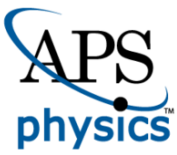

Multiway spectral community detection in networks

Xiao Zhang and M. E. J. Newman

Phys. Rev. E **92**, 052808 — Published 19 November 2015

DOI: 10.1103/PhysRevE.92.052808

Multiway spectral community detection in networks

Xiao Zhang[1] and M. E. J. Newman[1, 2]

[1]*Department of Physics, University of Michigan, Ann Arbor, Michigan, USA*
[2]*Center for the Study of Complex Systems, University of Michigan, Ann Arbor, Michigan, USA*

One of the most widely used methods for community detection in networks is the maximization of the quality function known as modularity. Of the many maximization techniques that have been used in this context, some of the most conceptually attractive are the spectral methods, which are based on the eigenvectors of the modularity matrix. Spectral algorithms have, however, been limited by and large to the division of networks into only two or three communities, with divisions into more than three being achieved by repeated two-way division. Here we present a spectral algorithm that can directly divide a network into any number of communities. The algorithm makes use of a mapping from modularity maximization to a vector partitioning problem, combined with a fast heuristic for vector partitioning. We compare the performance of this spectral algorithm with previous approaches and find it to give superior results, particularly in cases where community sizes are unbalanced. We also give demonstrative applications of the algorithm to two real-world networks and find that it produces results in good agreement with expectations for the networks studied.

I. INTRODUCTION

Community detection, the division of the vertices of a network into groups such that connections are dense within groups and sparser between them, has been a topic of vigorous research, particularly within statistical physics, for some years [1]. A broad range of different approaches to the problem have been tried, but perhaps those in widest current use are methods based on modularity maximization. Modularity [2] is a scalar objective function which assigns a numerical score to any division of a network into communities, with higher scores being associated with divisions that are better in the sense of having more edges within communities and fewer between them. Modularity maximization detects communities by finding the divisions that have the highest modularity scores. Unfortunately, the exhaustive numerical maximization of modularity over all divisions of a network is known to be an NP-hard task [3], computationally tractable only for the very smallest of networks, so we are forced to rely on approximate optimization heuristics, a large number of which have been tried. These include greedy algorithms [4, 5], simulated annealing [6, 7], extremal optimization [8], genetic algorithms [9], and the widely used multiscale "Louvain method" of Blondel *et al.* [10], which has been incorporated into a number of common software packages.

In this paper we focus on another class of algorithms for modularity maximization, the spectral algorithms, which are based on the examination of the leading eigenvalues and eigenvectors of the so-called modularity matrix [11]. These methods are of interest for a number of reasons. First, they give high-quality results in practical situations while also being fast, the eigenvalues and vectors normally being calculated using the Lanczos method [12], which is highly efficient for the sparse matrices that arise in typical network problems. Second, they are conceptually attractive, being based on well-understood principles of linear algebra. And third, they are amenable to formal analysis, for instance using random matrix theory [13], allowing one to make precise statements about their performance.

Spectral methods, however, do have their problems. A primary one is that it is difficult to construct a simple and principled spectral algorithm for dividing a network into an arbitrary number of communities. Good algorithms exist for two- and three-way divisions, and repeated two-way divisions can sometimes produce good multiway divisions, but sometimes not [11, 14]. One possible approach for direct multiway community detection has been proposed by Gong *et al.* [15] and makes use of several leading eigenvectors of the modularity matrix simultaneously. This method, however, divides the network into $2k$ communities if k eigenvectors are used, which cannot be optimal since it is known that the optimal division has at most $k + 1$ communities [16]. Another approach that also uses several leading eigenvectors is to represent the leading eigenvectors as points in a high-dimensional space, then divide up those points using a standard data clustering method—the most popular choice is k-means clustering, as employed for instance in [17]. This method, which is analogous to previous algorithms for the different but related problem of Laplacian spectral graph partitioning [18, 19], is attractive for its simplicity and ease of implementation. On the other hand, while the strong similarity between graph partitioning and modularity maximization [16, 20] makes it natural to think that k-means would work in this situation, it is not clear what quantity, if any, the k-means approach is optimizing. In particular, the algorithm is not derived as an approximation to modularity maximization, so there are no formal guarantees that it will indeed maximize modularity, and in practice, as we show in this paper, there are situations where it can fail badly.

In this paper we introduce a different method for single-step, multiway, spectral community detection. Our method is not a generalization of the previous two-way method, which is based on a relaxation of the dis-

crete modularity optimization problem to a continuous optimization that can be solved by differentiation. Instead the method is based on the observation, made previously in [16], that modularity maximization is equivalent to a max-sum vector partitioning problem. (A similar equivalence for the graph partitioning problem was explored in [21, 22].) We propose a simple heuristic for the rapid solution of vector partitioning problems and apply it to the task in hand to create an efficient multiway community detection algorithm.

II. SPECTRAL COMMUNITY DETECTION AND VECTOR PARTITIONING

The modularity Q is a score assigned to a given division into any number of communities of a given network, such that "good" divisions—those in which most edges fall within communities and few edges fall between them—get a high score and "bad" divisions a low one. Formally, the modularity is equal to the fraction of edges that fall within communities minus the expected fraction if edges were placed at random [2]. Consider an undirected, unweighted network of n vertices and define an adjacency matrix $\mathbf{A}$ to represent the network structure, with elements $A_{ij} = 1$ if vertices i and j are connected by an edge and 0 otherwise. Now consider a division of the vertices of this network into k non-overlapping groups, labeled by integers $1 \dots k$, and define g_i to be the label of the group to which vertex i belongs. Then the modularity is given by [5]

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta_{g_i,g_j}, \tag{1}$$

where d_i is the degree of vertex i, m is the total number of edges in the network, and δ_{st} is the Kronecker delta. The modularity may be either positive or negative (or zero), with a maximum value of $+1$. Positive values indicate that the number of edges within groups is greater than one would expect by chance, and large positive values are considered indicative of strong community structure.

For convenience we also define the modularity matrix to be the symmetric $n \times n$ matrix $\mathbf{B}$ with elements

$$B_{ij} = A_{ij} - \frac{d_i d_j}{2m}, \tag{2}$$

in terms of which the modularity (1) can be written

$$Q = \frac{1}{2m} \sum_{ij} B_{ij} \delta_{g_i,g_j}. \tag{3}$$

Given that $\sum_i A_{ij} = d_j$ and $\sum_i d_i = 2m$, every row and column of the modularity matrix must sum to zero:

$$\sum_i B_{ij} = \sum_i A_{ij} - \sum_i \frac{d_i d_j}{2m} = 0, \tag{4}$$

which implies that the uniform vector $\mathbf{1} = (1, 1, 1, \dots)$ is an eigenvector of the modularity matrix with eigenvalue zero, a result that will be important shortly.

Now consider the problem of dividing a network with n vertices into k communities. If good divisions have high modularity scores and bad divisions low scores, we can find good divisions by maximizing modularity over divisions. Exact maximization is known to be very slow [3], so we turn instead to approximate methods. Following [16, 22], we note that the delta function in Eq. (3) can be written as

$$\delta_{g_i,g_j} = \sum_{s=1}^{k} \delta_{s,g_i} \delta_{s,g_j}, \tag{5}$$

and since the modularity matrix is symmetric it can always be written as an eigenvector decomposition

$$B_{ij} = \sum_{l=1}^{n} \lambda_l U_{il} U_{jl}, \tag{6}$$

where λ_l is an eigenvalue of $\mathbf{B}$ and U_{il} is an element of the orthogonal matrix $\mathbf{U}$ whose columns are the corresponding eigenvectors. Without loss of generality, we will assume that the eigenvalues are numbered in decreasing order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Combining Eqs. (3), (5), and (6), we now have

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{ij} \sum_{l=1}^{n} \lambda_l U_{il} U_{jl} \sum_s \delta_{s,g_i} \delta_{s,g_j} \\ &= \frac{1}{2m} \sum_{l=1}^{n} \lambda_l \sum_s \left[\sum_i U_{il} \delta_{s,g_i} \right]^2. \end{aligned} \tag{7}$$

We observe that (apart from the uninteresting leading constant) this is a sum over eigenvalues λ_l times the nonnegative quantities $\sum_r \left[\sum_s U_{il} \delta_{s,g_i} \right]^2$, so the largest (most positive) contributions to the modularity are typically made by the terms corresponding to the most positive eigenvalues. A standard approximation, used in essentially all spectral algorithms, is, instead of maximizing the entire sum, to maximize only these largest terms, neglecting the others. That is, we approximate the modularity by

$$Q = \frac{1}{2m} \sum_{l=1}^{p} \lambda_l \sum_s \left[\sum_i U_{il} \delta_{s,g_i} \right]^2. \tag{8}$$

for some integer $p < n$. At a minimum, we maximize only those terms corresponding to positive values of λ_l. (Maximizing ones corresponding to negative λ_l would reduce, not increase, the modularity.) In effect, we are making a rank-p approximation to the modularity matrix, based on its leading p eigenvectors, then calculating the modularity using that approximation rather than the true modularity matrix.

Noting that all λ_l in Eq. (8) are now positive, we can rewrite the equation as

$$Q = \frac{1}{2m}\sum_{s=1}^{k}\sum_{l=1}^{p}\Bigl[\sum_i \sqrt{\lambda_l}U_{il}\delta_{s,g_i}\Bigr]^2. \tag{9}$$

We define a set of n p-dimensional *vertex vectors* $\mathbf{r}_i$ with elements

$$\bigl[\mathbf{r}_i\bigr]_l = \sqrt{\lambda_l}U_{il}, \tag{10}$$

in terms of which the modularity is

$$Q = \frac{1}{2m}\sum_{s=1}^{k}\sum_{l=1}^{p}\Bigl[\sum_{i\in s}\bigl[\mathbf{r}_i\bigr]_l\Bigr]^2 = \frac{1}{2m}\sum_{s=1}^{k}\Bigl|\sum_{i\in s}\mathbf{r}_i\Bigr|^2, \tag{11}$$

where the notation $i \in s$ denotes that vertex i is in group s.

In other words, we assign to each vertex a vector $\mathbf{r}_i$, which can be calculated solely from the structure of the network (since it is expressed in terms of the eigenvalues and eigenvectors of the modularity matrix) and hence is constant throughout the optimization procedure. Then the modularity of a division of the network into groups is given (apart from the leading constant $1/2m$) as a sum of contributions, one from each group s, equal to the square of the sum of the vectors for the vertices in that group. Our goal is to find the division that maximizes this modularity.

Generically, problems of this kind are called *max-sum vector partitioning* problems, or just vector partitioning for short. In the following section we propose a heuristic algorithm to rapidly solve vector partitioning problems and show how it can be applied to perform efficient multiway spectral community detection in arbitrary networks.

We have not yet said what the value should be of the constant p that specifies the rank at which we approximate the modularity matrix in Eq. (8). We have said that p should be no greater than the number of positive eigenvalues of the modularity matrix. On the other hand, as shown in [16], if p is less than $k-1$ then the division of the network with maximum modularity always has less than k communities, since there will be at least one pair of communities whose amalgamation into a single community will increase the modularity. Thus p should be greater than or equal to $k-1$. In all of the calculations presented in this paper we make the minimal choice $p = k-1$, which gives the fastest algorithm and in most cases gives excellent results. However, it is worth bearing in mind that larger values of p are possible and, in principle, give more accurate approximations to the true value of the modularity.

III. VECTOR PARTITIONING ALGORITHM

Vector partitioning is computationally easier than many discrete optimization tasks. In particular, it is solvable in polynomial, rather than exponential time. A general k-way partitioning of n different p-dimensional vectors can be solved exactly in time $\mathrm{O}(n^{p(k-1)-1})$ [23]. Thus if we use the leading two eigenvectors of the modularity matrix to divide a network into two communities the calculation can be done in time $\mathrm{O}(n)$, as shown previously in [16]. However the running time quickly becomes less tractable for larger numbers of communities. As discussed above, for a division of a network into k communities we must use at least $k-1$ eigenvectors, which gives a running time $\mathrm{O}(n^{k^2-2k})$. Even for just three communities this gives $\mathrm{O}(n^3)$, which is practical only for rather small networks, and for four communities it gives $\mathrm{O}(n^8)$ which is entirely impractical. For applications to realistically large networks with $k > 2$, therefore, we must abandon exact solution of the problem and look for faster approximate methods.

Previous approaches to vector partitioning include that of Wang *et al.* [24], who suggest dividing the space of vectors into octants (or their generalization in higher dimensions) and looking through all 2^{k-1} of them to find the k octants that contain the largest numbers of vectors. Then we use these as an initial coarse division and assign the remaining vectors to these groups by brute-force optimization. This method works reasonably well for small values of k but is not ideal as k becomes larger because the number of octants increases exponentially with k. Richardson *et al.* [14] proposed a divide-and-conquer method that works by splitting the space into octants again, but then splitting these into smaller wedges, and repeating until further subdivision gives no improvement. This method works well for the $k = 3$ case with two eigenvectors but does not generalize well to higher k. Alpert and Yao [22] proposed a greedy algorithm that works for any value of k by adding vectors one by one to the set to be partitioned, with vectors of larger magnitude being added first (on the grounds that these contribute most to the sums in Eq. (11)). This method works well when the largest magnitude vectors are distributed evenly among the final groups, but more poorly when they are concentrated in a few groups. Unfortunately, as we show in Section IV A, when network communities are of unequal sizes the largest vertex vectors do indeed tend to be concentrated in a few groups and the method of [22] works less well.

Here we introduce an alternative and well-motivated heuristic for finding the solution to vector partitioning problems for general values of k. The algorithm is analogous to the k-means algorithm for the standard data partitioning problem. The k-means method is an algorithm for partitioning a set of data points in any number of dimensions into k clusters in which we start by choosing k index locations or centroids in the space. These could be chosen in several ways: entirely at random, at random from among the set of data points, or (most commonly) as the centroids of some initial approximate partition of the data. Once these are chosen, we compute the distance from each data point to each of the k centroids

and divide the data points into k groups according to which they are closest to. Then we compute the k centroids of these new groups, replace the old centroids with the new ones, and repeat. The process continues until the centroids stop changing.

Our algorithm adopts a similar idea for vector partitioning, with points being replaced by vectors and distances by vector inner products. We start by choosing an initial set of k *group vectors* $\mathbf{R}_s$, one for each group or community s, then we assign each of our vertex vectors $\mathbf{r}_i$ to one of the groups according to which group vector it is closest to, in a sense we will define in a moment. Then we calculate new group vectors for each community from these assignments and repeat. The new group vectors are calculated simply as the sums of the vertex vectors in each group:

$$\mathbf{R}_s = \sum_{i\in s} \mathbf{r}_i, \tag{12}$$

so that the modularity, Eq. (11), is equal to

$$Q = \frac{1}{2m}\sum_s \bigl|\mathbf{R}_s\bigr|^2. \tag{13}$$

We observe the following property of this modularity. Suppose we move a vertex i from one community s to another t. Let $\mathbf{R}_s$ and $\mathbf{R}_t$ represent the group vectors of the two communities excluding the contribution from vertex i. Then, before the move, the group vectors of the communities are $\mathbf{R}_s + \mathbf{r}_i$ and $\mathbf{R}_t$, and after the move they are $\mathbf{R}_s$ and $\mathbf{R}_t + \mathbf{r}_i$. All other communities remain unchanged in the meantime and hence the change ΔQ in the modularity upon moving vertex i is

$$\begin{aligned}\Delta Q &= \frac{1}{2m}\bigl[|\mathbf{R}_s|^2 + |\mathbf{R}_t + \mathbf{r}_i|^2 - |\mathbf{R}_s + \mathbf{r}_i|^2 - |\mathbf{R}_t|^2\bigr] \\ &= \frac{1}{m}\bigl[\mathbf{R}_t^T\mathbf{r}_i - \mathbf{R}_s^T\mathbf{r}_i\bigr].\end{aligned} \tag{14}$$

Thus the modularity will either increase or decrease depending on which is the larger of the two inner products $\mathbf{R}_t^T\mathbf{r}_i$ and $\mathbf{R}_s^T\mathbf{r}_i$. Or, to put that another way, in order to maximize the modularity we should assign vertex i to the community whose group vector has the largest inner product with $\mathbf{r}_i$.

This then defines our equivalent of "distance" for our k-means style vector partitioning algorithm. Given a set of group vectors $\mathbf{R}_s$, we calculate the inner product $\mathbf{R}_s^T\mathbf{r}_i$ between $\mathbf{r}_i$ and every group vector and then assign vertex i to the community with the highest inner product.

Note, however, that the group vectors $\mathbf{R}_s$ and $\mathbf{R}_t$ appearing in Eq. (14) are defined *excluding* $\mathbf{r}_i$ itself. To be correct, therefore, we should do the same thing in our partitioning algorithm. For every vertex vector $\mathbf{r}_i$ there will be one group vector $\mathbf{R}_s$ that contains that vertex vector (in the sense of Eq. (12)) and before calculating the inner product for that group we should subtract $\mathbf{r}_i$ from the group vector. In practice this subtraction typically makes little difference when the network is large—the subtraction or not of a single vertex from a large group is not going to change the results much. In many cases, therefore, one can omit the subtraction step. On the other hand, the algorithm is not significantly slower with the subtraction, so one could also argue for its inclusion, purely on grounds of correctness. We do include it in the calculations of this paper, but in the end it makes little difference to the results.

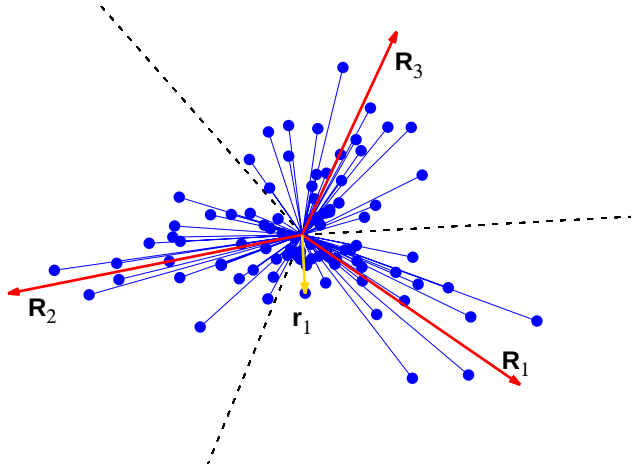

FIG. 1: Depiction of the operation of our vector partitioning heuristic for, in this case, a set of two-dimensional vectors being divided into three groups. The blue lines and dots denote the individual vectors. The red lines are the group vectors. (The magnitudes of the group vectors have been rescaled to fit into the figure—normally they would be much larger, since they are the sums of the individual vectors in each group.) The dashed lines indicate the borders between communities, which are determined both by the angles and relative magnitudes of the group vectors. For example, the vector labeled $\mathbf{r}_1$ will be assigned to group 1 in this case, because it has its largest inner product with $\mathbf{R}_1$.

Our complete vector partitioning algorithm is the following:

1. Choose an initial set of group vectors $\mathbf{R}_s$, one for each of the k communities.
2. Compute the inner product $\mathbf{R}_s^T\mathbf{r}_i$ for all vertices i and all communities s, or $(\mathbf{R}_s - \mathbf{r}_i)^T\mathbf{r}_i$ if vertex i is currently assigned to group s.
3. Assign each vertex to the community with which it has the highest (most positive) inner product.
4. Update the group vectors using the definition of Eq. (12).
5. Repeat from step 2 until the group vectors stop changing.

See Fig. 1 for an illustration of the working of the algorithm.

The algorithm is efficient, with a time complexity of $O(nk)$ for each iteration of steps 2 to 4. Total running

time depends on the number of iterations required to reach convergence, for which we do not have any theoretical results. In practice, however, the number of iterations is small—around half a dozen or less in most of the examples we have looked at—and one could reduce the number further by halting the algorithm when the changes become small or after some maximum number of iterations, just as some k-means implementations also do. As discussed below, one may also want to repeat the entire calculation more than once with different initial group vectors (step 1) to avoid poor results due to unlucky initial conditions.

We still need to decide how the initial group vectors are to be chosen. In the simplest case we might just choose them to be of equal magnitude and point in random directions. However, if there is community structure in the network then we expect the vertex vectors to be clustered, pointing in a small number of directions, with no or few vectors pointing in the remaining directions. It makes little sense to pick initial group vectors pointing in directions well away from where the clusters lie, so in practice we have found that, rather than giving the group vectors random directions, we can get good results by picking them randomly from among the vertex vectors themselves. This ensures that, if most vectors point in a few directions, we will be likely to choose initial group vectors that also point in those directions. (The group vectors must also be pointing in directions at least 90 degrees away from one another, otherwise one can always increase the modularity in Eq. (13) by merging two groups together.)

Note that we need only pick $k-1$ of the k group vectors in this fashion, the final vector being fixed by the fact that the group vectors sum to zero. To see this, recall that the uniform vector $\mathbf{1} = (1, 1, 1, \ldots)$ is always an eigenvector of the modularity matrix, which implies that the elements of all other eigenvectors—i.e., the columns of the orthogonal matrix $\mathbf{U}$—must sum to zero (since they must be orthogonal to the uniform vector). Then the definition of Eq. (10) implies that

$$\sum_{i=1}^{n} [\mathbf{r}_i]_l = \sqrt{\lambda_l} \sum_{i=1}^{n} U_{il} = 0, \tag{15}$$

and hence

$$\sum_{i=1}^{n} \mathbf{r}_i = 0, \tag{16}$$

and

$$\sum_s \mathbf{R}_s = \sum_s \sum_{i \in s} \mathbf{r}_i = \sum_{i=1}^{n} \mathbf{r}_i = 0. \tag{17}$$

Thus, once we have chosen $k-1$ of the group vectors randomly, the final one is fixed to be equal to minus the sum of the rest.

Since there is a random element in the initialization of our algorithm, its result is not always guaranteed to be the same, even when applied to the same network with the same parameter values; it may give different results for the modularity on different runs. In applications, therefore, we typically do several runs of the algorithm with different initial conditions, choosing from among the results the community division that gives the highest value of the modularity.

IV. APPLICATIONS

In this section we give example applications of our method, first to computer-generated test networks and then to two real-world examples.

A. Synthetic networks

For our first tests of the method we look at a set of computer-generated ("synthetic") benchmark networks that contain known community structure. Our goal is to see whether, and how accurately, the algorithm can recover that structure. In our tests we make use of networks generated using the *degree-corrected stochastic block model* [25]. The *stochastic block model* (not degree-corrected) is a generative model of community-structured networks whose origins go back to the 1980s [26, 27]. Vertices are divided into groups and edges are placed between pairs independently at random with probabilities ω_{st} that depend only on the groups s, t that the vertices belong to. If the diagonal probabilities ω_{ss} are larger than the off-diagonal ones, then the network will display classic "assortative" community structure with more connections within groups than between them. The stochastic block model is unrealistic, however, in generating a Poisson distribution of vertex degrees, which is quite different from the highly right-skewed distributions commonly seen in real networks. The degree-corrected block model remedies this problem by fixing the (expected) degrees of the vertices at any values we choose. In this model edges are placed independently between pairs of vertices i, j with probability $d_i d_j \omega_{st}$, where d_i is the desired degree of vertex i. For a detailed discussion see [25].

Our tests consist of generating a number of networks using the degree-corrected block model, analyzing them using our algorithm, then comparing the communities found with those planted in the networks in the first place. To quantify the similarity of the two sets of communities, planted and detected, we make use of a standard measure, the *normalized mutual information* or NMI [29, 30]. The (unnormalized) mutual information of two sets X, Y of numbers or measurements is defined to be

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}, \tag{18}$$

where $p(x, y)$ is the joint probability or frequency of

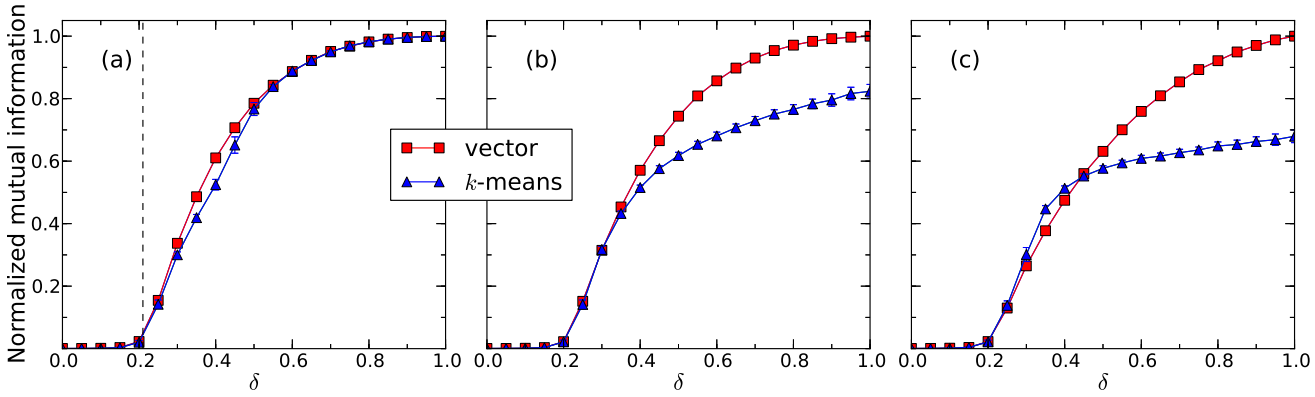

FIG. 2: Normalized mutual information (NMI) as a function of the parameter δ for communities detected in randomly generated test networks using the vector partitioning algorithm of this paper (red squares) and the k-means method (blue triangles). The networks consist of $n = 3600$ vertices each, divided into three communities thus: (a) equally sized communities of 1200 vertices each; (b) communities of sizes 1800, 1200, and 600; (c) communities of sizes 2400, 900, and 300. Each data point represents the highest NMI found over 20 runs of the relevant algorithm with different random initializations, averaged over 100 networks. The vertical dashed line in panel (a) indicates the position of the detectability threshold below which all methods must fail [28].

x and y within the data set and $p(x)$, $p(y)$ are their marginal probabilities. The mutual information measures how much you learn about one of the two sets of measurements by knowing the other. If X and Y are uncorrelated then each tells you nothing about the other and the mutual information is zero. If they are perfectly correlated then each tells you everything about the other and the mutual information takes its maximum value, which is equal to all of the information that either set contains, which is simply the entropy, $H(X)$ or $H(Y)$, of the set.

Having the maximum value of the mutual information be equal to the entropy is in some ways inconvenient, since we don't know in advance what that value will be. So commonly one normalizes the mutual information by dividing by the mean of the entropies of the two sets, thus:

$$\mathrm{NMI}(X;Y) = \frac{I(X;Y)}{\frac{1}{2}[H(X)+H(Y)]}. \tag{19}$$

This normalized value falls in the interval from zero to one, with uncorrelated variables giving zero and perfect correlation giving one.

The NMI is commonly used to quantify the match between two clusterings of the vertices of a network. In the present case, the original assignments of vertices to groups in the block model (the "planted communities") are used as one set of measurements X and the assignments found by our algorithm (the "detected communities") are the other Y. An NMI of 1 denotes perfect recovery of the planted partition; an NMI of 0 indicates complete failure.

In the tests presented here we use networks of $n = 3600$ vertices divided into $k = 3$ communities and with two different (expected) degrees: half the vertices in each group have degree 10 and the other half have degree 30. The parameters ω_{st} are varied in order to tune the difficulty of the community detection according to

$$\omega_{st} = (1-\delta)\omega_{st}^{\mathrm{random}} + \delta\omega_{st}^{\mathrm{planted}}, \tag{20}$$

where δ is a parameter that varies from zero to one and

$$\omega_{st}^{\mathrm{random}} = \frac{1}{2m}, \qquad \omega_{st}^{\mathrm{planted}} = \frac{\delta_{st}}{\sum_{i\in s} d_i}, \tag{21}$$

with m being the total number of edges in the network, as previously. With this choice, the parameter δ tunes the edge probabilities from a value of $d_i d_j/2m$ when $\delta = 0$, which corresponds to a purely random edge distribution with no community structure at all (the so-called configuration model [31–33]), to a value of $d_i d_j / \sum_{i\in s} d_i$ within each group s and zero between groups when $\delta = 1$—effectively three separate, unconnected configuration models, one for each group, which is the strongest form of community structure one could have. This choice of ω_{st} also has the nice property that the expected fraction of within-group edges that a vertex has is the same for all vertices.

We have tested our algorithm on these networks using two eigenvectors to define the vertex vectors $\mathbf{r}_i$ (the minimum viable number). The results are shown as a function of the parameter δ in Fig. 2, along with results for the same networks analyzed by clustering the vertex vectors using the k-means algorithm of Ref. [17].

As $\delta \to 1$ the community structure in the network becomes strong and any reasonable algorithm should be able to detect it. As we approach this limit our algorithm assigns 100% of vertices to their correct communities and the NMI approaches one. Conversely as $\delta \to 0$ the community structure in the network vanishes and neither algorithm should detect anything, so NMI approaches zero. Furthermore, it is known that there is a critical strength

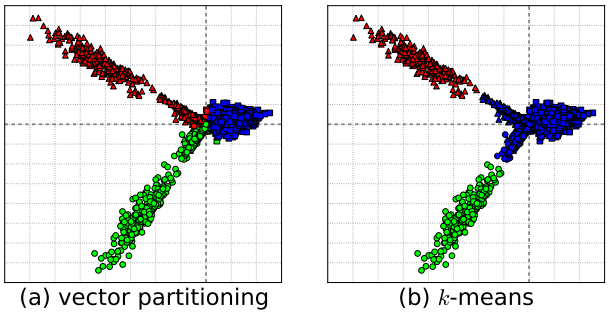

FIG. 3: Illustration of the division of a synthetic three-group network using (a) the algorithm of this paper and (b) the k-means algorithm. Shapes indicate the planted communities while colors indicate the communities found by the two algorithms. Observe how the k-means algorithm assigns a significant portion of vertices belonging to the red and green communities incorrectly to the blue one, while the vector partitioning approach does not have this problem. The network in this case has $n = 4000$ vertices with communities of size 3000, 500, and 500.

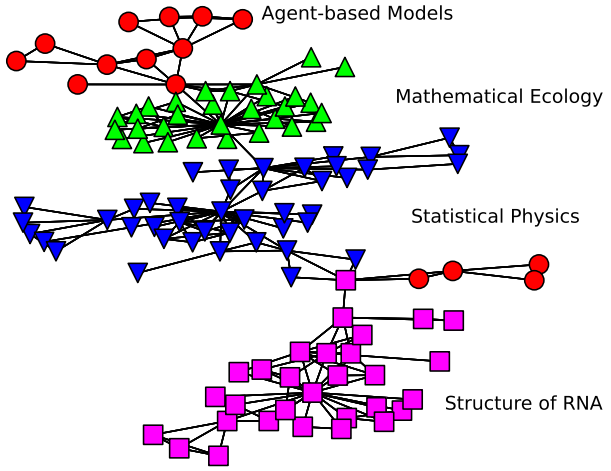

FIG. 4: Four-way division into communities of a collaboration network of scientists at the Santa Fe Institute. Different colors and shapes indicate the communities discovered by the vector partitioning algorithm of this paper. The communities split roughly along lines of research topic.

of the structure—which translates to a critical value of our parameter δ—below which the structure is so weak that no algorithm can detect it [28]. This "detectability threshold" is marked in Fig. 2a with a vertical dashed line. Above this point it should be possible to detect the communities, albeit with a certain error rate, and indeed we see that both algorithms achieve a nonzero NMI in this region.

As the figure shows, the vector partitioning algorithm does as well or better than k-means in almost all cases. In panel (a) the three communities in the network have equal sizes, and in this case the two algorithms perform comparably, there being only a small range of parameter values in the middle of the plot where vector partitioning outperforms k-means by a narrow margin. In panels (b) and (c) the communities have unequal sizes—moderately so in (b) and highly in (c)—and in these cases vector partitioning does significantly better than k-means. Indeed for unequal group sizes the k-means algorithm fails to achieve perfect community classification (NMI = 1) even in the limit where $\delta = 1$. The reason for this is illustrated in Fig. 3, which shows a scatter plot of the vertex vectors for an illustrative example network along with the communities into which each algorithm divides the vertices (shown by the colors). As the figure shows, when the groups are unequal in size the largest group is closer to the origin than the smaller ones—necessarily so since the centroid of the vertex vectors lies at the origin (Eq. (16)). This tends to throw off the k-means algorithm, which by definition splits the points into groups of roughly equal spatial extent. The vector partitioning method, which is (correctly) sensitive only to the direction and not the magnitude of the vertex vectors, has no such problems.

B. Real-world examples

Our next two example applications are to real-world networks, two collaboration networks among scientists. The first, taken from Ref. [34], represents scientists working at the Santa Fe Institute, an interdisciplinary research institute in New Mexico. The vertices in the network represent the scientists and the edges indicate that two scientists coauthored a paper together at least once. The network is small enough to allow straightforward visualization of the results and is interesting in that the scientists it represents, in keeping with the interdisciplinary mission of the institute, come from a range of different research fields, in this case statistical physics, mathematical ecology, RNA structure, and agent-based modeling. It is plausible that the communities in the network might reflect these subject areas.

Figure 4 shows the result of a four-way community division of this network using vertex vectors constructed from the first three eigenvectors of the modularity matrix. Overall the results mirror our expectations, with the four subject areas corresponding roughly to the four communities found by the method. We note, however, that there are also four vertices in the middle-right of the figure that are clearly misclassified as being in the "agent-based models" group when they would be more plausibly placed in the "structure of RNA" group. This illustrates a potential weakness of the algorithm: the defining feature of these vertices is that their vertex vectors have very small magnitude, meaning that they do not strongly belong to any group. For such vertices even a small error—such as that introduced by making our low-rank approximation to the true modularity matrix—can alter the direction of the vertex vector substantially and

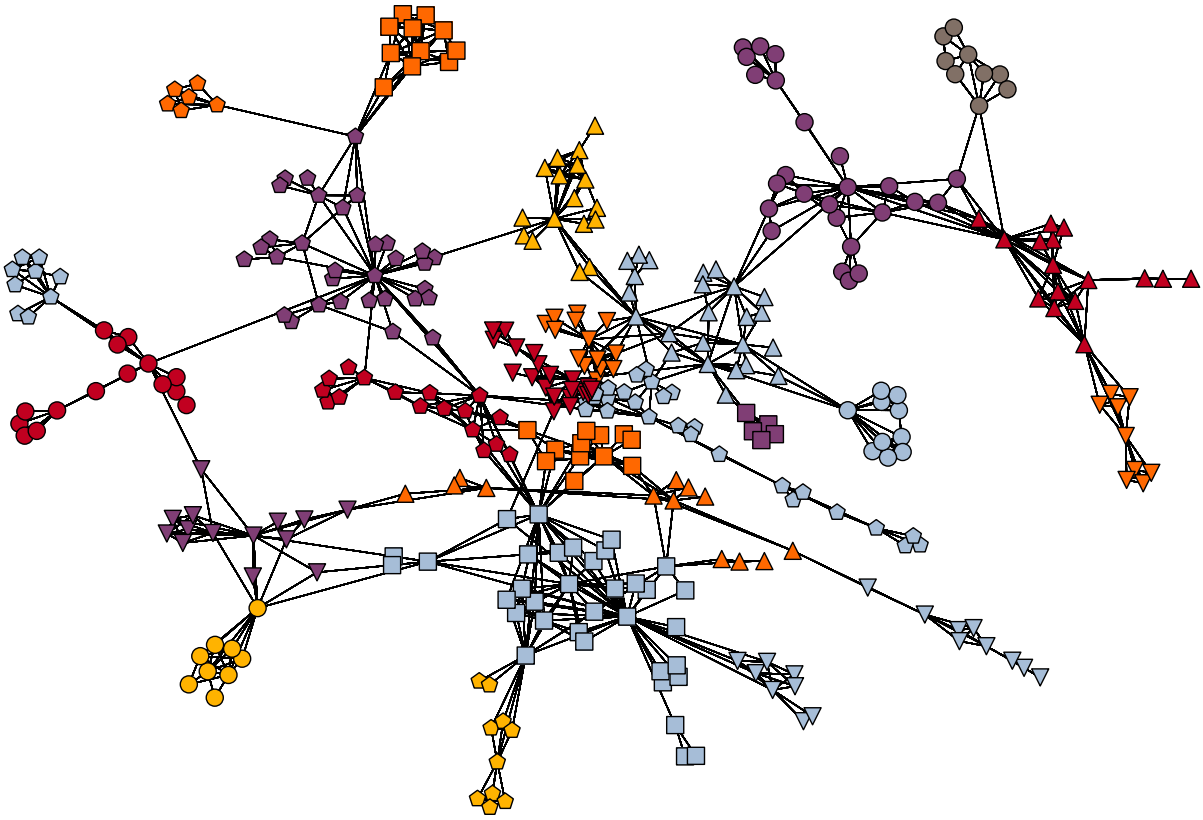

FIG. 5: The 21 communities found in a collaboration network of network scientists using the algorithm proposed in this paper.

hence move a vertex to a different group. Problems like this are, in fact, common to many spectral algorithms and are typically handled by combining the algorithm with a subsequent iterative refinement or "fine tuning" step, in which individual vertices or small sets are moved from group to group in an effort to improve the value of the modularity [11, 14]. The spectral algorithm is good at determining the "big picture," rapidly doing an overall division of the network into broad groups of vertices; the subsequent fine tuning tidies up the remaining details. Based on the results we see here, our algorithm might be a good candidate for combination with a fine tuning step of this kind.

Our second real-world example is a collaboration network of scientists working in the field of network science itself and is taken from Ref. [16]. Apart from being rather larger than the Santa Fe Institute network, at 379 scientists, this network also differs in that all its members are, ostensibly at least, studying the same subject, so there is no obvious "ground truth" for the communities as there was in the previous example, or even for how many communities there should be. Choosing the number of communities into which a network should be divided is a deep problem in its own right, and one that is not completely solved. Here, however, we simply borrow a technique from the literature and estimate the number of communities in the network by counting the real eigenvalues of the so-called non-backtracking matrix that are greater than the largest real part among the complex eigenvalues. (For a discussion of why this is a good heuristic, see [35].) In the present case this suggests that there should be 26 communities in the network, so we choose $k = 26$ for our community detection algorithm and construct the vertex vectors from the leading 25 eigenvectors of the modularity matrix. The results are shown in Fig. 5. In fact, in this case we find that the algorithm does not make use of all 26 communities—the figure contains only 21. Nonetheless, the algorithm has succeeded in finding a division with high modularity: the modularity value is $Q = 0.83$, comparable to the value given for example in [14] for the same network. We note, however, that, as is typical for larger values of k, the algorithm finds a range of different divisions of the network in different runs that all have competitive modularity. The existence of competing good community divisions in the same network is a well-known phenomenon and has been previously discussed for instance by Good *et al.* [36].

V. CONCLUSIONS

In this paper we have described a mapping of a multiway spectral community detection method onto a vector partitioning problem and proposed a simple heuristic algorithm for vector partitioning that returns good results in this application. We have tested our method on computer-generated benchmark networks, comparing it with a competing spectral algorithm that makes use of k-means clustering, and find our method to give superior performance, particularly in cases where the sizes of the communities are unequal. We have also given two example applications of our method to real-world networks.

There remain a number of open questions not answered in this paper. Although the algorithm we propose is simple and efficient, it is only approximate and we have no formal results on its expected performance. The algorithm also assumes we have prior knowledge of the number of communities in the network, where in reality this is not usually the case. Determining the number of communities in a network is an interesting open problem. Finally, as we (and others) have pointed out, the best community detection methods are typically hybrids of two or more elementary methods. It would be interesting to see how the vector partitioning algorithm we propose works in combination with other methods. These problems, however, we leave for future work.

Acknowledgments

The authors thank Maria Riolo and Raj Rao Nadakuditi for helpful conversations. This research was funded in part by the US National Science Foundation under grants DMS–1107796 and DMS–1407207.

[1] S. Fortunato, Phys. Rep. **486**, 75 (2010).
[2] M. E. J. Newman and M. Girvan, Phys. Rev. E **69**, 026113 (2004).
[3] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, in *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science No. 4769 (Springer, Berlin, 2007).
[4] M. E. J. Newman, Phys. Rev. E **69**, 066133 (2004).
[5] A. Clauset, M. E. J. Newman, and C. Moore, Phys. Rev. E **70**, 066111 (2004).
[6] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral, Phys. Rev. E **70**, 025101 (2004).
[7] A. Medus, G. Acuña, and C. O. Dorso, Physica A **358**, 593 (2005).
[8] J. Duch and A. Arenas, Phys. Rev. E **72**, 027104 (2005).
[9] L. Shuzhuo, C. Yinghui, D. Haifeng, and M. W. Feldman, Complexity **15**, 53 (2010).
[10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech. **2008**, P10008 (2008).
[11] M. E. J. Newman, Proc. Natl. Acad. Sci. USA **103**, 8577 (2006).
[12] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, 2000).
[13] R. R. Nadakuditi and M. E. J. Newman, Phys. Rev. Lett. **108**, 188701 (2012).
[14] T. Richardson, P. J. Mucha, and M. A. Porter, Phys. Rev. E **80**, 036111 (2009).
[15] X. Gong, K. Li, M. Li, and C.-H. Lai, Europhys. Lett. **101**, 48001 (2013).
[16] M. E. J. Newman, Phys. Rev. E **74**, 036104 (2006).
[17] S. White and P. Smyth, in *Proceedings of the 5th SIAM International Conference on Data Mining*, edited by H. Kargupta, J. Srivastava, C. Kamath, and A. Goodman (Society for Industrial and Applied Mathematics, Philadelphia, 2005).
[18] U. Elsner, *Graph partitioning—a survey*, Technical Report 97-27 (Technische Universität Chemnitz, 1997).
[19] P.-O. Fjällström, Linköping Electronic Articles in Computer and Information Science **3** (1998).
[20] M. E. J. Newman, Phys. Rev. E **88**, 042822 (2013).
[21] C. J. Alpert, A. B. Kahng, and S.-Z. Yao, Discrete Applied Mathematics **90**, 3 (1999).
[22] C. J. Alpert and S.-Z. Yao, in *Proceedings of the 32nd International Conference on Design Automation*, edited by B. T. Preas, P. G. Karger, B. S. Nobandegani, and M. Pedram (Association of Computing Machinery, New York, NY, 1995) pp. 195–200.
[23] S. Onn and L. J. Schulman, Mathematics of Operations Research **26**, 583 (2001).
[24] G. Wang, Y. Shen, and M. Ouyang, Computers & Mathematics with Applications **55**, 2746 (2008).
[25] B. Karrer and M. E. J. Newman, Phys. Rev. E **83**, 016107 (2011).
[26] P. W. Holland, K. B. Laskey, and S. Leinhardt, Social Networks **5**, 109 (1983).
[27] A. Coja-Oghlan and A. Lanka, SIAM Journal on Discrete Mathematics **23**, 1682 (2009).
[28] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová, Phys. Rev. Lett. **107**, 065701 (2011).
[29] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas, J. Stat. Mech. **2005**, P09008 (2005).
[30] M. Meilă, Journal of Multivariate Analysis **98**, 873 (2007).
[31] M. Molloy and B. Reed, Random Structures and Algorithms **6**, 161 (1995).
[32] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, Phys. Rev. E **64**, 026118 (2001).
[33] F. Chung and L. Lu, Annals of Combinatorics **6**, 125 (2002).
[34] M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA **99**, 7821 (2002).
[35] F. Krzakala, C. Moore, E. Mossel, J. Neeman, A. Sly, L. Zdeborová, and P. Zhang, Proc. Natl. Acad. Sci. USA **110**, 20935 (2013).
[36] B. H. Good, Y.-A. de Montjoye, and A. Clauset, Phys. Rev. E **81**, 046106 (2010).