derives the number of clusters and then performs a constrained run of the method one wishes to use, might lead to much better results. We have shown this for modularity optimization, whose unconstrained version preferentially leads to clusters of a given scale, which may or may not have anything to do with the actual scale of the clusters of the system at study. Instead, once the number of clusters is given, and the optimization is constrained on the set of partitions with that number of clusters, the method gives much better results.

In general, since the prediction of the number of clusters might not be correct, one could constrain the method on the set of partitions with number of clusters in a small range centered at the predicted value. This way, even if the actual value is missed but close, it can still be recovered by the procedure. It would be already valuable to “push” a method to the interesting range of values, instead of letting it free to be attracted somewhere else by intrinsic biases, as it happens for modularity.

On the practical side, computing the eigenvalues of the nonbacktracking or the flow matrix is lengthy. Both are $2m \times 2m$ matrices. The adjacency matrix $A$ has $N \times N$ elements, so $B$ and $F$ are larger by a factor of $(d)^2$, where $(d)$ is the average degree of the network. Krzakala et al.
showed that the spectrum of $B$ can be computed by working on a $2N \times 2N$ matrix [3], which reduces the complexity of the calculation by the factor $(d)^2$. Still, an approximate but reliable computation of the spectra requires a time which scales approximately quadratically with the network size, by using standard software libraries (see the Appendix). This is because the time needed to compute a single eigenvector or eigenvalue is linear in the graph size, but the number of clusters, $q$, is proportional to $N$, so the complexity of the calculation of $q$ eigenvectors or eigenvalues is approximately quadratic. Consequently, the problem is intractable for graphs with a number of links of the order of millions or higher. This is the real bottleneck of the calculation. On the other hand, many systems of interest remain within reach. Besides, one could find other good predictors of the number of clusters which do not require lengthy calculations. In the example we have seen of the LFR benchmarks, Infomap is very good at guessing the right number of clusters and it runs much faster than the computation of the spectra of $B$ and $F$. However, we have seen that it gives poor results on the classic planted partition model. Even if the latter is not a good model of real networks with community structure, it means that we should extensively test the reliability of a method before using it in applications. This paper shows that it might be a very good investment.

ACKNOWLEDGMENTS

We thank Paolo De Los Rios, Lucio Floretta, Chris Moore, Lenka Zdeborova, and Pan Zhang for useful exchanges. R.K.D. and S.F. gratefully acknowledge MULTIPLEX, Grant No. 317532 of the European Commission.

APPENDIX: METHODS

In this section, we describe the community detection methods used in this work and their respective parameters.

1. Modularity

Network modularity is a measure which rates the quality of a network partition, with higher modularity presumably indicating better communities. Modularity compares the actual number of intracommunity links to the expected number of links in a random graph with the same degree distribution. The details of modularity optimization have been extensively described in other work, for example [24]. We adopt a simulated annealing approach in the spirit of Ref. [25]. The Hamiltonian formulation of modularity reads

$$H(\sigma) = -\frac{1}{2m} \sum_{i<j} (A_{ij} - \frac{k_i k_j}{2m}) \delta(\sigma_i, \sigma_j), \quad \text{(A1)}$$

with $A_{ij} = 1$ if there is an edge between nodes $i$ and $j$ and zero otherwise; $k_i$, $k_j$ the degrees of nodes $i$ and $j$, respectively; $m$ the total number of (undirected) edges of the graph; $\sigma_i$, $\sigma_j$ the respective community of nodes $i$ and $j$; and $\delta(i,j)$ the Kronecker $\delta$ function. A variety of trial moves are applied, and accepted according to the Metropolis criteria [26]:

(a) **Shift moves:** A node may be shifted from one community to another (possibly empty) community. This may possibly result in the number of communities decreasing or increasing if the node is moved to a new community or was previously in a singleton community.

(b) **Merge moves:** Two randomly selected communities are selected and merged.

(c) **Split moves:** A community is chosen at random and split. When splitting, each node has an equal and independent probability of 1/2 to join the new community being formed by the split. A more sophisticated method could run community detection within the community in order to determine the optimal community split.

**Mod.** For regular modularity calculations, first all nodes are assigned to singleton communities (every node in its own distinct community), and the Hamiltonian or energy is calculated from Eq. (A1). Then, for each round, we do one trial move of each node into a different random community ($N$ moves). Then, there is one merge move between two random communities, and one split move of a random community. After every round, the network modularity is calculated and the configuration is stored if the current partition is better than the previously optimal partition. We choose an initial temperature $T_0 = 1/10$, and after each round, temperature is updated via $T_{\text{new}} = 0.999 T_{\text{old}}$. Trials continue until the energy does not decrease by more than $1 \times 10^{-10}$ for 1000 rounds. This entire procedure is repeated five times for each graph, and the lowest energy result of all trials is selected as the final partition. We achieved 100 averages for all points with these parameters.

**Mod + q.** This procedure, used in Figs. 1–3, uses simulated annealing with an additional term and different parameters to fix the number of clusters to $q_0$, the true number of communities. The Hamiltonian (note different normalization) is

$$H(\sigma) = -\sum_{i<j} (A_{ij} - \frac{k_i k_j}{2m}) \delta(\sigma_i, \sigma_j) + E_q, \quad \text{(A2)}$$

with

$$E_q = \epsilon (q - q_0)^2 \quad \text{(A3)}$$

being the term to fix $q$. We choose $\epsilon = 1$. Thus, as temperature decreases, the number of clusters becomes energetically fixed at the true number. The initial configuration is set with all nodes divided randomly among $q_0$ clusters, though $q$ may fluctuate during the minimization. Each round consists of $N$ moves, with $p_{\text{split}} = 0.95$, $p_{\text{merge}} = 0.025$, and $p_{\text{split}} = 0.025$ for each move. We choose an initial temperature $T_0 = 100$, and after each round, temperature is decreased by a factor $T_{\text{new}} = T_{\text{old}}/1.001$. Trial rounds continue until there are no more changes accepted for 10 rounds. This procedure is repeated 10 times per graph, with the lowest final energy being taken as the final partition. We achieved 100 averaging runs for all points with these parameters.

2. Absolute Potts model (APM)

The APM detects communities via minimization of a Hamiltonian without a null model,

$$H(\sigma) = -\sum_{i<j} (A_{ij} - \gamma B_{ij}) \delta(\sigma_i, \sigma_j), \quad \text{(A4)}$$

where $B_{ij} = 1 - A_{ij}$ is the inverse adjacency matrix. In addition, $\gamma$ is a resolution parameter which can be varied across
a range of values in order to determine the optimal community size. The optimal $\gamma$ is inferred by the self-similarity of multiple greedy trial minimizations performed on the same graph. We use the classic normalized mutual information [22] to select the optimal $\gamma$ and in general follow the procedure of Ref. [27]. We achieved 100 averaging runs for all points with these parameters. 

$APM + q$: For constant-$q$ APM runs, we do not use the multiresolution procedure of the previous paragraph. Instead, we fix $\gamma = 10$ and use the constant-$q$ simulated annealing procedure described under Mod $+ q$ above, but with the APM Hamiltonian plus $E_q$ term. We achieved 100 averaging runs for all points with these parameters.

3. OSLOM

OSLOM seeks statistically significant clusters in networks. We use the oslom code available at http://oslom.org/. All default parameters are accepted, but in addition we enable the option to ensure that all nodes are a member of at least one community (no homeless nodes, the default in the latest released code), which is sensible for the case where all nodes are contained in at least one community. The default of 10 trial optimizations of the lowest hierarchical level is used, and the lowest hierarchical level is selected as the resulting partition. All OSLOM data are averaged over 100 graph realizations unless otherwise indicated.

4. Infomap

Infomap is a method based on the compression of a random walk on graphs. The source code can be downloaded at http://mapequation.org/. All default options are accepted, but notably we restrict the algorithm to a single nonhierarchical partition ($−20 −1\text{eval}$) and request 10 trial minimizations for each graph. All Infomap data are averaged over 100 graph realizations unless otherwise indicated.

5. Spectral methods

For a description of the nonbacktracking walk and flow matrices, see Sec. III. We use a home-built implementation of these methods, as described in Refs. [3,4]. Actual eigenvalue computation is via the SCIPY sparse linear algebra package [28], which is an interface to the ARPACK library [29], which itself uses the implicitly restarted Arnoldi method for computing $k$ eigenvalues [30]. In order to infer that a graph has $q$ communities, we must compute at least $q + 1$ eigenvalues ($q$ eigenvalues outside of the circle, and one inside of the circle to know that the calculation is finished). The time taken to compute $q + 1$ eigenvalues scales as $t/s \sim N^{2.3}$, as $q$ is generally a function of $N$ (approximately linear). All spectral data are averaged over 100 graph realizations unless otherwise indicated.