Detection of core–periphery structure in networks using spectral methods and geodesic paths

MIHAI CUCURINGU[1], PUCK ROMBACH[1], SANG HOON LEE[2,3] and
MASON A. PORTER[2,4]

[1]*Department of Mathematics, UCLA, Los Angeles, CA, USA*
emails: mihai@math.ucla.edu, rombach@math.ucla.edu
[2]*Oxford Centre for Industrial and Applied Mathematics, Mathematical Institute,*
University of Oxford, Oxford, UK
email: porterm@maths.ox.ac.uk
[3]*School of Physics, Korea Institute for Advanced Study, Seoul, Korea*
email: lshlj82@kias.re.kr
[4]*CABDyN Complexity Centre, University of Oxford, Oxford, UK*

We introduce several novel and computationally efficient methods for detecting "core–periphery structure" in networks. Core–periphery structure is a type of mesoscale structure that consists of densely connected core vertices and sparsely connected peripheral vertices. Core vertices tend to be well-connected both among themselves and to peripheral vertices, which tend not to be well-connected to other vertices. Our first method, which is based on transportation in networks, aggregates information from many geodesic paths in a network and yields a score for each vertex that reflects the likelihood that that vertex is a core vertex. Our second method is based on a low-rank approximation of a network's adjacency matrix, which we express as a perturbation of a tensor-product matrix. Our third approach uses the bottom eigenvector of the random-walk Laplacian to infer a coreness score and a classification into core and peripheral vertices. We also design an objective function to (1) help classify vertices into core or peripheral vertices and (2) provide a goodness-of-fit criterion for classifications into core versus peripheral vertices. To examine the performance of our methods, we apply our algorithms to both synthetically generated networks and a variety of networks constructed from real-world data sets.

1 Introduction

Network science has grown explosively during the past two decades [65], and myriad new journal articles on network science appear every year. One focal area in the networks literature is the development and analysis of algorithms for detecting local, mesoscale, and global structures in various types of networks [35,74]. Mesoscale features are particularly interesting, as they arise neither at the local scale of vertices (i.e., nodes) and edges nor at the global scale of summary statistics. In the present paper, we contribute to research on mesoscale network structures by developing and analysing new (and computationally

efficient) algorithms for detecting a feature known as *core–periphery structure*, which consists of densely connected *core* vertices and sparsely connected *peripheral* vertices.

The importance of investigating mesoscale network structures is acknowledged widely [35, 74], but almost all of the research on this topic concerns a specific type of feature known as *community structure*. In studying community structure, one typically employs some algorithm to detect sets of vertices called *communities* that consist of vertices that are densely connected to each other, such that the connection density between vertices from different communities is comparatively sparse [35, 38, 67, 74]. A diverse array of methods exist to detect community structure, and they have been applied to numerous areas, such as committee networks in political science [73], friendship networks [40, 86], protein interaction networks [19, 57], functional brain networks [8], and mobile phone networks [68]. Popular methods include optimization of a quality function called "modularity" [64, 66, 67], spectral partitioning [43, 84], dynamical approaches based on random walkers or other dynamical systems [3, 46, 71, 72, 78], and more. Most community-detection methods require a vertex to belong to a distinct community, but several methods also allow the detection of overlapping communities (see, e.g., [1, 4, 46, 69]).

Core–periphery structure is a mesoscale feature that is rather different from community structure. The main difference is that core vertices are well-connected to peripheral vertices, whereas the standard perspective on community structure views communities as nearly decomposable modules (which leads to trying to find the best block-diagonal fit to a network's adjacency matrix) [76, 92]. Core–periphery structure and community structure are thus represented by different types of block models [46, 70]. The quantitative investigation of core–periphery structure has a reasonably long history [26], and qualitative notions of core–periphery structure have long been considered in fields such as international relations [18, 82, 85, 90], sociology [30, 54], and economics [53] (and have been examined more recently in applications such as neuroscience [9], transportation [56], and faculty movements in academia [21]), but the study of core–periphery structure remains poorly developed—especially in comparison to the study of community structure [35, 74]. Most investigations of core–periphery structure tend to use the perspective that a network's adjacency matrix has an intrinsic block structure (which is different from the block structure from community structure) [13, 25, 76]. Very recently, for example, Ref. [92] identified core–periphery structure by fitting a stochastic block model (SBM) to empirical network data using a maximum likelihood method, and the SBM approach in Ref. [70] can also be used to study core–periphery structure. Importantly, it is possible to examine core–periphery structure (and related structures) using a wealth of different perspectives, such as overlapping communities [91], k-cores [44], network capacity [27], and random walks [29]. It is also interesting to examine growth mechanisms to generate networks with core–periphery structure [89]. The notion of "nestedness" [7] from ecology is also related to core–periphery structure [55].

The main contribution of the present paper is the development of novel algorithms for detecting core–periphery structure. Our aim is to develop algorithms that are both computationally efficient and robust to high levels of noise in data, as such situations can lead to a blurry separation between core vertices and peripheral vertices.

The rest of this paper is organized as follows. In Section 2, we give an introduction to core–periphery structure and briefly survey a few of the existing methods to detect such

structure. In Section 3, we introduce the PATH-CORE method, which is based on computing shortest paths between vertices of a network, for detecting core–periphery structure. In Section 4, we introduce an objective function for detecting core–periphery structure that exploits our proposed algorithms and helps in the classification of vertices into a core set and a periphery set. In Section 5, we propose the spectral method LOWRANK-CORE, which detects core–periphery structure by considering the adjacency matrix of a network as a low-rank perturbation matrix. In Section 6, we investigate two Laplacian-based methods (LAP-CORE and LAPSGN-CORE) for computing core–periphery structure in a network, and we discuss related work in community detection that uses a similar approach. In Section 7, we compare the results of applying the above algorithms using several synthetically generated networks and real-world networks. Finally, we summarize and discuss our results in Section 8, and we also discuss several open problems and potential applications. In Appendix A, we detail the steps of our proposed PATH-CORE algorithm for computing the PATH-CORE scores, and we include an analysis of its computational complexity. In Appendix B, we discuss the spectrum of the random-walk Laplacian of a graph (and the spectrum of the random-walk Laplacian of its complement). In Appendix C, we detail an experiment with artificially planted high-degree peripheral vertices that illustrates the sensitivity of a degree-based method (which we call DEGREE-CORE and which uses vertex degree as a proxy to measure coreness) to such outlier vertices. Finally, in Appendix D, we calculate Spearman and Pearson correlation coefficients between the coreness scores that we obtain from the different methods applied to several real-world networks.

2 Core–periphery structure in networks

The best-known quantitative approach to studying core–periphery structure was introduced by Borgatti and Everett [13], who developed algorithms for detecting discrete and continuous versions of core–periphery structure in weighted, undirected networks. (For the rest of the present paper, note that we will use the terms "network" and "graph" interchangeably.) Their discrete methods start by comparing a network to an ideal block matrix in which the core is fully connected, the periphery has no internal edges, and the periphery is well-connected to the core.

Borgatti and Everett's main algorithm for finding a discrete core–periphery structure assigns each vertex either to a single "core" set of vertices or to a single "periphery" set of vertices. One seeks a vector $\mathbf{C}$ of length n whose entries are either 1 or 0, depending on whether or not the associated vertex has been assigned to the core (1) or periphery (0). We let $H_{ij} = 1$ if $C_i = 1$ (i.e., vertex i is assigned to the core) or $C_j = 1$ (i.e., vertex j is assigned to the core), and we otherwise let $H_{ij} = 0$ (because neither i nor j are assigned to the core). We define $\rho_C = \sum_{i,j} A_{ij} H_{ij}$, where A (with elements A_{ij}) is the adjacency matrix of the (possibly weighted) network G. Borgatti and Everett's algorithm searches for a value of ρ_C that is high compared to the expected value of ρ if $\mathbf{C}$ is shuffled such that the number of 0 and 1 entries is preserved but their order is randomized. The final output of the method is the vector $\mathbf{C}$ that gives the highest z-score for ρ_C. In a variant algorithm for detecting discrete core–periphery structure, Borgatti and Everett still let $H_{ij} = 1$ if both C_i and C_j are equal to 1 and let $H_{ij} = 0$ if neither i nor j are assigned to the core, but they now let $H_{ij} = a \in [0, 1]$ if either $C_i = 1$ or $C_j = 1$ (but not both). To

detect a continuous core–periphery structure [13], Borgatti and Everett assigned a vertex i a core value of C_i and let $H_{ij} = C_i C_j$. A recent method that builds on the continuous notion of core–periphery structure from Ref. [13] was proposed in Ref. [76]. It calculates a core score for weighted, undirected networks; and it has been applied (and compared to community structure) in the investigation of functional brain networks [9]. We use the term CORE-SCORE to describe one of the methods from Ref. [76].

The method of core–periphery detection in the popular network-analysis software UCINET [14] uses the so-called *minimum residual* (MINRES) method [25], which is a technique for factor analysis. One uses factor analysis to describe observed correlations between variables in terms of a smaller number of unobserved variables called the "factors" [28]. MINRES aims to find a vector C that minimizes

$$S(A, \mathbf{C}) = \sum_{i=1}^{n} \sum_{j \neq i} \left(A_{ij} - C_i C_j \right)^2,$$

where $C_i \geqslant 0$ for all vertices i. Note that one ignores the diagonal elements of the network's adjacency matrix. Additionally, because $\mathbf{CC}^T$ is symmetric, this method works best for undirected networks G. For directed networks, one can complement the results of MINRES with a method based on a singular value decomposition (SVD) [15]. In practice, UCINET reports $\mathbf{C}/\sqrt{\sum_i C_i^2}$.

In Ref. [91], it was argued that core–periphery structure can arise as a consequence of community structure with overlapping communities. The authors of [91] presented a so-called "community-affiliation graph model" to capture dense overlaps between communities. In the approach in Ref. [91], the likelihood that two vertices are adjacent to each other is proportional to the number of communities in which they have shared membership. Della Rossa *et al.* recently proposed a method for detecting a continuous core–periphery profile of a (weighted) network by studying the behaviour of a random walker on a network [29]. Approaches based on random walks and other Markov processes have often been employed in the investigation of community structure [46,71,72,78], and it seems reasonable to examine them for other mesoscale structures as well. Very recently, Ref. [92] identified core–periphery structure by fitting an SBM to empirical network data using a maximum-likelihood method. The review article [26] discusses several other methods to detect core–periphery structure in networks.

3 PATH-CORE: Transport-based core–periphery detection via shortest paths in a network

In transportation systems, some locations and routes are much more important than others. This motivates the idea of developing notions of core–periphery structure that are based on transportation. In this section, we restrict our attention to undirected and unweighted networks, although we have also examined transport-based core–periphery structure in empirical weighted and directed networks [56]. In Section 3.1, we explain the intuition behind the proposed PATH-CORE algorithm, and we examine its performance on several synthetic networks. We end this section by commenting on a randomized version of the PATH-CORE algorithm that samples a subset of edges in a graph and computes shortest paths only between the endpoints of the associated vertices.

3.1 Path-Core

The first transport-based algorithm that we propose for detecting core–periphery structure is reminiscent of *betweenness centrality* (BC) in networks [2, 36, 63]. One seeks to measure the extent to which a vertex controls information that flows through a network by counting the number of shortest paths (i.e., geodesic paths) on which the vertex lies between pairs of other vertices in the network. Geodesic vertex BC is defined as

$$B_C(i) = \sum_{j,k \in V(G) \setminus i} \frac{\sigma_{jk}(i)}{\sigma_{jk}}, \tag{3.1}$$

where σ_{jk} is the number of different shortest paths (i.e., the path count) from vertex j to vertex k, and $\sigma_{jk}(i)$ is the number of such paths that include vertex i. Our approach also develops a scoring methodology for vertices that is based on computing shortest paths in a network. Such a score reflects the likelihood that a given vertex is part of a network's core. Instead of considering shortest paths between all pairs of vertices in a network, we consider shortest paths between pairs of vertices that share an edge *when that edge is excluded from the network*. Specifically, we calculate

$$\text{Path-Core}(i) = \sum_{(j,k) \in E(V(G) \setminus i)} \frac{\sigma_{jk}(i)|_{G \setminus (j,k)}}{\sigma_{jk}|_{G \setminus (j,k)}}, \tag{3.2}$$

where $\sigma_{jk}(i)|_{G \setminus (j,k)}$ and $\sigma_{jk}|_{G \setminus (j,k)}$ are defined, respectively, as the path counts σ_{jk} and $\sigma_{jk}(i)$ in the graph $G \setminus (j,k)$, and $E(X)$ denotes the edge set induced by the vertex set X. The network $G \setminus (j,k)$ denotes the subgraph of G that one obtains by removing the edge $(j,k) \in E$. Alternatively, one can define the Path-Core score of a vertex i as the BC of this vertex when considering paths only between pairs of adjacent vertices j and k, but for which the edge e_{jk} incident to the two vertices is discarded. Note that one can apply Path-Core to weighted graphs by using generalizations of BC to weighted graphs.

A related approach was used in Ref. [88] to derive measures of "bridging" in networks based on the observation that edges that reduce distances in a network are important structural bridges. In the calculations in Ref. [88], which employed a modification of closeness centrality, one deletes edges and measures the resulting changes in the lengths of shortest paths (and hence the resulting changes in closeness centralities). We also note the recent paper [33] about bridging centrality.

Let $G(V,E)$ be a graph with a vertex set V of size n (i.e., there are $|V| = n$ vertices) and an edge set E of size m. The set of core vertices is V_C (and its size is n_c), and the set of peripheral vertices is V_P (and its size is n_p). We also sometimes use the notation $\mathtt{C} = |V_C|$ for the size of the core set. Suppose that a network (i.e., a graph) contains exactly one core set and exactly one periphery set, and that these sets are disjoint: $V_C \cup V_P = V$ and $V_C \cap V_P = \emptyset$. The goal of the Path-Core algorithm is to compute a score for each vertex in the graph G that reflects the likelihood that that vertex belongs to the core. In other words, high-scoring vertices have a high probability of being in the core, and low-scoring vertices have a high probability of being in the periphery. Throughout the paper, we use the term "Path-Core scores" to indicate the scores that we associate with a network's vertices by using the Path-Core algorithm.

Table 1. *Block model for the ensemble of graphs* $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$. *Note that either* $p_{cc} \geqslant p_{cp} > p_{pp}$ *or* $p_{cc} > p_{cp} \geqslant p_{pp}$.

$A_{\{CC\}}$	$A_{\{CP\}}$
$A_{\{CP\}}$	$A_{\{PP\}}$

We illustrate our methodology in the context of a generalized block model, such as the one in Table 1, where the submatrices $A_{\{CC\}}$, $A_{\{CP\}}$, and $A_{\{PP\}}$, respectively, represent the interactions between a pair of core vertices, a core vertex and a peripheral vertex, and a pair of peripheral vertices. Suppose that $A_{\{CC\}}$ and $A_{\{PP\}}$ are adjacency matrices that we construct using the $G(n, p)$ random-graph model[1] by considering $G(n_c, p_{cc})$ and $G(n_p, p_{pp})$, respectively, and that $A_{\{CP\}}$ is the adjacency matrix of a random bipartite graph $G(n_c, n_p, p_{cp})$ in which each edge that is incident to both a core and peripheral vertex is present with independent probability p_{cp}. As indicated by the above notation, p_{cc} denotes the probability that there is an edge between any given pair of core vertices, and p_{pp} denotes the probability that there is an edge between any given pair of peripheral vertices. In the context of the block model in Table 1, core–periphery structure arises naturally when either $p_{cc} \geqslant p_{cp} > p_{pp}$ or $p_{cc} > p_{cp} \geqslant p_{pp}$. The above family of random networks, which we denote by $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$, was also considered in Ref. [76]. It contains exactly one set of core vertices, and the remaining vertices are peripheral vertices. More complicated core–periphery structures can also occur [76], such as a mix of (possibly hierarchical) community structures and core–periphery structures [70].

We now present the intuition behind the PATH-CORE algorithm and the reason that the resulting PATH-CORE score is a plausible indicator of the likelihood that a vertex is in the core or in the periphery of a graph G. If i and j are adjacent core vertices, then we expect that shortest paths in $G \setminus (i, j)$ between i and j consist mostly or even entirely of other core vertices. If $i \in V_C$ and $j \in V_P$, then a shortest path in $G \setminus (i, j)$ between i and j should also mostly contain core vertices. Finally, even when $i, j \in V_P$, it is still likely that a shortest path in $G \setminus (i, j)$ between i and j is composed of many core vertices and few peripheral vertices. Intuitively, because $p_{cc} \geqslant p_{cp} \geqslant p_{pp}$, once a shortest path reaches the set V_C, it has a high probability of staying within the core set V_C until it returns to the periphery set V_P and reaches the terminal vertex j. To summarize, we expect core vertices to be on many shortest paths in a graph, whereas peripheral vertices should rarely be on such shortest paths. In other words, because shortest paths between a pair of core vertices are the ones that should, on average, contain the largest fraction of vertices that are in the core, we find that oversampling such paths is an effective way to extract core parts of a graph. Importantly, it is not sufficient in general to simply use a quantity like weighted BC. For example, for a stock-market correlation network that was examined in Ref. [56], weighted BC cannot distinguish the importance of vertices at all, whereas coreness measures (in particular, CORE-SCORE and PATH-CORE) are able to successfully detect core vertices.

---

[1] In the random-graph model $G(n, p)$ on n vertices, an edge is present between each pair of vertices independently with probability p [32, 37]. It is common to abuse terminology and use the name "Erdős–Rényi random graph" for $G(n, p)$.
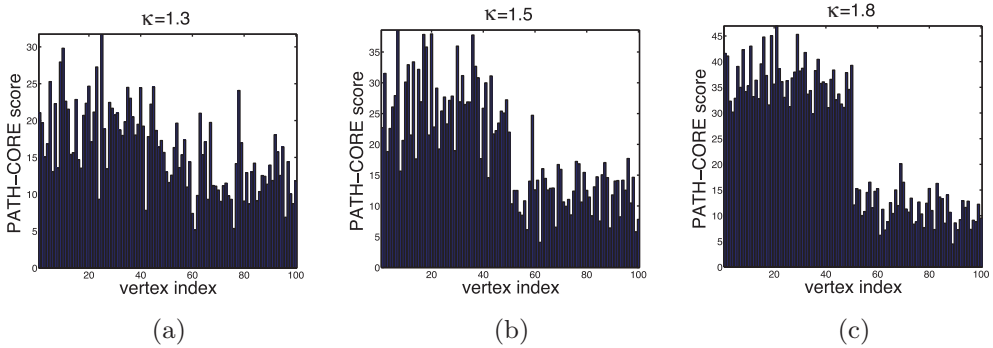
FIGURE 1. PATH-CORE scores of all $n = 100$ vertices for graphs drawn from three different random-graph ensembles in the family $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$. The vector $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$ gives the edge probabilities between between a pair of core vertices (p_{cc}), a core vertex and a peripheral vertex (p_{cp}), and a pair of peripheral vertices (p_{pp}). These probabilities are $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = p$, and we use the fixed value $p = 0.25$. The scalar κ then parametrizes the ensemble. The values of κ are (a) 1.3, (b) 1.5, and (c) 1.8. The first 50 vertices are planted core vertices, and the remaining 50 vertices are planted peripheral vertices.

   To illustrate the effectiveness of the PATH-CORE algorithm, we consider (see Figure 1) several instances of the random-graph ensemble (i.e., model) $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $p_{cc} > p_{cp} > p_{pp}$. Let $\beta = n_p/n$, where $n = n_c + n_p$, denote the fraction of vertices in the core. We assign the edges independently at random according to the following procedure. The edge probabilities for the core–core, core–periphery, and periphery–periphery pairs of vertices are given by the vector $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$, where $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = p$. In our simulations, we fix $n = 100$, $\beta = 0.5$, and $p = 0.25$, and we compute core–periphery structure for 10 instances of the above random-graph ensemble for each of the parameter values $\kappa = 1.1, 1.2, \ldots, 1.9, 2$. To illustrate the effectiveness of the PATH-CORE algorithm, we show in Figure 1 the PATH-CORE for all vertices for three different instances of the above block model. We use the parameter values $\kappa = 1.3$ (which yields $\mathbf{p} = (0.4225, 0.325, 0.25)$), $\kappa = 1.5$ (which yields $\mathbf{p} = (0.5625, 0.375, 0.25)$), and $\kappa = 1.8$ (which yields $\mathbf{p} = (0.81, 0.45, 0.25)$).

   For each of the plots in Figure 1, we place the core vertices in the first 50 positions on the horizontal axis, and we place the peripheral vertices in the remaining 50 positions. The vertical axis indicates the PATH-CORE score associated to each vertex. As expected, vertices in the core set have larger PATH-CORE scores than vertices in the periphery set. For $\kappa = 1.3$, the separation between core and peripheral vertices is not very clear. As we increase κ, the separation becomes clearer, and $\kappa = 1.8$ has a clear separation between core and peripheral vertices. As expected, larger differences between the edge probabilities $p_{cc} \geqslant p_{cp} \geqslant p_{pp}$ in the random-graph ensemble result in clearer separations between core and periphery sets.

   For some networks, it is sufficient to have a coreness measure that reflects the probability that a vertex is a core or peripheral vertex. In such a scenario, we view such scores as akin to centrality values [76]. In other situations, however, it is desirable to obtain a classification of a network's vertices as part of a core set or a periphery set. With this in mind, we let PATH-CORE(i) denote the PATH-CORE score of vertex i, and we assume
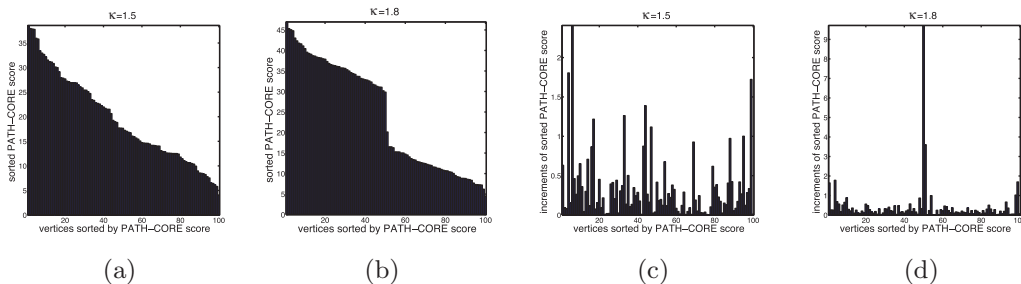
(a)　　　　　(b)　　　　　(c)　　　　　(d)

FIGURE 2. PATH-CORE scores, sorted in decreasing order, for the random-graph ensemble $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, $p_{pp} = p$, where $p = 0.25$, with (a) $\kappa = 1.5$ and (b) $\kappa = 1.8$. When the core–periphery structure is sufficiently prominent, it is possible to separate the vertices by sorting the vector of PATH-CORE scores and inferring the threshold between core and peripheral vertices by considering the largest increment that occurs between two consecutive entries in the vector of sorted PATH-CORE scores. We show the result for $\kappa = 1.5$ in panel (c) and the result for $\kappa = 1.8$ in panel (d). In panels (b) and (c) of Figure 1 and panels (a) and (b) of the present figure, the largest PATH-CORE score of a peripheral vertex is approximately 20, whereas the lowest PATH-CORE score of a core vertex is approximately 30 (the difference of 10 is revealed by the peak in panel (d) of this figure), and we obtain a clear discrete classification into a set of core vertices and a set of peripheral vertices.

without loss of generality that $\text{PATH-CORE}(1) \geqslant \text{PATH-CORE}(2) \geqslant \cdots \geqslant \text{PATH-CORE}(n-1) \geqslant \text{PATH-CORE}(n)$. Because the PATH-CORE score gives our calculation for the likelihood that a vertex is in the core set or periphery set (a high PATH-CORE suggests a core vertex), we are left with inferring what constitutes a good "cut" of PATH-CORE values to separate core vertices from peripheral ones. In other words, we seek to determine a threshold ξ such that we classify i as a core vertex if $\text{PATH-CORE}(i) \geqslant \xi$ and we classify i as a peripheral vertex if $\text{PATH-CORE}(i) < \xi$.

If the size $n_c = \beta n$ of the core set is known, then the problem becomes significantly easier, as we can select the top n_c vertices with the largest PATH-CORE scores and classify them as core vertices. That is, we set $a = n_c = \beta n$. However, in most realistic scenarios, the size of the core is not known in advance, and it should thus be inferred from the graph G (or from the graph ensemble) and the distribution of the PATH-CORE scores. One possible heuristic approach to obtain such a separation is to sort the vector of PATH-CORE scores in decreasing order and to infer a by searching for a large jump in the sizes of the vector elements. That is, one can seek a "natural" separation between high and low PATH-CORE scores (if one exists). An alternative approach is to detect two clusters in the vector of PATH-CORE scores using a clustering algorithm (such as k-means clustering). The examples in Figure 2 (which we generate from the random-graph ensemble $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = p$ for $\kappa \in \{1.3, 1.5, 1.8\}$) illustrate this heuristic very well, as there exists a natural cut point that corresponds to a PATH-CORE score of approximatively $a = 20$. This cut correctly assigns the first 50 vertices to the core set and the remaining 50 vertices to the periphery set. In our experiments, we fix $p = 0.25$ and use $\kappa \in [1, 2]$, which implies that $p_{cc}, p_{cp}, p_{pp} \in [0, 1]$.

Unfortunately, for "noisy" networks from this graph ensemble (and for many empirical networks), for which the edge probabilities p_{cc}, p_{cp}, and p_{pp} are not well-separated, the

aforementioned heuristic procedure can yield unsatisfactory results, so a more systematic approach is desirable. In Section 4, we thus introduce the FIND-CUT algorithm, which maximizes an objective function for partitioning a network into a set of core vertices and a set of peripheral vertices. Using the vector of PATH-CORE scores as an input—or, indeed, using any other vector of scores that reflects the likelihood that each vertex belongs to the core set—we consider a large number of possible values of the vector to attempt to find an optimal separation of vertices into a core set and a periphery set that maximizes the objective function in (4.4). See Section 4 for a discussion of this objective function and how we maximize it.

We present an explicit algorithm for PATH-CORE in Algorithm 5 (see Appendix A) for the case of unweighted and undirected graphs. This algorithm runs in $\mathcal{O}(m^2)$ time, where we recall that $m = |E|$ is the number of edges in the graph. Intuitively, this is the best that one can achieve (even when computing a PATH-CORE score for just a single vertex), because one must separately consider each graph $G \setminus e$ for all $e \in E$, and finding shortest paths between two vertices has a complexity of $\Theta(m)$. In Appendix A, we prove the above complexity results and provide pseudocode for the algorithm.

One potential way to drastically reduce the temporal complexity of the above approach is to sample edges from G via some random process and compute shortest paths only for pairs of adjacent vertices that use these sampled edges. An investigation of the trade-off between accuracy and computational efficiency of this method is beyond the scope of our paper, but it is an interesting direction for future research.

4 An objective function for detecting core–periphery structure

In this section, we introduce an objective function that is suitable for detecting core–periphery structure when there is exactly one core set of vertices and one periphery set. Our function bears some similarity to the rich-club coefficient [24], although a crucial difference is that it takes the connectivity of the core, the periphery, and the inter-connectivity between the two into account. (In contrast to rich clubs, our methodology allows low-degree vertices to be core vertices [92].) Using this objective function, we propose the FIND-CUT algorithm for partitioning the vertex set V into core and periphery sets. As an input, FIND-CUT takes a vector of scores that reflect the likelihood that each vertex belongs to a network's core set (the probability of belonging to the core set is higher for larger scores), and it attempts to find an optimal separation that maximizes the proposed objective function. That is, instead of trying to find a global optimum of the objective function, the algorithm FIND-CUT optimizes the objective function over all partitions in which the core vertices have higher likelihood scores than the periphery vertices. A fast general optimization algorithm for this objective function is likely very difficult to achieve, and it is beyond the scope of this paper. We believe that the construction of a suitable objective function brings three advantages. First, the subject of network community structure has benefited greatly from having objective functions to optimize [35, 74], and we expect similar benefits for investigations of core–periphery structure. Second, it allows a local-refinement search after the initial algorithm has been applied. (Such a local refinement follows the spirit of Kernighan–Lin vertex-swapping steps for community detection [64, 75] and gradient-descent refinement steps in non-convex optimization [62].)

Finally, it allows one to compare distinct methods by comparing the corresponding value of the objective function. Nevertheless, one has to proceed cautiously, as a value of an objective function need not provide a definitive answer, and it can be misleading [41, 70].

Before introducing an objective function for studying core–periphery structure, we first revisit a well-known graph-partitioning problem to highlight the similarity between the two situations. MIN-CUT, an instance of a graph-partitioning problem, is concerned with dividing a graph into two (similarly sized) subgraphs while minimizing the number of edges that are incident to vertices in both subgraphs. More generally, a large family of graph-partitioning problems seek to decompose a graph into k disjoint subgraphs (i.e., clusters) while minimizing the number of cut edges (i.e., edges with endpoints in different clusters). Given the number g of clusters, the g-way graph-partitioning problem searches for a partition $V_1, \ldots, V_g$ of the vertex set V that minimizes the number of cut edges,

$$\text{Cut}(V_1, \ldots, V_g) = \sum_{i=1}^{g} |E(V_i, \overline{V_i})|, \qquad (4.1)$$

where $\overline{X} = V \setminus X$ and the number of edges between $X \subset V$ and $Y \subset V$ is $|E(X, Y)| = \sum_{i \in X, j \in Y} A_{ij}$. However, it is well-known that trying to minimize $\text{Cut}(V_1, \ldots, V_g)$ favours cutting off weakly connected individual vertices from a graph and can thus lead to trivial partitions. To penalize clusters V_i of small size, Shi and Malik [80] suggested minimizing the normalized cut

$$\text{NCut}(V_1, \ldots, V_g) = \sum_{i=1}^{g} \frac{\text{Cut}(V_i, \overline{V_i})}{\text{SK}(V_i)}, \qquad (4.2)$$

where $\text{SK}(V_i) = \sum_{i \in V_i} d_i$ and d_i denotes the degree of vertex i in the original graph G.

A natural choice for an objective function to detect core–periphery structure is to maximize the number of edges between pairs of core vertices and also between core and peripheral vertices, while allowing as few edges as possible between pairs of peripheral vertices. In other words, our approach is complementary to that of the graph-cut objective function (4.1). However, instead of minimizing the number of cut edges across the core and periphery sets (i.e., across clusters), we maximize the connectivity between pairs of core vertices and between core and peripheral vertices, while minimizing the connectivity between pairs of peripheral vertices. We thus want to maximize

$$\text{CP-connectivity}(V_C, V_P) = E(V_C, V_C) + E(V_C, V_P) - E(V_P, V_P). \qquad (4.3)$$

Our aim is to find a partition $\{V_C, V_P\}$ of the vertex set V that maximizes CP-connectivity(V_C, V_P), under the constraint that $|V_C|, |V_P| \geqslant b$, where b is the minimum number of core or peripheral vertices (hence, $n - b$ is the maximum number of core or peripheral vertices) to avoid a large imbalance between the sizes of the core and periphery sets. In other words, we seek a balanced partition, and a higher value of b indicates a smaller difference between the sizes of the core and periphery sets. This constraint is required to avoid a trivial solution in which all of the vertices are placed in the core set. Furthermore, note that the objective function (4.3) has only one variable because of the constraint $E(V_C, V_C) + E(V_C, V_P) + E(V_P, V_P) = m$. In practice, we have found this approach to be rather unstable in the sense that (4.3) often attains its maximum

at $|V_C| = b$ or $|V_P| = b$. It thereby leads to disproportionately-sized sets of core and peripheral vertices compared to the "ground truth" in problems with planted core–periphery structure. For example, one can use the block model $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$, where we recall (see Section 3) that n_c (respectively, n_p) denotes the size of the core (respectively, periphery) sets, p_{cc} is the probability that there is an edge between a pair of core nodes, p_{cp} is the probability that there is an edge between a core node and a peripheral node, and p_{pp} is the probability that there is an edge between a pair of peripheral nodes. This situation is analogous to the trivial solution that one obtains for unconstrained graph-partitioning problems. We have been able to ameliorate this problem (though not remove it completely) by incorporating a normalization term in the spirit of the normalized cut function (4.2). Instead of maximizing the number of edges between core vertices and between core and peripheral vertices while minimizing the number of edges between peripheral vertices, we choose to maximize the edge *density* among core vertices and between core and peripheral vertices while minimizing the edge density among peripheral vertices. Finally, we also add a term to the objective function that penalizes imbalances between the sizes of the core and periphery sets (or penalizes a deviation from the expected proportion of core vertices) if such information is available. The maximization of our new objective function is over the set of all possible partitions of the vertex set into two disjoint sets (the core set V_C and the periphery set V_P). The function is

$$\text{CP-density}(V_C, V_P) = \frac{|E(V_C, V_C)|}{\text{Vol}(V_C, V_C)} + \frac{|E(V_C, V_P)|}{\text{Vol}(V_C, V_P)} - \frac{|E(V_P, V_P)|}{\text{Vol}(V_P, V_P)} - \gamma \left| \frac{|V_C|}{n} - \beta \right|, \quad (4.4)$$

where

$$\text{Vol}(X, Y) = \begin{cases} |X||Y|, & \text{if } X \neq Y \\ \frac{1}{2}|X|(|X| - 1), & \text{if } X = Y \end{cases}, \quad (4.5)$$

denotes the total possible number of edges between sets X and Y. In the penalty term, β denotes the prescribed fraction of core vertices in the graph (if it is known in advance), and γ tunes the sensitivity of the objective function to the size imbalance between the core and periphery sets. Note that β can either be prescribed in advance or construed as a parameter that guides the maximization towards a solution with a certain target size for the core set. For simplicity, we limit ourselves to the case $\gamma = 0$. We thereby assume no prior knowledge of the ratio between the numbers of core and peripheral vertices. In practice, however, we do implicitly assume a lower bound on the sizes of the core and periphery sets of vertices to ameliorate a "boundary effect" that yields solutions with a very small number of vertices in the core set or periphery set. If one explicitly wants to allow the possibility of a small set of core or peripheral vertices, then one can set $b = 0$ (see Algorithm 1). For some of our experiments on synthetic graphs in Section 7, we compare the performance of our proposed algorithms both when β is known and when it is unknown.

We summarize the FIND-CUT approach in Algorithm 1, and we remark that one can also add an iterative post-processing refinement step that is reminiscent of the gradient-descent algorithm [62] or of Kernighan–Lin vertex swaps [64, 75]. At each iteration, one can choose to move the vertex from the core set to the periphery set (or the other way around) that leads to the largest increase in the objective function (4.4). Alternatively,

---

Algorithm 1 FIND-CUT: Classifies the vertices of a graph G into a set V_C of core vertices and a set V_P of peripheral vertices based on a score associated to each vertex that reflects the likelihood that it is in the core.

Input: Vector of scores, $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{R}^n$, associated to the n vertices of a graph.

1: Sort the entries of the vector $\mathbf{s}$ in decreasing order. Assume without loss of generality that $s_1 \geqslant s_2 \geqslant \cdots \geqslant s_{n-1} \geqslant s_n$.

2: Let $X_C = \{1, \ldots, n_c\}$ and $Y_C = \{n_c + 1, \ldots, n\}$ for any $n_c \in \{1, \ldots, n\}$. Find the value n_c that maximizes the objective function given by equation (4.4) with $\gamma = 0$. That is, we find

$$\Phi^* = \frac{1}{n} \left[\max_{n_c \in \{b, \ldots, n-b\}} \left(\frac{|E(X_C, X_C)|}{\mathrm{Vol}(X_C, X_C)} + \frac{|E(X_C, Y_C)|}{\mathrm{Vol}(X_C, Y_C)} - \frac{|E(Y_C, Y_C)|}{\mathrm{Vol}(Y_C, Y_C)} \right) \right], \quad (4.6)$$

where b denotes a lower bound on the size of the core and periphery sets (which we use to avoid solutions with either a very small core set or a very small periphery set).

3: Define the core set $V_C = \{1, \ldots, n_c\}$ and the periphery set $V_P = \{n_c + 1, \ldots, n\}$.

---

if one wishes to maintain the current size of the core and periphery sets, then one can choose to swap a pair of vertices from their assignments (of core or periphery) that leads to the largest increase in the objective function.

5 LOWRANK-CORE: **Core–periphery detection via low-rank matrix approximation**

Another approach for detecting core–periphery structure in an unweighted network[2] is to interpret its adjacency matrix as a perturbation of a low-rank matrix. Perturbations of low-rank matrices were used recently in Ref. [5] for classifying networks and identifying small-world structure—by capturing the dense connectivity of nodes within communities and the sparse connectivity between communities—and this type of an approach should also be useful for studying core–periphery structure.

Consider the block model

$$A_0 = \left[\begin{array}{c|c} \mathbf{1}_{n_c \times n_c} & \mathbf{1}_{n_c \times n_p} \\ \hline \mathbf{1}_{n_p \times n_c} & \mathbf{0}_{n_p \times n_p} \end{array} \right], \quad (5.1)$$

which assumes (1) that core vertices are fully connected among themselves and to all vertices in the periphery set and (2) that no edges exist between any pair of peripheral vertices. We use the notation $\mathbf{1}_{n_1 \times n_2}$ to represent an $n_1 \times n_2$ matrix in which every entry is a 1, and we use the analogous notation $\mathbf{0}_{n_1 \times n_2}$ for an $n_1 \times n_2$ matrix in which every entry is a 0. The block model in equation (5.1) corresponds to an idealized block model that Borgatti and Everett [13] employed in a discrete notion of core–periphery structure. The rank of the matrix A_0 is 2, as any 3×3 submatrix has at least two identical rows or columns. Consequently, $\det(A_0) = 0$. Alternatively, when the core and periphery sets have the same size, $n_c = n_p$ with $n = n_c + n_p$, one can write the matrix A_0 as the following

---

[2] We have not investigated this approach for weighted networks. It is an interesting direction to pursue in future studies.
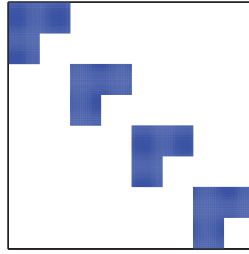
FIGURE 3. A block model with $g = 4$ diagonal blocks that are each of the form of the block model in equation (5.1).

tensor product of matrices:

$$\tilde{A}_0 = \left[\begin{array}{c|c} \mathbf{1}_{n_c \times n_c} & \mathbf{1}_{n_c \times n_c} \\ \hline \mathbf{1}_{n_c \times n_c} & \mathbf{0}_{n_c \times n_c} \end{array}\right] = R \otimes \mathbf{1}_{n_c \times n_c}, \qquad R = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{5.2}$$

We obtain the eigenvalues of $\tilde{A}_0$ by taking a direct product of the sets of eigenvalues of R and $\mathbf{1}_{n_c \times n_c}$. These eigenvalues are

$$\left\{ \frac{1 - \sqrt{5}}{2}, \frac{1 + \sqrt{5}}{2} \right\} \otimes \left\{ n_c, 0^{(n_c - 1)} \right\} = \left\{ \left(n_c \frac{1 \pm \sqrt{5}}{2} \right), 0^{(n_c - 2)} \right\},$$

where a superscript denotes the multiplicity of an eigenvalue.

The simplistic block models in equations (5.1) and (5.2) assume that a network has exactly one core set and one periphery set. Consequently, the block-model matrix has a rank of 2. The matrix rank is higher for more complicated core–periphery block models. For example, the block model in Figure 3 has a global community structure—there are $g = 4$ communities, which each correspond to a block in the block-diagonal matrix—and a local core–periphery structure (because each community has a core–periphery structure). As indicated in Ref. [76], one can also construe such a structure (by permuting the rows and columns of the matrix) as having a global core–periphery structure and a local community structure.

Let $B_g(A_0)$ denote a "hierarchical" ensemble of size $n \times n$ that is composed of g diagonal blocks that are of each of size $l \times l$ (thus, $n = lg$), where each diagonal block is of the form of the block model in equation (5.1). If we let λ_1 and λ_2 denote the two non-zero eigenvalues of A_0 and let I_g denote the identity matrix of size g, then we can also write $B_g(A_0)$ as a tensor product of matrices

$$B_g(A_0) = I_g \otimes A_0, \quad \text{with eigenvalues} \quad \begin{aligned} \{B_g(A_0)\} &= \{1^{(g)}\} \otimes \{\lambda_1, \lambda_2, 0^{(l-2)}\} \\ &= \left\{ \lambda_1^{(g)}, \lambda_2^{(g)}, 0^{(n-2g)} \right\}. \end{aligned} \tag{5.3}$$

Therefore, in the simplistic scenario in which each diagonal block has one core set and one periphery set (and thus has rank 2), the rank of $B_g(A_0)$ is $2g$.

Motivated by the low-rank structure of the above block-model networks, it is useful to consider the possibility of recovering a network's unknown structure using a simple low-rank projection of its adjacency matrix. For the remainder of this section, we focus on the simple core–periphery structure whose rank-2 block model is given by equation (5.1) (with one core set and one periphery set). In practice, we construe the adjacency matrix A of an observed graph G as a low-rank perturbation of the block model A_0. In other words, we decompose A as

$$A = A_0 + W, \qquad (5.4)$$

where W is a "noise matrix" whose entries $\{-1, 0, 1\}$ are determined by a mixture model [58] that involves block-model parameters. The entries of W are

$$W_{ij} = \begin{cases} -1, & \text{with probability } 1 - p_{cc} & (\text{i.e., if } i, j \in V_C), \\ -1, & \text{with probability } 1 - p_{cp} & (\text{i.e., if } i \in V_C \text{ and } j \in V_P), \\ 1, & \text{with probability } p_{pp} & (\text{i.e., if } i, j \in V_P), \\ 0, & \text{otherwise}. \end{cases} \qquad (5.5)$$

Note that W is a random block-structured matrix with independent entries, and its expected value is the rank-2 matrix with entries

$$\mathbb{E}(W_{ij}) = \begin{cases} p_{cc} - 1, & \text{if } i, j \in V_C, \\ p_{cp} - 1, & \text{if } i \in V_C \text{ and } j \in V_P, \\ p_{pp}, & \text{if } i, j \in V_P. \end{cases} \qquad (5.6)$$

To "denoise" the adjacency matrix A and recover the structure of the block model, we consider its top two eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2\}$, whose corresponding two largest (in magnitude) eigenvalues are $\{\lambda_1, \lambda_2\}$, and we compute the rank-2 approximation

$$\hat{A} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix}. \qquad (5.7)$$

As A more closely approximates the block model, which we can construe as a sort of "null model", the spectral gap between the top two largest eigenvalues and the rest of the spectrum becomes larger (as illustrated by the plots in the second column of Figure 4). In other words, as the amount of noise in (i.e., the perturbation of) the network becomes smaller, the top two eigenvalues $\{\lambda_1, \lambda_2\}$ become closer to the eigenvalues $\{\lambda_1 = n_c(\frac{1+\sqrt{5}}{2}), \lambda_2 = n_c(\frac{1-\sqrt{5}}{2})\}$ of the block model.

To illustrate the effectiveness of our low-rank projection in computing a coreness score, we consider two synthetically generated networks based on the SBM that we introduced previously. We use the edge probabilities $(p_{cc}, p_{cp}, p_{pp}) = (0.7, 0.7, 0.2)$ and $(p_{cc}, p_{cp}, p_{pp}) = (0.8, 0.6, 0.4)$. In the left column of Figure 4, we show their corresponding adjacency matrices. The spectrum, which we show in the middle column, reveals the rank-2 structure of the networks. In the second example (which we show in the bottom row of the figure), the large amount of noise causes the second-largest eigenvalue to merge with the bulk of the spectrum.

We then use the denoised matrix $\hat{A}$ to classify vertices as part of the core set or the periphery set by considering the degree of each vertex (i.e., the row sums of $\hat{A}$). We

binarize $\hat{A}$ by setting its entries to 0 if they are less than or equal to 0.5 and setting them to 1 if they are larger than 0.5, and we denote the resulting binarized matrix by $\hat{A}_t$. We remark that, following the rank-2 projection, we observe in practice that all entries of $\hat{A}$ lie in the interval $[0, 1]$. (We have not explored the use of other thresholds besides 0.5 for binarizing $\hat{A}$.) In the right column of Figure 4, we show the recovered matrix $\hat{A}_t$ for our two example networks. Note in both examples that the denoised matrix $\hat{A}_t$ resembles the core–periphery block model $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ much better than the initial adjacency matrix A. Finally, we compute the degree of each vertex in $\hat{A}_t$, and we call these degrees the LowRank-Core scores of the vertices. We use the LowRank-Core scores to classify vertices as core vertices or peripheral vertices. If one knows the fraction β of core vertices in a network, then we choose the top βn vertices with the largest LowRank-Core score as the core vertices. Otherwise, we use the vector of LowRank-Core scores as an input to the Find-Cut algorithm that we introduced in Section 4. Although a theoretical analysis of the robustness to noise of our low-rank approximation for core–periphery detection is beyond the scope of the present paper, we expect that results from the matrix-perturbation literature, such as Weyl's inequality and the Davis–Kahan $\sin(\Theta)$-theorem [12], as well results on low-rank deformations of large random matrices [11] (analogous to the results of Féral and Péché on the largest eigenvalue of rank-1 deformations of real, symmetric random matrices [34]) could lead to theoretical results that characterize the sparsity and noise regimes for which the rank-2 projection that we proposed above is successful at separating core and peripheral vertices. A possible first step in this direction would be to consider a simplified version of the graph ensemble $G(p_{cc}, p_{cp}, p_{pp}, n_p, n_c)$ by setting $p_{cc} = p_{cp} = 1 - p_{pp} = 1 - \eta$, where $\eta \in (0, 1)$.

Algorithm 2 LowRank-Core: Detects core–periphery structure in a graph based on a rank-2 approximation.

Input: Adjacency matrix A of the simple graph $G = (V, E)$ with n vertices and m edges.
1: Compute $\{\lambda_1, \lambda_2\}$, the top two largest (in magnitude) eigenvalues of A, together with their corresponding eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2\}$.
2: Compute $\hat{A}$, a rank-2 approximation of A, as indicated in equation (5.7).
3: Threshold the entries of $\hat{A}$ at 0.5 (so that entries strictly above 0.5 are set to 1 and all other entries are set to 0), and let $\hat{A}_t$ denote the resulting graph.
4: Compute the LowRank-Core scores as the degrees of $\hat{A}_t$.
5: If the fraction β of core vertices is known, identify the set of core vertices as the top βn vertices with the largest LowRank-Core scores.
6: If β is unknown, use the vector of LowRank-Core scores as an input to the Find-Cut algorithm in Algorithm 1.

---

6 Laplacian-based core–periphery detection

In this section, we explore the utility of employing Laplacian eigenvectors for detecting core–periphery structure. (As with Path-Core, this approach is applicable to either unweighted or weighted graphs.) The *combinatorial Laplacian* matrix associated to the adjacency matrix A of a graph G is $F = D - A$, where D is a diagonal matrix and D_{ii} denotes the degree of vertex i in the case of an unweighted graph. For a weighted
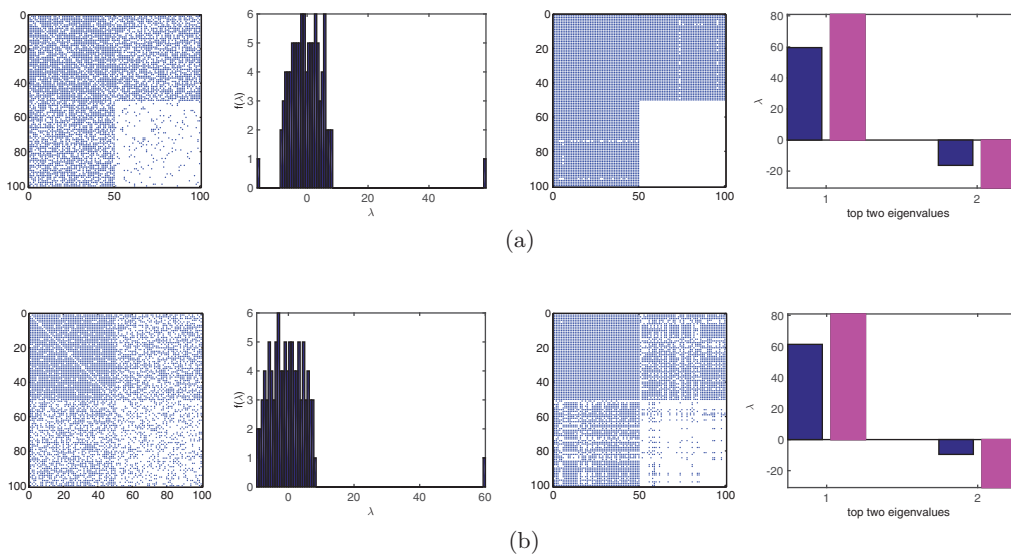
FIGURE 4. (Column 1) Original adjacency matrices A from the stochastic block model (SBM) $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with edge probabilities p_{cc} for edges between two core vertices, p_{cp} for edges between core vertices and peripheral vertices, and p_{pp} for edges between two peripheral vertices. (Column 2) Histogram $f(\lambda)$ of the eigenvalues of the original adjacency matrices A. (Column 3) The matrices $\hat{A}_t$ that we obtain after the rank-2 projection and thresholding. (Column 4) Bar plot of the top two eigenvalues $\left\{\lambda_1 = n_c\left(\frac{1+\sqrt{5}}{2}\right), \lambda_2 = n_c\left(\frac{1-\sqrt{5}}{2}\right)\right\}$ of the block model (5.2) (blue/dark) versus the top two eigenvalues of the SBM $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ (pink/light), averaged over 100 instantiations. We use the probabilities (a) $p_{cc} = 0.7, p_{cp} = 0.7, p_{pp} = 0.2$ and (b) $p_{cc} = 0.8, p_{cp} = 0.6, p_{pp} = 0.4$.

graph, D_{ii} denotes the sum of the weights associated to vertex i. The solutions of the generalized eigenvalue problem $F\mathbf{x} = \lambda D\mathbf{x}$ are related to the solutions of the eigenvalue problem $L\mathbf{x} = \lambda\mathbf{x}$, where $L = D^{-1}A$ is often called the *random-walk Laplacian* of G. Using $L = I - D^{-1}F$, one can write the random-walk Laplacian in terms of the combinatorial Laplacian. Because L is a row-stochastic matrix, one can interpret it as a transition probability matrix of a Markov chain whose states are the vertices of G. In this interpretation, the matrix element L_{ij} denotes the transition probability that a random walker jumps from vertex i to vertex j in a single step. If the pair $(\lambda, \mathbf{v})$ is an (eigenvalue, eigenvector) solution to $L\mathbf{x} = \lambda\mathbf{x}$, then $(1 - \lambda, \mathbf{v})$ is a solution to $F\mathbf{x} = \lambda D\mathbf{x}$. The top[3] eigenvectors of the random-walk Laplacian define the coarsest modes of variation (i.e., slowest modes of mixing) in a graph, and they have a natural interpretation in terms of a random walk on the graph (and thus as a toy model of a conservative diffusion process). There exists a rich literature in the machine-learning, data-analysis, and image-processing communities [10, 22, 23, 59, 79, 83] on the use of such eigenvectors for tasks like clustering, ranking, image segmentation, and data visualization.

[3] The "top" eigenvectors of the random-walk Laplacian L are the eigenvectors that correspond to the largest eigenvalues of L. That is, these are the eigenvalues closest to $\lambda_1 = 1$, the largest eigenvalue of L. The "bottom" eigenvectors of L correspond to the smallest eigenvalues of L. The eigenvalues $\lambda_1 = 1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$ of L satisfy $|\lambda_i| \leqslant 1$ for all $i \in \{1, \ldots, n\}$.

For core–periphery detection, it is useful to consider the bottom eigenvector of the associated random-walk Laplacian. Considering the block model in equation (5.1) or the generalized block model $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ (see the depiction in Table 1) with $p_{cc} \approx p_{cp} < p_{pp}$, the task of finding core–periphery structure in a given graph G amounts to trying to detect a dense connected component between the peripheral vertices in the complement graph $\bar{G}$ (in which the 0 non-diagonal entries of A become 1, and the 1 entries become 0), as such vertices have many non-edges between them in the original graph. If $p_{cc} \approx p_{cp} < p_{pp}$ (i.e., the above scenario) and there exists a single densely connected component in a given graph—such as in examples (a) and (b) in Figure 5— the eigenvector that corresponds to the second-largest (in magnitude) eigenvalue of the associated random-walk Laplacian provides an accurate separation of the vertices in the dense component from the rest of the graph. The complement of the block-model graph has a periphery component of size n_p that is fully connected (i.e., it is K_{n_p}, the complete graph on n_p vertices), a core component without any edges between pairs of core vertices, and no edges between core and peripheral vertices. In practice, $\bar{G}$ is a perturbed version of the above complement block model; that is, the peripheral vertices are very well-connected among themselves, and there are few core–core and core–periphery connections. Our task then amounts to identifying a well-connected "community" of peripheral vertices. In other words, we have replaced the problem of identifying a core set and periphery set in G with the problem of finding the periphery set in $\bar{G}$, for which we can use methods from the large set of available techniques for community detection [35, 74].

In many applications, the initial graph G is rather sparse, and the above approach thus has the drawback that the complement graph $\bar{G}$ is very dense, which significantly increases the time that is necessary for the computational task of identifying communities [17] (though we note that we only seek to identify a single dense subgraph rather than a graph's entire community structure). As we discussed above, one way to find a dense subgraph of an initial graph is to use the first non-trivial eigenvalue (i.e., the second-largest eigenvalue) of the random-walk Laplacian. In Figure 5(a), we show an example of such a computation. In this case, we start with a block-model graph from $G(p_{cc} = 0.8, p_{cp} = 0.2, p_{pp} = 0.2, n_c, n_p)$, for which the first non-trivial eigenvalue (see the second column) clearly separates the planted dense subgraph from the rest of the network. In the eigenvector computation for the random-walk Laplacian, note that every iteration of the power method is linear in the number of edges in the graph, and the number of iterations is strictly greater than $\mathcal{O}(1)$ because it depends on the spectral gap. For sparse graphs G, the complement $\bar{G}$ is a dense graph, which significantly increases the computational effort needed to find eigenvectors. Instead of working in the complement space, we turn our attention to the other end of the spectrum and consider the smallest eigenvalue of the random-walk Laplacian. Recall that all of the eigenvalues of the random-walk Laplacian are less than or equal to 1 in magnitude [20].

We now focus on the combinatorial Laplacian $F = D - A$. Let $\bar{F}$ denote the combinatorial Laplacian associated to the graph $\bar{G}$. Note that $\bar{A} = J_n - A - I_n$, where J_n denotes the matrix of size $n \times n$ whose entries are all 1 and I_n is the $n \times n$ identity matrix. Additionally, $\bar{D} = (n-1)I_n - D$. A well-known relationship [20] between the combinatorial Laplacian of a graph and that of its complement is given by

$$\bar{F} = \bar{D} - \bar{A} = (n-1)I_n - D - (J_n - A - I_n) = nI_n - J_n - F. \tag{6.1}$$
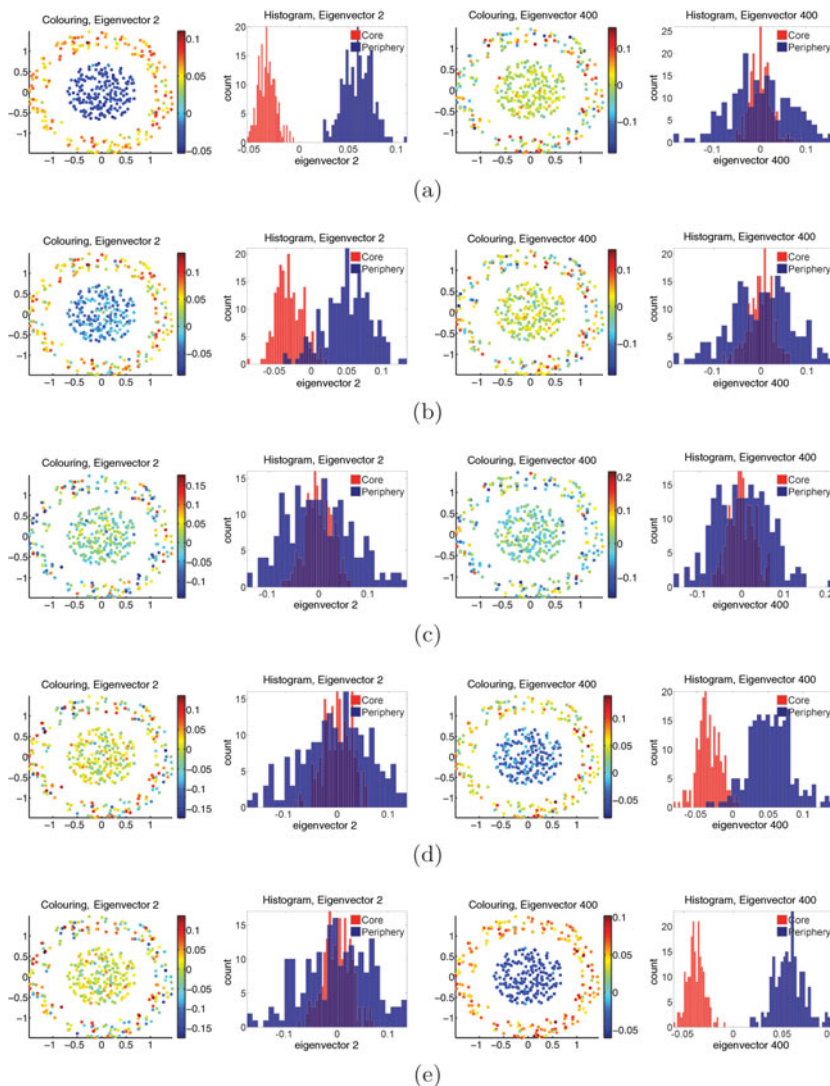
FIGURE 5. Our simulations illustrate the interplay between the top and bottom parts of the spectrum of the random-walk Laplacian matrix L as a network transitions from a block model with block-diagonal "community structure" to a block model with core–periphery structure. Each row uses one network from the SBM $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $n = 400$ vertices (with 200 core and 200 peripheral vertices) with a fixed core–core interaction probability $p_{cc} = 0.8$, a fixed periphery–periphery interaction probability $p_{pp} = 0.3$, and a varying core–periphery interaction probability $p_{cp} \in [0.3, 0.7]$. We vary p_{cp} in increments of 0.1, so the top row has $p_{cp} = 0.3$, the second row has $p_{pp} = 0.4$, and so on. The first and third columns give a colouring of a two-dimensional visualization of the graph vertices; the core vertices are contained in a disc that is centred at the origin, and the peripheral vertices lie on a ring around the core vertices. The second and fourth columns, respectively, show histograms of the entries of the eigenvectors $\mathbf{v}_2$ and $\mathbf{v}_{400}$. These eigenvectors correspond, respectively, to the largest (non-trivial) and smallest eigenvalues of the associated random-walk Laplacian matrix. The red/light colour indicates core vertices, and the blue/dark colour indicates peripheral vertices. In Figure 6, we plot the spectrum associated to each of the above six networks. The probabilities are (a) $p_{cc} = 0.8$, $p_{cp} = 0.3$, $p_{pp} = 0.3$; (b) $p_{cc} = 0.8$, $p_{cp} = 0.4$, $p_{pp} = 0.3$; (c) $p_{cc} = 0.8$, $p_{cp} = 0.5$, $p_{pp} = 0.3$; (d) $p_{cc} = 0.8$, $p_{cp} = 0.6$, $p_{pp} = 0.3$; and (e) $p_{cc} = 0.8$, $p_{cp} = 0.7$, $p_{pp} = 0.3$.

If **x** is an eigenvector of F (other than the trivial eigenvector $\mathbf{1}_n$) with $\mathbf{x} \perp \mathbf{1}_n$ (which implies that $J\mathbf{x} = \mathbf{0}$) and associated eigenvalue λ, then **x** is also an eigenvector of $\bar{F}$ (with associated eigenvalue $n - \lambda$). A result due to Kelmans [48–50], that connects the characteristic polynomial of the combinatorial Laplacian matrix of G to that of its complement implies that

$$\lambda_j(\bar{F}) = n - \lambda_{n+2-j}(F) \qquad \text{for all} \quad j \in \{2, \dots, n\}. \tag{6.2}$$

Equation (6.2) relates the eigenvalues of the combinatorial Laplacian of G to those of its complement $\bar{G}$. In other words, the spectrum exhibits a certain symmetry, and questions regarding $\lambda_{n+2-j}(F)$ of a graph are equivalent to questions about $\lambda_j(\bar{F})$ of its complement. Furthermore, keeping in mind the usefulness of the second-largest eigenvector of the combinatorial Laplacian, we stress that questions involving $\lambda_2(\bar{F})$ (i.e., the case $j = 2$) are equivalent to questions involving $\lambda_n(F)$.

In practice, none of the eigenvectors of the combinatorial Laplacian are able to distinguish a coherent core set and periphery set in a graph (or a single community in the graph's complement). We calculate the top and bottom eigenvectors (and intermediate ones) of the combinatorial Laplacian and find that none of them captures the distinction between core and periphery sets. Instead, we are able to effectively separate core and periphery sets if we use the random-walk Laplacian L, with the goal of identifying a dense subgraph in the complement graph $\bar{G}$. To do this, one can calculate the second eigenvector $\bar{\mathbf{v}}_2$ of its associated Laplacian $\bar{L}$. However, because graphs are sparse in most applications, considering the complement tends to yield a dense graph, which could render computations prohibitive for large n. Instead, we propose to use the following approach. Motivated by the analogy in the beginning of this section and the interplay between the bottom eigenvalues of a graph and the top eigenvalues (and their associated eigenvectors) of the graph's complement for the combinatorial Laplacians F and $\bar{F}$, we propose to use the bottom eigenvalue (and its associated eigenvector) of the random-walk Laplacian L associated with our initial graph G. The downside of working with the random-walk Laplacian L is that (to the best of our knowledge) there does not exist a statement similar to equation (6.2) that makes an explicit connection between the random-walk Laplacian eigenvalues of a graph and those of its complement. In Appendix B, we explain that such a symmetry exists for the random-walk Laplacian only under certain restrictive conditions. When these conditions are not met, we still make an implicit analogy between the random-walk Laplacian eigenvalues of a graph and those of its complement, but we do not know how to characterize this relationship mathematically.

In Algorithms 3 and 4, we summarize the main steps of two viable algorithms for core–periphery detection using the random-walk Laplacian of a graph. The only difference between Algorithms 3 and 4 is as follows. The former uses the entries of $\mathbf{v}_n$ (the bottom eigenvector that corresponds to the smallest algebraic[4] eigenvalue as an input to the FIND-CUT algorithm to infer an optimal separation of the vertices into core and periphery sets by maximizing the objective function (4.4). By contrast, in Algorithm 4, the same bottom eigenvector $\mathbf{v}_n$ of the random-walk Laplacian provides an implicit threshold (i.e.,

---

[4] Because all of the random-walk Laplacian eigenvalues are real and no larger than 1 in magnitude, the smallest algebraic eigenvalue corresponds to the smallest real eigenvalue.
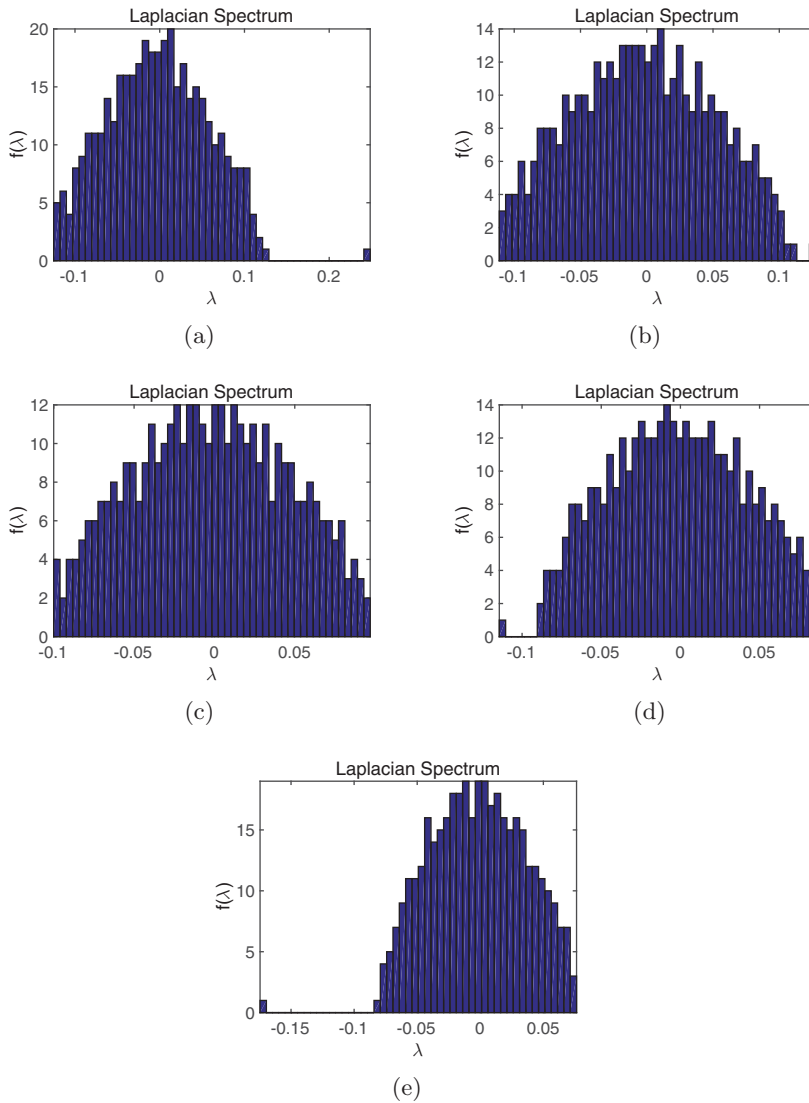
FIGURE 6. Spectra $f(\lambda)$ of random-walk Laplacian matrices' eigenvalues λ for several instances of the SBM $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$. In Figure 5, we plotted histograms of the eigenvectors corresponding to the smallest and second-largest eigenvalues for these matrices. We use the probabilities (a) $p_{cc} = 0.8$, $p_{cp} = 0.3$, $p_{pp} = 0.3$; (b) $p_{cc} = 0.8$, $p_{cp} = 0.4$, $p_{pp} = 0.3$; (c) $p_{cc} = 0.8$, $p_{cp} = 0.5$, $p_{pp} = 0.3$; (d) $p_{cc} = 0.8$, $p_{cp} = 0.6$, $p_{pp} = 0.3$; and (e) $p_{cc} = 0.8$, $p_{cp} = 0.7$, $p_{pp} = 0.3$.

the value 0), and one is able to classify each vertex as part of a core set or a periphery set by considering the sign of each entry. To choose a global sign, we multiple by -1 if necessary to maximize the objective function (4.4) and ensure that the positive entries correspond to core vertices. (If $\mathbf{v}_n$ is an eigenvector of L, then so is $-\mathbf{v}_n$.)

Algorithm 3 LAP-CORE: Detects core–periphery structure in a graph using a core score that is based on the eigenvector corresponding to the smallest non-zero eigenvalue of the associated random-walk graph Laplacian.

Input: Adjacency matrix A of the simple graph $G = (V, E)$ with n vertices and m edges.

1: Compute the random-walk Laplacian $L = D^{-1}A$, where D is a diagonal matrix with elements $D_{ii} = \sum_{j=1}^{n} A_{ij}$ given by the strength (i.e., the sum of weights of the edges incident to the vertex) of vertex i for each i.

2: Compute λ_n, which denotes the smallest non-zero eigenvalue of L, and its corresponding eigenvector $\mathbf{v}_n$. The eigenvector components give the LAP-CORE scores of the vertices.

3: If β is known, identify the set of core vertices as the top βn vertices with the largest LAP-CORE scores.

4: If β is unknown, use the vector of LAP-CORE scores as an input to the FIND-CUT algorithm.

Algorithm 4 LAPSGN-CORE: Detects core–periphery structure in a graph using the signs of the components of the eigenvector corresponding to the smallest non-zero eigenvalue of the associated random-walk graph Laplacian.

Input: Adjacency matrix A of the simple graph $G = (V, E)$ with n vertices and m edges.

1: Compute the random-walk Laplacian $L = D^{-1}A$.

2: Compute λ_n, which is the smallest non-zero eigenvalue of L, and its corresponding eigenvector $\mathbf{v}_n$. The eigenvector components give the LAP-CORE scores of the vertices.

3: Set $z_i = \text{sign}(\mathbf{v}_n(i))$ for $i \in \{1, \ldots, n\}$. Because the eigenvector $\mathbf{v}_n$ is determined up to a global sign change, do the following:

4: Let vertex $u_i \in V_C$ if $z_i \geqslant 0$, and otherwise let $u_i \in V_P$. Let η_1 denote the resulting value of the objective function (4.4).

5: Let vertex $u_i \in V_C$ if $z_i \leqslant 0$, and otherwise let $u_i \in V_P$. Let η_2 denote the resulting value of the objective function (4.4).

6: If $\eta_1 > \eta_2$, let the final solution be $u_i \in V_C$ if $z_i \geqslant 0$; otherwise, let $u_i \in V_P$.

7: If $\eta_2 > \eta_1$, let the final solution be $u_i \in V_C$ if $z_i \leqslant 0$; otherwise, let $u_i \in V_P$.

8: If $\eta_1 = \eta_2$, there is no clear separation of the network vertices into core and periphery sets.

To illustrate the above interplay between the top and bottom parts of the spectrum of the random-walk Laplacian matrix, we consider the SBM $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$, where we fix the core–core interaction probability $p_{cc} = 0.8$ and the periphery–periphery interaction probability $p_{pp} = 0.3$, but we vary the core–periphery interaction probability $p_{cp} \in [0.3, 0.7]$ in increments of 0.1. The goal of these numerical experiments, whose results we show in Figure 5, is to demonstrate the ability of the bottom eigenvector $\mathbf{v}_n$ of L to reveal a core–periphery separation when one exists. To help visualize our results, we also employ a two-dimensional representation of the network vertices in which the core vertices (i.e., the vertices in the set V_C) are concentrated within a disc centred at the origin and the peripheral vertices (i.e., the vertices in the set V_P) lie on a circular ring around the core vertices. In Figure 6, we plot the spectrum of the random-walk Laplacian associated

to each of the p_{cp} values in the above experiment. Note that we disregard the trivial eigenvector $\mathbf{v}_1 = \mathbf{1}_n$ that corresponds to the trivial eigenvalue $\lambda_1 = 1$ of L.

For small values of p_{cp} (e.g., $p_{cp} = 0.3$ or $p_{cp} = 0.4$), the network does not exhibit core–periphery structure. Instead, it has a single community that is represented by the densely connected graph of vertices in the set V_C. As expected, the eigenvector $\mathbf{v}_2$ is able to highlight the separation between the V_C and V_P vertices very well, whereas the bottom eigenvector $\mathbf{v}_n$ is not particularly helpful. For $p_{cp} = 0.5$, neither of the two eigenvectors above are able to capture the separation between V_C and V_P. However, as p_{cp} increases to $p_{cp} = 0.6$ and $p_{cp} = 0.7$, such that we are closer to the idealized block model in (5.1), there now exists a densely connected subgraph of V_P in the complement graph $\bar{G}$. Instead of using the top non-trivial eigenvector $\bar{\mathbf{v}}_2$ of $\bar{L}$, we use the eigenvector $\mathbf{v}_n$ that corresponds to the smallest eigenvalue λ_n of G, as this eigenvector is able to highlight core–periphery structure in G. In Figure 6(a), we show that there is a clear separation between λ_2 and the bulk of the spectrum. Similarly, Figure 6(e) illustrates a clear separation between λ_n and the bulk of the spectrum. For intermediate values of p_{cp}, such a spectral gap is significantly smaller or even non-existent.

In conclusion, for this approach to core–periphery detection, one should consider the eigenvector $\mathbf{v}_n$ as in Algorithm 4, whereas one should use the eigenvector $\mathbf{v}_2$ when trying to detect a single dense community. (As illustrated in Figure 6, one can also use the spectrum of the random-walk Laplacian for guidance.) The former scenario is hinted by the presence of a spectral gap to the left of the bulk of the distribution, and the latter scenario is hinted by a spectral gap to the right of the bulk of the distribution.

7 Numerical experiments

In this section, we conduct a series of numerical experiments to compare different methods for detecting core–periphery structure and to assess the robustness of our methods to perturbations of a network. In Section 7.1, we examine synthetic networks with a global community structure and local core–periphery structure. In Section 7.2, we apply our various methods for detecting core–periphery structure to several empirical data sets. In Appendix C, we examine networks with "planted" high-degree vertices in the periphery. We are motivated by Ref. [92], which pointed out that it is often necessary to use more sophisticated ideas than a degree-based classification to assign nodes to core and periphery sets, although there are some situations (e.g., sometimes when there is very strong core–periphery structure) when it is sufficient to simply use node degree. Throughout this section, we use the term DEGREE-CORE to refer to the method of detecting core–periphery structure by simply computing the vertex degrees and then applying the FIND-CUT method. In doing so, we assume that we have knowledge of the "boundary" sizes and thereby assume that there is a lower bound on the sizes of the core and periphery sets.

As we illustrate in Figure 7, the LAPSGN-CORE method (see Algorithm 4) yields the same results whether or not we impose lower bounds on the sizes of the core and periphery sets, as it does not rely on information about the size of the core set. As we discussed in Section 6, it depends only on the sign of the entries of the top eigenvector of L. All of the other methods that we examine suffer from a "boundary effect," as the FIND-CUT algorithm finds a global optimum at (or very close to) the boundary of the search interval.
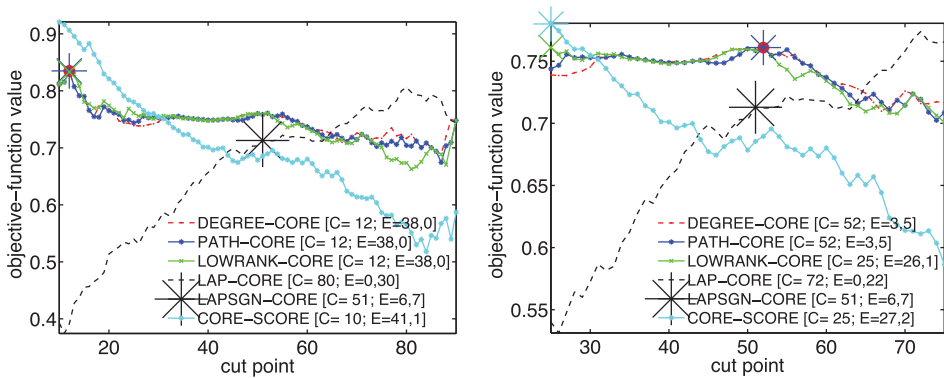
FIGURE 7. Comparison of methods for detecting core–periphery structure for a graph from the ensemble $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $n = 100$ vertices (and, in particular, $n_c = 50$ core vertices and $n_p = 50$ peripheral vertices) and edge probabilities $(p_{cc}, p_{cp}, p_{pp}) = (0.5, 0.5, 0.27)$ for the objective function in equation (4.6). We assume a minimum size for the core and periphery sets of at least (left) 10 vertices and (right) 25 vertices. We mark the cut points on the curves with a large asterisk for LapSgn-Core and using symbols whose colours match the colours of the corresponding curves for the other methods. The cut point refers to the number of core vertices. In the legends, C denotes the size of the core set, and $\mathbf{E} = (y_1, y_2)$ denotes the corresponding 2-vector of errors. The first component of $\mathbf{E}$ indicates the number of core vertices that we label as peripheral vertices, and the second indicates the number of peripheral vertices that we label as core vertices.

When β is known, we are planting core and periphery sets of known sizes, so we can examine the number of false-positive errors (i.e., vertices incorrectly assigned to the core set) and false-negative errors (i.e., vertices incorrectly assigned to the periphery set) for the various methods for detecting core–periphery structure. If we enforce a minimum size of 20 for the core and periphery sets, we find that LapSgn-Core is the only method that yields satisfactory results from this perspective, because all other methods find a maximum of the objective function that lies close to the boundary. When we increase the lower bound of the core and periphery sets from 10 each to 25 each, the Degree-Core and Path-Core methods yield very good results (in terms of the numbers of false positives and false negatives), followed by LapSgn-Core, Lap-Core, LowRank-Core, and Core-Score.[5] When the fraction of vertices that belong to the core is known, then Degree-Core, Path-Core, and LowRank-Core again yield the best results, followed by LapSgn-Core, Lap-Core, and Core-Score.

   We again evaluate the methods in terms of the number of false positives and false negatives. One can increase the accuracy of the methods to detect core–periphery structure by considering other local maxima of the objective function (4.4), especially if one is searching further away from the boundary. However, for these examples, the LapSgn-Core and Core-Score methods still yield unsatisfactorily results even when considering additional local maxima. Interestingly, their objective functions are monotonic (increasing for the former and decreasing for the latter) with respect to the vector of sorted scores. After assigning vertices to a core set or peripheral set using any of the methods above, one

---

[5]   Several versions of CoreScore were introduced in Ref. [76]. In our comparisons in the present paper, we use the same version that we employed in Ref. [56].

can also add a post-processing step in the spirit of either the gradient-descent refinement step in non-convex optimization [62] or Kernighan–Lin vertex swaps in community detection [64, 75].

The critical eye may object that a separation based on vertex degree yields results that are as good as the other best-performing methods. However, Zhang et al. [92] pointed out and discussed why relying on vertex degree tends to be suboptimal for separating core and periphery sets, although DEGREE-CORE may not actually perform worse than more sophisticated methods when there is a weak core–periphery structure and can perform reasonably well when a network's core and periphery are separated very strongly. For pronounced core–periphery structure that is neither too weak nor too strong (i.e., in the most relevant situation for applications [92]), one needs to use methods that are more sophisticated than simply considering vertex degrees. Ref. [76] also includes a salient discussion of examining a network's core–periphery structure simply by computing vertex degrees. To illustrate the sensitivity of the DEGREE-CORE method to the presence of high-degree peripheral vertices, we perform a pair of numerical experiments in which we purposely plant high-degree vertices in the periphery set (see Appendix C). In these experiments, the LAPSGN-CORE method yields the smallest number of errors, whereas DEGREE-CORE is one of the worst performers. In addition, one can see from Table D1, which gives the Pearson and Spearman correlation coefficients for various coreness measures, that the results of our proposed methods are often only moderately correlated with DEGREE-CORE, and they can thus return solutions that differ significantly from naive separation based on vertex degree. From the perspective of applications, we note the work of Kitsak et al. [51] on the identification of influential spreaders in networks. Kitsak et al. argued that the position of a vertex relative to the organization of a network determines its spreading influence to a larger extent than any local property (e.g., degree) of a vertex. Their findings also suggest that a network's core vertices (as measured by being in the k-core of a network with high k) are much better spreaders of information than vertices with merely high degree. Recent followup work has also suggested that many core spreaders need not have high degrees [60], further highlighting the substantive difference between core vertices and high-degree vertices.

7.1 A family of synthetic networks

In this section, we detail our numerical results when applying our methods to a family of synthetic networks with a planted core–periphery structure. We again examine the performance of the methods with respect to how many core and peripheral vertices they classify correctly.

We use variants of the random-graph ensemble that was introduced in Ref. [76]. Let $C_1(n, \beta, p, \kappa)$ denote a family of networks with the following properties: n is the number of vertices, β is the fraction of vertices in the core, and the edge probabilities for core–core, core–periphery, and periphery–periphery connections are given by $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$ with $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = p$. Let $C_2(n, \beta, p, \kappa)$ denote a family of networks, from a slight modification of the above model, in which the edge probabilities are now given by $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$ with $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = \kappa p$. In our simulations, we fix $n = 100$, $\beta = 0.5$, and $p = 0.25$, and we examine core–periphery structure using each of
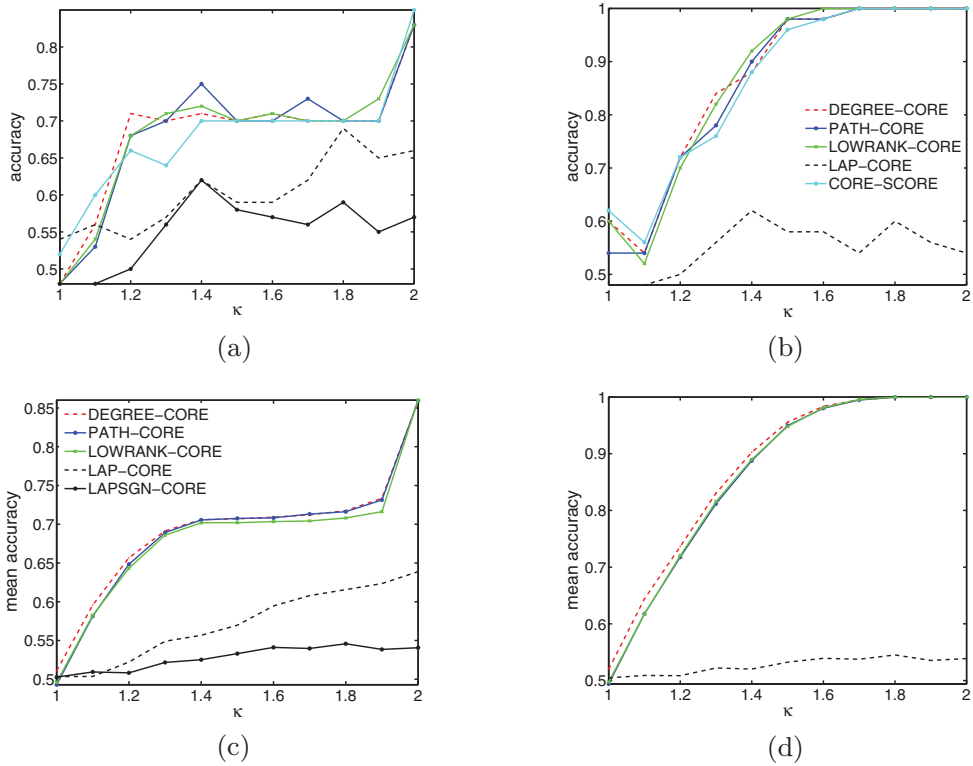
FIGURE 8. Comparison of methods for core–periphery detection using the graph ensemble $C_1(n, \beta, p, \kappa)$ with $n = 100$, $\beta = 0.5$, $p = 0.25$, and edge probabilities $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$, where $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = p$. We vary $\kappa \in [1, 2]$ in increments of 0.1. The top plots illustrate our results for all methods on a single graph from $C_1(n, \beta, p, \kappa)$, and the bottom plots give results averaged over 100 different graphs from the ensemble for all methods except CORE-SCORE. The left plots do not use information about the size (β) of the core, as they rely only on the objective function that one maximizes; the right plots explicitly use knowledge of β. The colours and symbols in the legend in (c) also apply to (a), and the colours and symbols in the legend in (b) also apply to (d). In summary, we show comparisons for the following situations: (a) without knowledge of β, single experiment; (b) with knowledge of β, single experiment; (c) without knowledge of β, averaged over 100 experiments; and (d) with knowledge of β, averaged over 100 experiments.

the proposed methods. We average our results over 100 different instantiations of the above graph ensembles for each of the parameter values $\kappa = 1, 1.1, \ldots, 2$. We also compare our results with (one version of) the CORE-SCORE algorithm introduced in Ref. [76], and we remark that the results of CORE-SCORE are for single networks drawn from the above ensembles. The inefficient running time of the CORE-SCORE algorithm renders it infeasible to average over 100 different instantiations of a graph ensemble.

In Figure 8, we examine the ensemble $C_1(n, \beta, p, \kappa)$. When β is unknown, we find that DEGREE-CORE, PATH-CORE, LOWRANK-CORE, and CORE-SCORE yield similar results to each other. However, when β is known (i.e., when we assume a lower bound on the sizes of the core and periphery sets), we find that DEGREE-CORE and LOWRANK-CORE tend
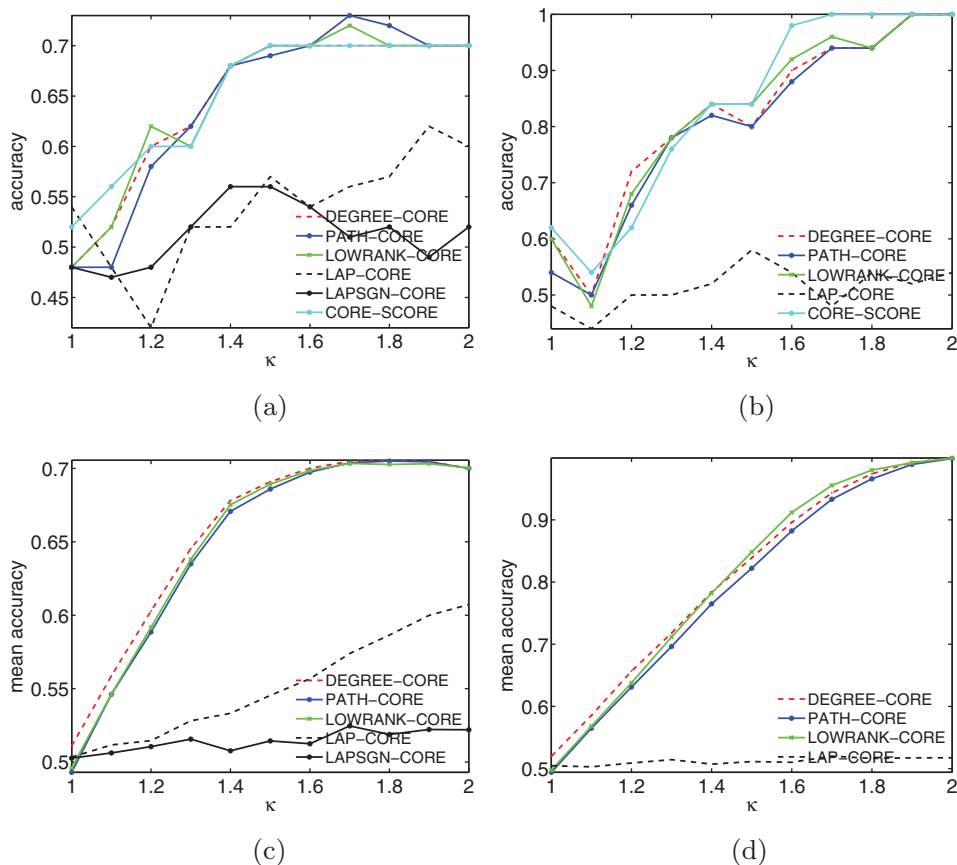
FIGURE 9. Comparison of methods for detecting core–periphery structure for the graph ensemble $C_2(n, \beta, p, \kappa)$ with $n = 100$, $\beta = 0.5$, $p = 0.25$, and edge probabilities $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp})$, where $p_{cc} = \kappa^2 p$, $p_{cp} = \kappa p$, and $p_{pp} = \kappa p$. We vary $\kappa \in [1, 2]$ in increments of 0.1. We show comparisons for the following situations: (a) without knowledge of β, single experiment; (b) with knowledge of β, single experiment; (c) without knowledge of β, averaged over 100 experiments; and (d) with knowledge of β, averaged over 100 experiments.

to perform slightly better than CORE-SCORE and PATH-CORE. As expected, the aggregate performance of the various algorithms improves significantly when we assume knowledge of β. Unfortunately, in both scenarios, the two Laplacian-based methods yield very poor results. Recall that LAPSGN-CORE yields exactly the same results both with and without knowledge of β, so we only show it in the plots without knowledge of β.

In Figure 9, we plot our numerical results for the ensemble $C_2(n, \beta, p, \kappa)$. When β is unknown, DEGREE-CORE, PATH-CORE, LOWRANK-CORE, and CORE-SCORE again yield similar results. When we assume that β is known, we find that CORE-SCORE, LOWRANK-CORE, and DEGREE-CORE still perform similarly to each other, and they all do slightly better than PATH-CORE. The Laplacian-based methods again perform very poorly, though LAP-CORE does slightly better than LAPSGN-CORE when β is unknown.
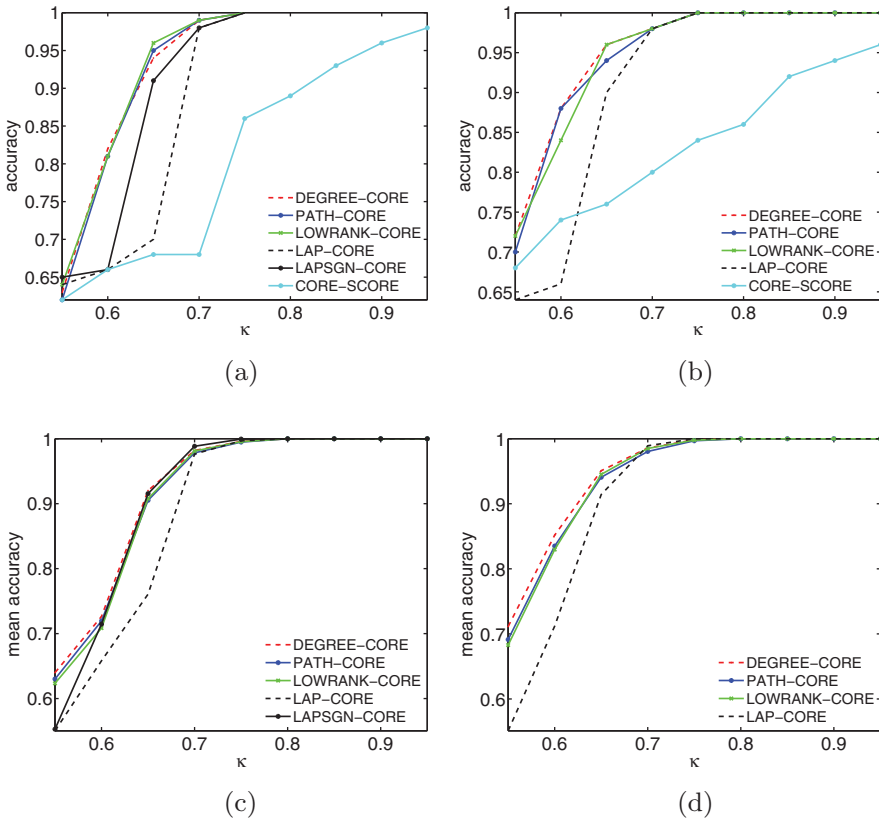
FIGURE 10. Comparison of the methods for graphs with $n = 100$ vertices generated by a core–periphery block model with edge probabilities $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp}) = (\kappa, \kappa, 1 - \kappa)$ for $\kappa \in \{0.55, 0.60, \ldots, 0.95\}$. We show comparisons for the following situations: (a) without knowledge of β, single experiment; (b) with knowledge of β, single experiment; (c) without knowledge of β, averaged over 100 experiments; and (d) with knowledge of β, averaged over 100 experiments.

In Figure 10, we consider graphs with a core–periphery structure from a random-graph ensemble $G(p_{cc}, p_{cp}, p_{pp})$ (see Table 1) with edge probabilities $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp}) = (\kappa, \kappa, 1 - \kappa)$ for different values of κ. The common feature of this set of experiments—both when the boundary size β is known and when it is unknown—is that DEGREE-CORE, LOWRANK-CORE, and PATH-CORE give the best results, whereas CORE-SCORE consistently comes in last place (except for doing somewhat better than the Laplacian-based methods for values of κ in the range $[0.5, 0.6]$ when β is known) in terms of accuracy. In Figure 11, we consider the values of the objective function (4.6), averaged over 100 runs, that we obtain using the different partitions of a network's vertices into core and periphery sets as we sweep along the sorted scores that we compute using each of the methods (except CORE-SCORE, which we omit because of its slow computation time). In Figure 12, we compare the actual values of the objective function for a single experiment across all methods (including CORE-SCORE) as we vary the parameter κ. For each method, we also show the evolution of the value of the objective function as we sweep through the vector of scores.
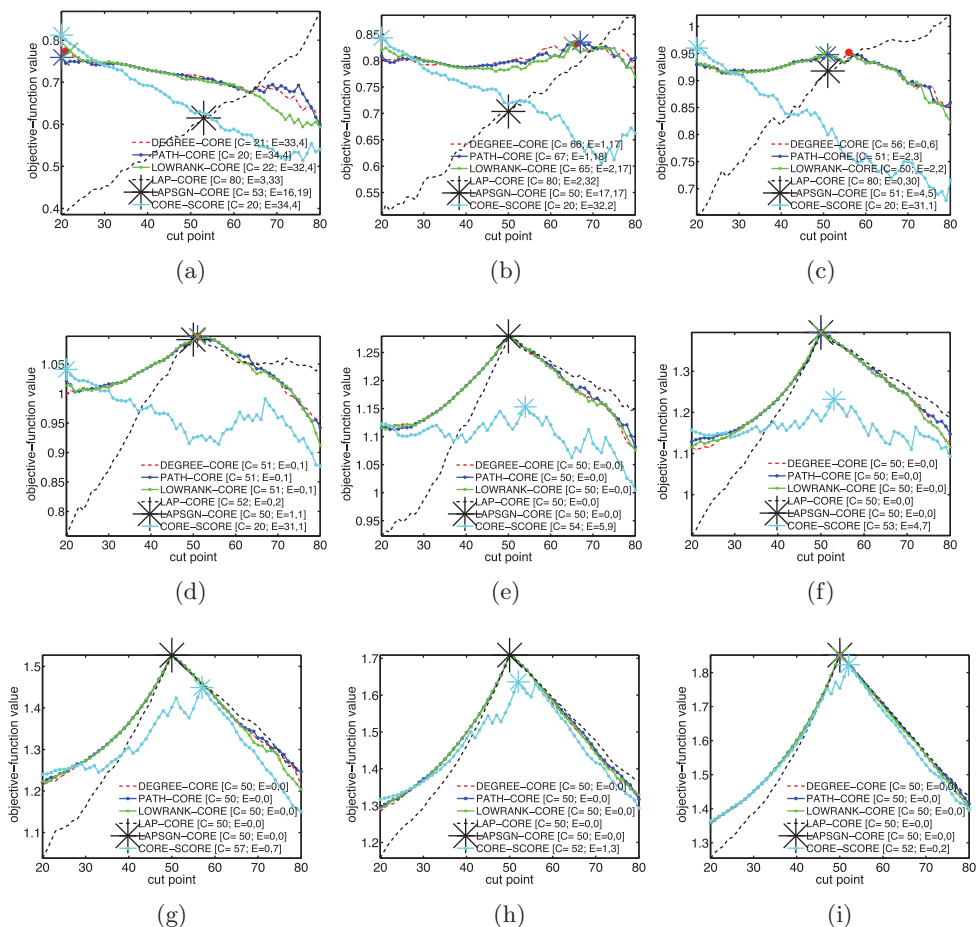
FIGURE 11. Comparison of the values of the objective function (4.6) of the partition of networks into a core set and a periphery set. We calculate these values from the sorted scores from the various methods for detecting core–periphery structure as we vary the parameter κ in the ensemble $G(p_{cc}, p_{cp}, p_{pp})$ from Table 1 with $n = 100$. The probability vector in the block model is $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp}) = (\kappa, \kappa, 1 - \kappa)$. The "cut point" refers to the number of vertices in the core set. In the legends, C denotes the size of the core set, and $\mathbf{E} = (y_1, y_2)$ denotes the corresponding 2-vector of errors. The first component of $\mathbf{E}$ indicates the number of core vertices that we label as peripheral vertices, and the second indicates the number of peripheral vertices that we label as core vertices. We examine the following parameter values: (a) $\kappa = 0.55$, (b) $\kappa = 0.60$, (c) $\kappa = 0.65$, (d) $\kappa = 0.70$, (e) $\kappa = 0.75$, (f) $\kappa = 0.80$, (g) $\kappa = 0.85$, (h) $\kappa = 0.90$, and (i) $\kappa = 0.95$.

In Figure 13, we compare the computation times (in seconds and on a $\log_{10}$ scale) for all of the methods that we examine. The computers that we use for this comparison have 12 CPU cores (Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz) and have 48 GB RAM. The most computationally expensive method is CORE-SCORE, which is 1–2 orders-of-magnitude slower than PATH-CORE, which is in turn 3–4 orders of magnitude slower than the spectral LOWRANK-CORE and LAP-SCORE methods (which have very similar computation times). Finally, as expected, the trivial DEGREE-CORE method has the fastest computation times.
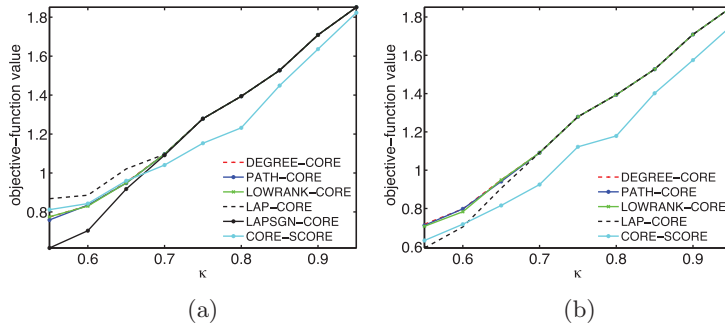
FIGURE 12. Comparison of the values of the objective function (4.6) for a single experiment for all methods as a function of the parameter κ in the ensemble $G(p_{cc}, p_{cp}, p_{pp})$ from Table 1 with $n = 100$ and $\beta = 0.5$. The probability vector in the block model is $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp}) = (\kappa, \kappa, 1 - \kappa)$. We do our comparison both (a) without knowledge of β and (b) with knowledge of β.
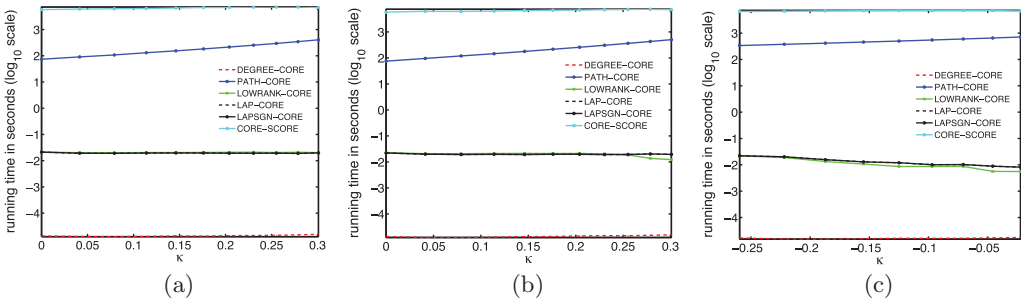

FIGURE 13. Comparison of the computation times (in seconds and on a $\log_{10}$ scale) for all methods and for three synthetic graph ensembles: (a) $C_1(n, \beta, p, \kappa)$, $n = 100, \beta = 0.5, p = 0.25$; (b) $C_2(n, \beta, p, \kappa)$, $n = 100, \beta = 0.5, p = 0.25$; and (c) $G(p_{cc}, p_{cp}, p_{pp})$ with $\mathbf{p} = (p_{cc}, p_{cp}, p_{pp}) = (\kappa, \kappa, 1 - \kappa)$.

7.2 Application to empirical data

In a recent publication [56], a subset of us applied the PATH-CORE and CORE-SCORE algorithms for detecting core–periphery structure in a variety of real-world networks. In the present paper, we use our various methods on a few other empirical data sets.

We consider four examples of social networks using all of the methods that we have discussed for detecting core–periphery structure. The first two graphs are publicly available networks of network scientists from 2006 (NNS2006) [64] and 2010 (NNS2010) [31] with 379 and 552 vertices, respectively, in their largest connected components (LCCs). Ref. [76] considered core–periphery structure in both of these networks. The vertices are scholars (predominantly from physics) who study network science, and the weight of each (undirected) edge represents the strength of a coauthorship relationship. (See the original references for additional discussion of these networks and for more details about the weights, which are not necessarily defined in the same way in the two networks.) The other two networks are two universities (Caltech and Reed College) from the Facebook100 data set [86, 87], which consists of a single-time snapshot from the online social network

Facebook in autumn 2005 for each of 100 universities in the United States. Caltech has 762 vertices in its LCC, and Reed has 962 vertices in its LCC.

In Figures 14 (for the networks of network scientists) and 15 (for the Facebook networks), we present the objective-function values (4.6) for each method for detecting core–periphery structure. In Table D1 in Appendix D, we compare the Pearson and Spearman correlations between the coreness values that we obtain from applying the different methods to these empirical networks. For these networks, we find that the values of Degree-Core, Core-Score, Path-Core, and LowRank-Core are usually strongly correlated to each other, whereas the Lap-Core and LapSgn-Core values are very different from the others (and the Lap-Core values are sometimes almost entirely uncorrelated with them). We find similar results when we use a similarity measure to compare partitions into a core set and periphery set from maximizing the objective function (4.6). We compute a similarity between two measures using the expression

$$S_{\text{frac}} = \frac{w_1}{w_1 + w_0}\,, \tag{7.1}$$

where w_1 is the number of vertices classified in the same way (i.e., either both as core vertices or both as peripheral vertices) in both measures, and w_0 is the number of vertices that are classified differently in the two measures. (Thus, $w_0 + w_1 = n$ is the total number of vertices.) One can also observe that the two networks of network scientists are similar to each other and that the two Facebook networks are similar to each other in terms of their correlations and core–periphery partitions. See Table D1 in Appendix D, and also see Figures 14 and 15. For instance, the core–periphery separation points of Lap-Core and LapSgn-Core yield much closer S_{frac} values for Facebook networks than for the networks of network scientists.

8 Summary and discussion

We introduced several new methods for detecting core–periphery structure in graphs, and we compared these methods to each other and with Core-Score (an existing method) using both synthetic and empirical networks. Our approach based on transportation relies on computing shortest paths in a graph between a pair of adjacent vertices after temporarily removing the edge between the two vertices. Another approach, which is motivated by the putative existence of a low-rank structure in networks that exhibit core–periphery structure, relies on a low-rank approximation of the adjacency matrix of a graph. We also introduced two methods that rely on the bottom eigenvector of the random-walk Laplacian associated with a graph. Finally, we introduced an objective function that helps in the classification of vertices into core and peripheral vertices, and we showed how one can use this objective function after obtaining a vector of scores to measure coreness (using any of the above methods). Core–periphery structure is a common feature of real-world networks, and it is important to continue to develop methods to detect it and to compare the performance of such methods against each other on a wide variety of networks. We have introduced and explored the performance of several new methods in this paper. The different methods that we introduced are based on rather different ideas, and it is very important to explore core–periphery structure from a multitude of perspectives.
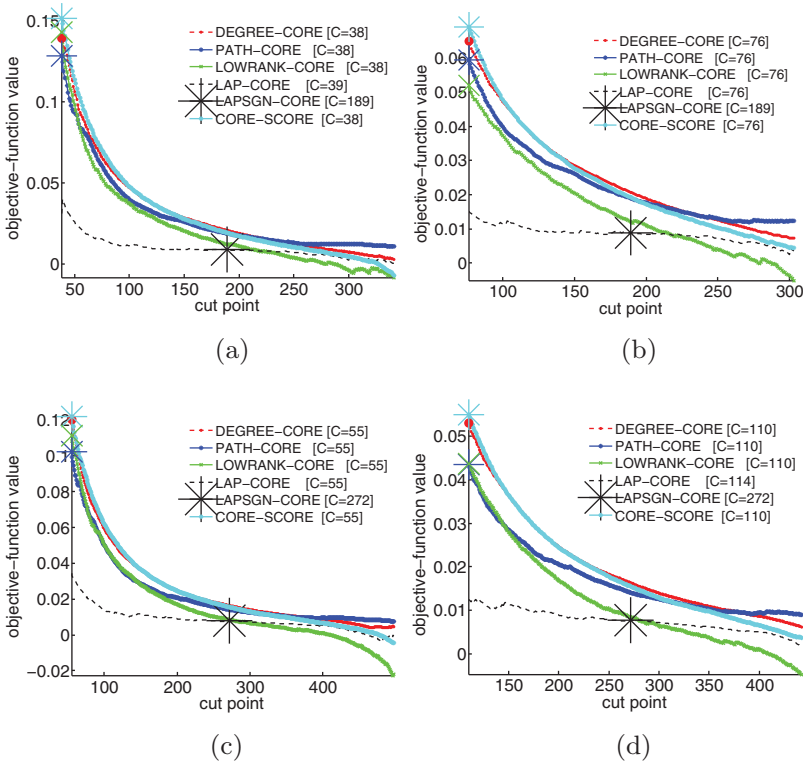
FIGURE 14. Comparison of the methods for detecting core–periphery structure for networks of network scientists in (a,b) a data set from 2006 [64] and (c,d) a data set from 2010 [31] for the objective function in equation (4.6). We assume a minimum size for the core and periphery sets of at least (a,c) 10% of the vertices and (b,d) 20% of the vertices. We mark the cut points on the curves with a large asterisk for LAPSGN-CORE and using symbols whose colours match the colours of the corresponding curves for the other methods. The cut point refers to the number of core vertices, and the C values in the legends give the cut points. In other words, there are C vertices in the core set. For all methods except LAPSGN-CORE, we determine a cut point to separate core and periphery sets by using the FIND-CUT algorithm, which in our computations finds a global optimum at (or very close to) the boundary of the search interval.

Given the common use of k-cores in the consideration of core parts of networks, it is also interesting to examine the assignment of vertices into core and periphery sets based only on vertex degrees. Although using vertex degree as a measure of centrality or likelihood of belonging to a core can often produce inaccurate results [76], it can sometimes be true that a degree-based classification of vertices as core vertices or peripheral vertices should be successful for certain random-graph ensembles (and certain empirical networks) [92]. One can thus ask what properties such ensembles ought to have. More generally, it is also important to compare coreness scores with other centrality measures [6, 56, 76]. Another interesting question is whether one can use current methods for solving the group-synchronization problem (such as the eigenvector method and semidefinite programming [39, 42, 81]) for the detection of core–periphery structure in various families in networks.
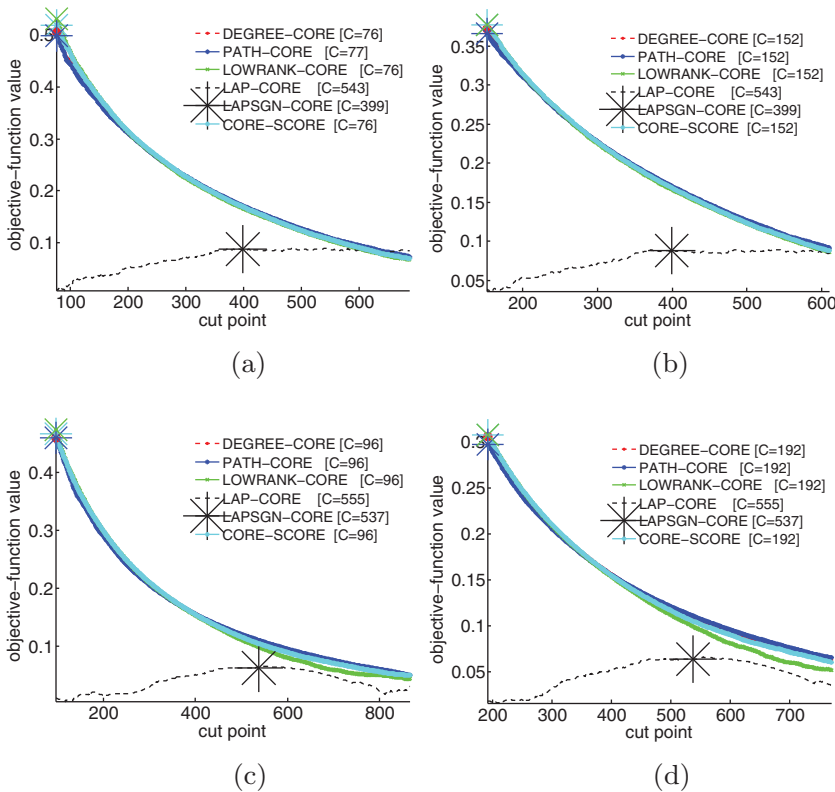
FIGURE 15. Comparison of the methods for detecting core–periphery structure for Facebook networks [86, 87] of (a,b) Caltech and (c,d) Reed College for the objective function in equation (4.6). We assume a minimum size for the core and periphery sets of at least (a,c) 10% of the vertices and (b,d) 20% of the vertices. We mark the cut points on the curves with a large asterisk for LAPSGN-CORE and using symbols whose colours match the colours of the corresponding curves for the other methods. The cut point refers to the number of core vertices, and the C values in the legends give the cut points. In other words, there are C vertices in the core set. For all methods except LAPSGN-CORE, we determine a cut point to separate core and periphery sets by using the FIND-CUT algorithm, which in our computations finds a global optimum at (or very close to) the boundary of the search interval.

An important future application is to examine core–periphery structure in temporal and multilayer networks [45, 52, 61]. Community structure (see, e.g., [47, 61]) has been studied in such contexts, and it should also be very insightful to also consider core–periphery structure in multilayer networks. Another interesting direction is developing additional objective functions for classifying vertices as core or peripheral vertices. It is also important to pursue statistical methods for studying core–periphery structure and other mesoscale network structures [70].

Networks have many different types of mesoscale structures. In most research thus far, community structure has taken centre stage. Other mesoscale structures, such as role assignment [77] and core–periphery structure [26], are also very important. These ideas are worthy of considerably more exploration.

Acknowledgements

References

[1] Ahn, Y.-Y., Bagrow, J. P. & Lehmann, S. (2010) Link communities reveal multiscale complexity in networks. *Nature* **466**, 761–764.

[2] Anthonisse, J. M. (1971) *The Rush in a Directed Graph*, Stichting Mathematisch Centrum, Amsterdam. Available at `http://oai.cwi.nl/oai/asset/9791/9791A.pdf`.

[3] Arenas, A., Díaz-Guilera, A. & Pérez-Vicente, C. J. (2006) Synchronization reveals topological scales in complex networks. *Phys. Rev. Lett.* **96**, 114102.

[4] Ball, B., Karrer, B. & Newman, M. E. J. (2011) Efficient and principled method for detecting communities in networks. *Phys. Rev. E* **84**, 036103.

[5] Barranca, V. J., Zhou, D. & Cai, D. (2015) Low-rank network decomposition reveals structural characteristics of small-world networks. *Phys. Rev. E* **92**, 062822.

[6] Barucca, P., Tantari, D. & Lillo, F. (2016) Centrality metrics and localization in core–periphery networks. *J. Stat. Mech. Theor. Exp.* **2016**, 023401.

[7] Bascompte, J., Jordano, P., Melián, C. J. & Olesen, J. M. (2003) The nested assembly of plant-animal mutualistic networks. *Proc. Natl. Acad. Sci. U.S.A.* **100**, 9383–9387.

[8] Bassett, D. S., Wymbs, N. F., Porter, M. A., Mucha, P. J., Carlson, J. M. & Grafton, S. T. (2011) Dynamic reconfiguration of human brain networks during learning. *Proc. Natl. Acad. Sci. U.S.A.* **108**, 7641–7646.

[9] Bassett, D. S., Wymbs, N. F., Rombach, M. P., Porter, M. A., Mucha, P. J. & Grafton, S. T. (2013) Task-based core–periphery organization of human brain dynamics. *PLoS Comput. Biol.* **9**, e1003171.

[10] Belkin, M. & Niyogi, P. (2003) Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* **15**, 1373–1396.

[11] Benaych-Georges, F. & Nadakuditi, R. R. (2011) The eigenvalues and eigenvectors of finite, low rank perturbations of large random matrices. *Adv. Math.* **227**, 494–521.

[12] Bhatia, R. (1997) *Matrix Analysis*, Graduate Texts in Mathematics, vol. 169, Springer-Verlag, Berlin, Germany.

[13] Borgatti, S. P. & Everett, M. G. (1999) Models of core/periphery structures. *Soc. Netw.* **21**, 375–395.

[14] Borgatti, S. P., Everett, M. G. & Freeman, L. C. (2011) *UCINET*, version 6.289. Available at `http://www.analytictech.com/ucinet/`.

[15] BOYD, J. P., FITZGERALD, W. J., MAHUTGA, M. C. & SMITH, D. A. (2010) Computing continuous core/periphery structures for social relations data with MINRES/SVD. *Soc. Netw.* **32**, 125–137.

[16] BRANDES, U. (2001) A faster algorithm for betweenness centrality. *J. Math. Sociol.* **25**, 163–177.

[17] BRANDES, U., DELLING, D., GAERTLER, M., GÖRKE, R., HOEFER, M., NIKOLOSKI, Z. & WAGNER, D. (2008) On modularity clustering. *IEEE Trans. Knowl. Data Eng.* **20**, 172–188.

[18] CHASE-DUNN, C. (1989) *Global Formation: Structures of the World-Economy*, Basil Blackwell, Oxford, UK.

[19] CHEN, J. & YUAN, B. (2006) Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics* **22**, 2283–2290.

[20] CHUNG, F. R. K. (1997) *Spectral Graph Theory*, CBMS Regional Conference Series, American Mathematical Society, Providence, RI.

[21] CLAUSET, A., ARBESMAN, S. & LARREMORE, D. B. (2015) Systematic inequality and hierarchy in faculty hiring networks. *Sci. Adv.* **1**, e1400005.

[22] COIFMAN, R. R. & LAFON, S. (2006) Diffusion maps. *Appl. Comput. Harmon. Anal.* **21**, 5–30.

[23] COIFMAN, R. R., LAFON, S., LEE, A. B., MAGGIONI, M., NADLER, B., WARNER, F. & ZUCKER, S. W. (2005) Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proc. Natl. Acad. Sci. U.S.A.* **102**, 7426–7431.

[24] COLIZZA, V., FLAMMINI, A., SERRANO, M. A. & VESPIGNANI, A. (2006) Detecting rich-club ordering in complex networks. *Nat. Phys.* **2**, 110–115.

[25] COMREY, A. L. (1962) The minimum residual method of factor analysis. *Psychol. Rep.* **11**, 15–18.

[26] CSERMELY, P., LONDON, A., WU, L.-Y. & UZZI, B. (2013) Structure and dynamics of core/periphery networks. *J. Cplx. Netw.* **1**, 93–123.

[27] DA SILVA, M. R., MA, H. & ZENG, A.-P. (2008) Centrality, network capacity, and modularity as parameters to analyze the core–periphery structure in metabolic networks. *Proc. IEEE* **96**, 1411–1420.

[28] DARLINGTON, R. B., WEINBERG, S. L. & WALBERG, H. J. (1973) Canonical variate analysis and related techniques. *Rev. Educ. Res.* **43**, 433–454.

[29] DELLA-ROSSA, F. D., DERCOLE, F. & PICCARDI, C. (2013) Profiling core–periphery network structure by random walkers. *Sci. Rep.* **3**, 1467.

[30] DOREIAN, P. (1985) Structural equivalence in a psychology journal network. *J. Assoc. Inf. Sci.* **36**, 411–417.

[31] EDLER, D. & ROSVALL, M. (2010) *The map generator software package (2010 network scientist coauthorship network)*. Accessed 12 September 2014. Available at http://mapequation.org/downloads/netscicoauthor2010.net.

[32] ERDŐS, P. & RÉNYI, A. (1959) On random graphs I. *Publ. Math. Debrecen* **6**, 290–297.

[33] EVERETT, M. G. & VALENTE, T. W. (2016) Bridging, brokerage and betweenness. *Soc. Netw.* **44**, 202–208.

[34] FÉRAL, D. & PÉCHÉ, S. (2007) The largest eigenvalue of rank one deformation of large Wigner matrices. *Comm. Math. Phys.* **272**, 185–228.

[35] FORTUNATO, S. (2010) Community detection in graphs. *Phys. Rep.* **486**, 75–174.

[36] FREEMAN, L. C. (1977) A set of measures of centrality based on betweenness. *Sociometry* **40**, 35–41.

[37] GILBERT, E. N. (1959) Random graphs. *Ann. Math. Stat.* **30**, 1141–1144.

[38] GIRVAN, M. & NEWMAN, M. E. J. (2002) Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **99**, 7821–7826.

[39] GOEMANS, M. X. & WILLIAMSON, D. P. (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **42**, 1115–1145.

[40] GONZÁLEZ, M. C., HERRMANN, H. J., KERTÉSZ, J. & VICSEK, T. (2007) Community structure and ethnic preferences in school friendship networks. *Physica A* **379**, 307–316.

[41] GOOD, B. H., DE MONTJOYE, Y.-A. & CLAUSET, A. (2010) Performance of modularity maximization in practical contexts. *Phys. Rev. E* **81**, 046106.

[42] GRANT, M. & BOYD, S. (2008) Graph implementations for nonsmooth convex programs, In: V. Blondel, S. Boyd & H. Kimura (editors), *Recent Advances in Learning and Control*, Lecture Notes Contr. Inf. Sci., Springer-Verlag, Berlin, Germany, pp. 95–110.

[43] GUATTERY, S. & MILLER, G. L. (1995) On the performance of spectral graph partitioning methods. In: *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms, SODA '95, January 22–24, 1995*, San Francisco, CA, USA. SIAM, Philadelphia, PA, USA, pp. 233–242.

[44] HOLME, P. (2005) Core–periphery organization of complex networks. *Phys. Rev. E* **72**, 046111.

[45] HOLME, P. & SARAMÄKI, J. (2012) Temporal networks. *Phys. Rep.* **519**, 97–125.

[46] JEUB, L. G. S., BALACHANDRAN, P., PORTER, M. A., MUCHA, P. J. & MAHONEY, M. W. (2015) Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Phys. Rev. E* **91**, 012821.

[47] JEUB, L. G. S., MAHONEY, M. W., MUCHA, P. J. & PORTER, M. A. (2015) A local perspective on community structure in multilayer networks, arXiv:1510.05185.

[48] KELMANS, A. K. (1965) The number of trees of a graph I. *Aut. Remote Contr.* **26**, 2118–2129.

[49] KELMANS, A. K. (1966) The number of trees of a graph II. *Aut. Remote Contr.* **27**, 233–241.

[50] KELMANS, A. K. (1997) Transformations of a graph increasing its Laplacian polynomial and number of spanning trees. *Europ. J. Comb.* **18**, 35–48.

[51] KITSAK, M., GALLOS, L. K., HAVLIN, S., LILJEROS, F., MUCHNIK, L., STANLEY, H. E. & MAKSE, H. A. (2010) Identification of influential spreaders in complex networks. *Nat. Phys.* **6**, 888–893.

[52] KIVELÄ, M., ARENAS, A., BARTHELEMY, M., GLEESON, J. P., MORENO, Y. & PORTER, M. A. (2014) Multilayer networks. *J. Cplx. Netw.* **2**, 203–271.

[53] KRUGMAN, P. (1996) *The Self-Organizing Economy*, Oxford University Press, Oxford, UK.

[54] LAUMANN, E. O. & PAPPI, F. U. (1976) *Networks of Collective Action: A Perspective on Community Influence*, Academic Press, New York, NY, USA.

[55] LEE, S. H. (2016) Network nestedness as generalized core–periphery structures. *Phys. Rev. E* **93**, 022306.

[56] LEE, S. H., CUCURINGU, M. & PORTER, M. A. (2014) Density-based and transport-based core–periphery structures in networks. *Phys. Rev. E* **89**, 032810.

[57] LEWIS, A. C. F., JONES, N. S., PORTER, M. A. & DEANE, C. M. (2010) The function of communities in protein interaction networks at multiple scales. *BMC Syst. Biol.* **4**, 100.

[58] MCLACHLAN, G. & PEEL, D. (2000) *Finite Mixture Models*, Wiley-Interscience, Hoboken, NJ, USA.

[59] MEILĂ, M. & SHI, J. (2001) A random walks view of spectral segmentation. In: *8th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, Key West, FL, USA.

[60] MORONE, F. & MAKSE, H. A. (2015) Influence maximization in complex networks through optimal percolation. *Nature* **524**, 65–68.

[61] MUCHA, P. J., RICHARDSON, T., MACON, K., PORTER, M. A. & ONNELA, J.-P. (2010) Community structure in time-dependent, multiscale, and multiplex networks. *Science* **328**, 876–878.

[62] NESTEROV, Y. (2004) *Introductory Lectures on Convex Optimization: A Basic Course*, Applied optimization, Kluwer Academic Publ., Dordrecht, the Netherlands.

[63] NEWMAN, M. E. J. (2005) A measure of betweenness centrality based on random walks. *Soc. Netw.* **27**, 39–54.

[64] NEWMAN, M. E. J. (2006) Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* **74**, 036104.

[65] NEWMAN, M. E. J. (2010) *Networks: An Introduction*, Oxford University Press, UK.

[66] NEWMAN, M. E. J. & GIRVAN, M. (2003) Mixing patterns and community structure in networks. In: R. Pastor-Satorras, M. Rubi & A. Díaz-Guilera (editors), *Statistical Mechanics of Complex Networks* Lecture Notes in Physics, vol. 625, Springer-Verlag, Berlin, Germany, pp. 66–87.

[67] NEWMAN, M. E. J. & GIRVAN, M. (2004) Finding and evaluating community structure in networks. *Phys. Rev. E* **69**, 026113.

[68] ONNELA, J.-P., SARAMÄKI, J., HYVÖNEN, J., SZABÓ, G., LAZER, D., KASKI, K., KERTÉSZ, J. & BARABÁSI, A. L. (2007) Structure and tie strengths in mobile communication networks. *Proc. Natl. Acad. Sci. U.S.A.* **104**, 7332–7336.

[69] PALLA, G., DERENYI, I., FARKAS, I. & VICSEK, T. (2005) Uncovering the overlapping community structure of complex networks in nature and society. *Nature* **435**, 814–818.

[70] PEIXOTO, T. P. (2014) Hierarchical block structures and high-resolution model selection in large networks. *Phys. Rev. X* **4**, 011047.

[71] PICCARDI, C. (2011) Finding and testing network communities by lumped Markov chains. *PLoS ONE* **6**, e27028.

[72] PONS, P. & LATAPY, M. (2006) Computing communities in large networks using random walks. *J. Graph Algorithms Appl.* **10**, 191–218.

[73] PORTER, M. A., MUCHA, P. J., NEWMAN, M. E. J. & WARMBRAND, C. M. (2005) A network analysis of committees in the U.S. House of Representatives. *Proc. Natl. Acad. Sci. U.S.A.* **102**, 7057–7062.

[74] PORTER, M. A., ONNELA, J.-P. & MUCHA, P. J. (2009) Communities in networks. *Notices Amer. Math. Soc.* **56**, 1082–1097, 1164–1166.

[75] RICHARDSON, T., MUCHA, P. J. & PORTER, M. A. (2009) Spectral tripartitioning of networks. *Phys. Rev. E* **80**, 036111.

[76] ROMBACH, M. P., PORTER, M. A., FOWLER, J. H. & MUCHA, P. J. (2014) Core–periphery structure in networks. *SIAM J. Appl. Math.* **74**, 167–190.

[77] ROSSI, R. A. & AHMED, N. K. (2015) Role discovery in networks. *IEEE Trans. Knowl. Data Eng.* **27**, 1112–1131.

[78] ROSVALL, M. & BERGSTROM, C. T. (2008) Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. U.S.A.* **105**, 1118–1123.

[79] ROWEIS, S. T. & SAUL, L. K. (2000) Nonlinear dimensionality reduction by locally linear embedding. *Science* **290**, 2323–2326.

[80] SHI, J. & MALIK, J. (2000) Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **22**, 888–905.

[81] SINGER, A. (2011) Angular synchronization by eigenvectors and semidefinite programming. *Appl. Comput. Harmon. Anal.* **30**, 20–36.

[82] SMITH, D. A. & WHITE, D. R. (1992) Structure and dynamics of the global economy: Network analysis of international trade. *Soc. Forces* **70**, 857–893.

[83] SPIELMAN, D. A. & TENG, S.-H. (1996) Spectral partitioning works: Planar graphs and finite element meshes. In: *Foundations Of Computer Science* (*FOCS*), IEEE Computer Society, Washington, D.C., USA, pp. 96–105.

[84] SPIELMAN, D. A. & TENG, S.-H. (2007) Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra Appl.* **421**, 284–305. Special Issue in honor of Miroslav Fiedler.

[85] STEIBER, S. (1979) The world system and world trade: An empirical explanation of conceptual conflicts. *Sociol. Quart.* **20**, 23–26.

[86] TRAUD, A. L., KELSIC, E. D., MUCHA, P. J. & PORTER, M. A. (2011) Comparing community structure to characteristics in online collegiate social networks. *SIAM Rev.* **53**, 526–543.

[87] TRAUD, A. L., MUCHA, P. J. & PORTER, M. A. (2012) Social structure of Facebook networks. *Physica A* **391**, 4165–4180.

[88] VALENTE, T. W. & FUJIMOTO, K. (2010) Bridging: Locating critical connectors in a network. *Soc. Netw.* **32**, 212–220.

[89] VERMA, T., RUSSMANN, F., ARAÚJO, N. A. M., NAGLER, J. & HERMANN, H. J. (2016) Emergence of core–peripheries in networks. *Nat. Commun.* **7**, 10441.

[90] WALLERSTEIN, I. (1974) *The Modern World-System I: Capitalist Agriculture and the Origins of the European World-Economy in the Sixteenth Century*, Academic Press, New York, NY, USA.

[91] YANG, J. & LESKOVEC, J. (2014) Structure and overlaps of ground-truth communities in networks. *ACM Trans. Intell. Syst. Technol.* **5**, 26.

[92] ZHANG, X., MARTIN, T. & NEWMAN, M. E. J. (2015) Identification of core–periphery structure in networks. *Phys. Rev. E* **91**, 032803.

Appendix A Algorithm for computing path-core scores

Let $G(V, E)$ be an unweighted graph without self-edges or multi-edges. (In other words, it is a simple graph.) Recall that we define the PATH-CORE score (3.2) of a vertex $i \in V = V(G)$ as the sum over all adjacent vertex pairs in G of the fraction of shortest non-trivial paths containing i between each vertex pair in $V(G) \setminus i$. By "non-trivial," we mean that the direct edge between those adjacent vertices does not count as a path. Our algorithm has strong similarities to the algorithm presented in Ref. [16], and we follow some of the notation introduced therein. Let $d_G(j, i)$ be the "distance" between vertices j and i; we define this distance as the minimum length of any path that connects j and i in G. Let $\sigma_{st}(i)$ be the number of shortest paths between s and t that contain i. Define the set of *predecessors* of a vertex i on shortest paths from s as

$$P_s(i) = \{j \in V : (j, i) \in E, d_G(s, i) = d_G(s, j) + 1\}.$$

We use the following observation: if i lies on a shortest path between s and t, then

$$\sigma_{st}(i) = \left(\sum_{k \in P_s(i)} \sigma_{si}(k) \right) \times \left(\sum_{l \in P_t(i)} \sigma_{it}(l) \right).$$

This will help us count the number of shortest paths on which a vertex lies without keeping track of the locations of these shortest paths. In the PATH-SCORE algorithm, $\sigma_s(i)$ is the number of paths between s and i of length $d_{G'}(s, i)$ if and only if i lies on a shortest path between s and t (i.e., if (s, t) is the edge that is currently removed), where G' is the graph $G \setminus (s, t)$. The algorithm records the distance between s and i in G' as $d_s(i)$. In Algorithm 5, we calculate PATH-CORE scores for every vertex.

Lemma 1 *Algorithm 5 outputs the* PATH-CORE *scores for all vertices in an unweighted graph G.*

Proof It suffices to show for one edge $(s, t) \in E(G)$ and one iteration (i.e., lines 3–44) that the algorithm counts, for each vertex $w \in V(G) \setminus (s, t)$, the number of shortest paths between s and t that contain w. This number $\sigma_{s,t}(w)$ is given by the algorithm as $\sigma_s(w) \cdot \sigma_t(w)$. In this case, $\sigma_s(w)$ is the number of paths between s and w of length $d_{G'}(s, w)$ (where the graph $G' = G \setminus (s, t)$) if and only if w lies on a shortest path between s and t.

Algorithm 5 performs three breadth-first-searches (BFSs). In the first BFS, it starts

Algorithm 5 PATH-CORE: Computes PATH-CORE scores for all vertices of a graph G.

Input: G	22: **for** each $u \in \Gamma_{G'}(w)$ **do**
Output: C_P	23: **if** $d_t(u) < 0$ **then**
	24: enqueue $u \to Q$;
1: $C_P(w) \leftarrow 0$, $w \in V$;	25: $d_t(u) \leftarrow d_t(w) + 1$;
	26: **end if**
2: **for** $(s,t) \in E(G)$ **do**	27: **if** $d_s(u) < d_s(w)$ **then**
	28: $\sigma_t(u) = \sigma_t(u) + \sigma_t(w)$;
3: $G' \leftarrow G \setminus (s,t)$;	29: **end if**
4: $\sigma_s(w), \sigma_t(w) \leftarrow 0, v \in V$;	30: **end for**
5: $\sigma_s(s), \sigma_t(t) \leftarrow 1$;	31: **end while**
6: $d_s(w), d_t(w) \leftarrow -1, v \in V$;	
7: $d_s(s), d_t(t) \leftarrow 0$;	32: enqueue $s \to Q$;
8: $Q \leftarrow$ empty queue;	33: **while** Q not empty **do**
	34: dequeue $w \leftarrow Q$;
9: enqueue $s \to Q$;	35: **for** each $u \in \Gamma_{G'}(w)$ **do**
10: **while** Q not empty **do**	36: **if** $d_t(u) < d_t(w)$ **then**
11: dequeue $w \leftarrow Q$;	37: enqueue $u \to Q$;
12: **for** each $u \in \Gamma_{G'}(w)$ **do**	38: $\sigma_s(u) = \sigma_s(u) + \sigma_s(w)$;
13: **if** $d_s(u) < 0$ **then**	39: **end if**
14: enqueue $u \to Q$;	40: **end for**
15: $d_s(u) \leftarrow d_s(w) + 1$;	41: **end while**
16: **end if**	
17: **end for**	42: **for** $w \in V \setminus (s,t)$ **do**
18: **end while**	43: $C_P(w) = C_P(w) + \sigma_s(w) \cdot \sigma_t(w)/\sigma_s(t)$;
	44: **end for**
19: enqueue $t \to Q$;	
20: **while** Q not empty **do**	45: **end for**
21: dequeue $w \leftarrow Q$;	

from vertex s and records the distances from s to all other vertices. It then performs a BFS starting from vertex t. During this second BFS, it records the distances to all vertices from t, and it also records $\sigma_t(w)$ for vertices that lie on a shortest path between s and t. The PATH-SCORE algorithm knows that u lies on a shortest path between s and t if it has a distance from s that is less than the distance from s of its predecessor in the BFS that started from t. In other words, if $d_t(w) < d_t(u)$, then an edge (w, u) lies on a shortest path between s and t if and only if $d_s(u) < d_s(w)$. Additionally,

$$\sigma_t(u) = \sum_{w \in P_t(u)} \sigma_t(w).$$

In the second BFS, Algorithm 5 finds a vertex u exactly once for each of its predecessors $w \in P_t(u)$, and it adds $\sigma_t(w)$ to $\sigma_t(u)$. Therefore, in the second BFS, for each vertex $u \in V(G) \setminus (s,t)$, PATH-SCORE records $\sigma_t(v)$ as the number of shortest paths from t to u if u is on a shortest path between s and t. If it is not, then $\sigma_t(u)$ is still 0.

By the same arguments, in the third BFS, for each vertex $u \in V(G) \setminus (s,t)$, PATH-CORE records $\sigma_s(u)$ as the number of shortest paths from s to u if u is on a shortest path between s and t. If it is not, then $\sigma_s(u)$ is still 0.

It should now be clear that for all $w \in V(G) \setminus (s,t)$, it follows that $\sigma_s(w) \cdot \sigma_t(w)$ equals $\sigma_{s,t}(w)$. $\square$

Lemma 2 *Algorithm 5 finishes in $\mathcal{O}(m^2)$ time.*

Proof Algorithm 5 iterates (i.e., it runs lines 3–44) once for each edge. In one iteration, it performs three BFSs. During a BFS, every edge of G' is considered exactly once; this is an $\mathcal{O}(1)$-time procedure. Therefore, every iteration of PATH-CORE runs in $\mathcal{O}(m)$ time, and the temporal complexity of PATH-CORE is $\mathcal{O}(m^2)$. $\square$

For weighted graphs, one can implement an algorithm that is very similar to Algorithm 5. This algorithm uses Dijkstra's algorithm for shortest paths instead of BFS, and it runs in $\mathcal{O}(m + n \log n)$ time instead of $\mathcal{O}(m)$, so the total temporal complexity becomes $\mathcal{O}(m^2 + mn \log n)$.

Appendix B Symmetry in the random-walk Laplacian

We now show that a symmetry relation like (6.2) exists for the random-walk Laplacian associated to an unweighted graph only under certain conditions. Additionally, the most obvious version of such a statement does not hold. To see this, let $\mathbf{x}$ be an eigenvector of $\bar{L}$. (This eigenvector is non-trivial, so $\mathbf{x} \perp \mathbf{1}_n$.) We use the notation $\bar{D} = \text{diag}(n - 1 - d_i)$, where d_i denotes the degree of vertex i, and calculate

$$\begin{aligned}
\bar{L} &= \bar{D}^{-1}\bar{A} \\
&= \bar{D}^{-1}(J_n - A - I_n) \\
&= \bar{D}^{-1}(J_n - I_n) - \bar{D}^{-1}A \\
&= \bar{D}^{-1}(J_n - I_n) - \bar{D}^{-1}DD^{-1}A \\
&= \bar{D}^{-1}(J_n - I_n) - \bar{D}^{-1}DL.
\end{aligned} \tag{B 1}$$

Because $\bar{D}^{-1} = \text{diag}\left(\frac{1}{n-1-d_i}\right)$ and $D = \text{diag}(d_i)$ are diagonal matrices, it follows that $\bar{D}^{-1}D = \text{diag}\left(\frac{d_i}{n-1-d_i}\right)$. Given an eigenvector $\mathbf{x}$ of $\bar{L}$, we obtain $\bar{L}\mathbf{x} = \bar{\lambda}\mathbf{x}$ for some eigenvalue $\bar{\lambda}$. Because $\bar{L}$ is a row-stochastic matrix, it has the trivial eigenvalue $\bar{\lambda}_1 = 1$ with associated eigenvector $\bar{\mathbf{v}}_1 = \mathbf{1}_n$. We apply both sides of equation (B 1) to the eigenvector $\mathbf{x}$ and note that $J_n\mathbf{x} = \mathbf{0}$ because $\mathbf{x} \perp \bar{\mathbf{v}}_1 = \mathbf{1}_n$. We thereby obtain

$$\begin{aligned}
\bar{\lambda}\mathbf{x} &= \bar{L}\mathbf{x} \\
&= \bar{D}^{-1}(J_n - I_n)\mathbf{x} - \bar{D}^{-1}DL\mathbf{x} \\
&= -\bar{D}^{-1}I_n\mathbf{x} - \bar{D}^{-1}DL\mathbf{x}.
\end{aligned} \tag{B 2}$$

Multiplying both sides of equation (B 2) by $D^{-1}\bar{D}$ on the left yields

$$D^{-1}\bar{D}\bar{\lambda}\mathbf{x} = -D^{-1}\bar{D}\bar{D}^{-1}\mathbf{x} - L\mathbf{x}, \tag{B 3}$$

so

$$
\begin{aligned}
L\mathbf{x} &= -D^{-1}\bar{D}\bar{\lambda}\mathbf{x} - D^{-1}\mathbf{x} \\
&= -(\bar{D}\bar{\lambda} + I_n)D^{-1}\mathbf{x} \\
&= -\mathrm{diag}\left(\frac{\bar{\lambda}(n-1-d_i)+1}{d_i}\right)\mathbf{x}.
\end{aligned}
$$

Therefore, $\mathbf{x}$ is not an eigenvector of L unless

$$\frac{\bar{\lambda}(n-1-d_i)+1}{d_i} = \theta \quad \text{for all} \quad i \in \{1,\ldots,n\},$$

for some constant θ. In other words, $d_i = \frac{\bar{\lambda}(n-1)}{\theta+\bar{\lambda}}$ for $i \in \{1,\ldots,n\}$, so the graph is d-regular, with $d_1 = d_2 = \cdots = d_n = d = \frac{\bar{\lambda}(n-1)}{\theta+\bar{\lambda}}$. Therefore,

$$\theta = \frac{\bar{\lambda}(n-1-d)+1}{d},$$

so the eigenvector $\mathbf{x}$ of $\bar{L}$ is also an eigenvector of L (with a corresponding eigenvalue of $-\theta$).

Appendix C Planted high-degree vertices

To illustrate the sensitivity of the DEGREE-CORE method to the presence of high-degree peripheral vertices, we conduct a numerical experiment in which we intentionally plant high-degree vertices in the periphery set. This helps illustrate that it is dangerous to use methods like k-core decomposition (which has very strong demands that vertices have a high degree to be construed as core vertices) to study core–periphery structure. In Figure C1, we consider a graph from the ensemble $G(p_{cc}, p_{cp}, p_{pp}, n_c, n_p)$ with $n = 100$ vertices, edge probabilities $(p_{cc}, p_{cp}, p_{pp}) = (0.4, 0.4, 0.2)$, n_c core vertices, n_p peripheral vertices (with $n = n_c + n_p$), and planted high-degree vertices in the periphery set. To perturb the graph G from the above ensemble to plant high-degree peripheral vertices, we proceed as follows. First, we select each peripheral vertex with independent probability 0.1. Second, we connect each such vertex to 15 non-neighbouring peripheral vertices that we choose uniformly at random. In the left panel of Figure C1, we show an example with a boundary size of 10%, so we are assuming that the core and periphery sets each have at least $0.1n = 10$ vertices. We then search for a cut point in the interval $[10, 90]$. In the right panel, we consider a larger boundary size and assume that the core and the periphery sets each have at least 25 vertices. We now search for an optimal cut in the interval $[25, 75]$. In the two planted-degree scenarios for which the size of the core set is unknown, all
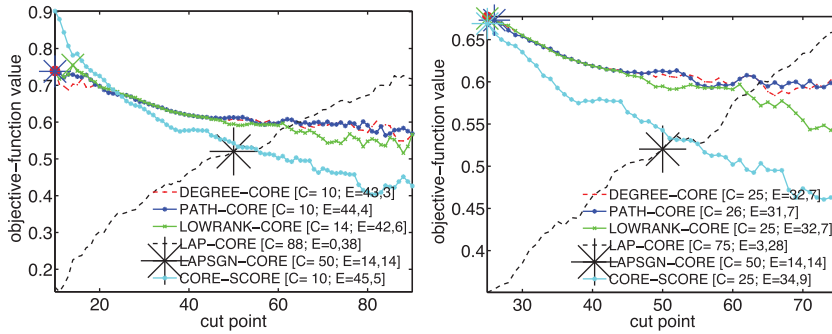
FIGURE C1. Comparison of the methods for one instantiation of the graph ensemble $G(p_{cc}, p_{cp}, p_{pp}, n, n_c, n_p)$ with $n = 100$ vertices, $n_c = 50$ core vertices, $n_p = 50$ peripheral vertices, edge probabilities $(p_{cc}, p_{cp}, p_{pp}) = (0.4, 0.4, 0.2)$, and planted high-degree vertices for the objective function in equation (4.6). The cut point refers to the number of core vertices. In the legends, C denotes the size of the core set, and $\mathbf{E} = (y_1, y_2)$ denotes the corresponding 2-vector of errors. The first component of $\mathbf{E}$ indicates the number of core vertices that we label as peripheral vertices, and the second indicates the number of peripheral vertices that we label as core vertices. In this graph, each peripheral vertex has a probability of 0.1 of becoming adjacent to 15 additional non-neighbouring peripheral vertices that we select uniformly at random. We mark the cut points on the curves with a large asterisk for LapSgn-Core and using symbols whose colours match the colours of the corresponding curves for the other methods. For all methods except LapSgn-Core, we determine a cut point to separate core and periphery sets by using the Find-Cut algorithm, which in our computations finds a global optimum at (or very close to) the boundary of the search interval.

methods yield many misclassified vertices, although the LapSgn-Core method has the lowest number (28) of misclassifications in both cases.

Appendix D Correlations between coreness values from different methods

In Table D1, we consider several empirical networks and give the numerical values for the Pearson and Spearman correlations between the coreness values that we obtain for several core–periphery-detection methods.

Table D1. *Pearson and Spearman correlation coefficients for various coreness measures and the similarity measure S_{frac} for core–periphery partitioning with a boundary of 20% of the vertices (see the right panels in Figures 14 and 15) between the objective function in equation (4.6) for several empirical networks. We use the notation q for* DEGREE-CORE, *$\mathscr{C}$ for* CORE-SCORE, *$\mathscr{P}$ for* PATH-CORE, *$\mathscr{R}$ for* LOWRANK-CORE, *$\mathscr{L}$ for* LAP-CORE, *and $\mathscr{LS}$ for* LAPSGN-CORE. *We use the designation* * *for correlation values that have a p-value smaller than 0.01 and the designation* † *for z-scores whose absolute value is larger than 2. We construe these results as statistically significant. We calculate the z-scores by randomly permuting the vertex indices (with 10000 different applications of such a permutation for each calculation) as described in [86]: $z = (S_{frac} - \mu)/(std)$, where μ and "std", respectively, are the means and standard deviations of the S_{frac} values for random permutations. We use* NNS2006 *to denote the 2006 network of network scientists,* NNS2010 *to denote the 2010 network of network scientists,* FB-Caltech *to denote the Facebook network for Caltech, and* FB-Reed *to denote the Facebook network for Reed.*

NNS2006	$q,\mathscr{C}$	$q,\mathscr{P}$	$q,\mathscr{R}$	$q,\mathscr{L}$	$\mathscr{C},\mathscr{P}$	$\mathscr{C},\mathscr{R}$	$\mathscr{C},\mathscr{L}$	$\mathscr{P},\mathscr{R}$	$\mathscr{P},\mathscr{L}$	$\mathscr{R},\mathscr{L}$
Pearson	0.79*	0.89*	0.72*	0.02	0.64*	0.56*	0.03	0.62*	0.03	−0.01
Spearman	0.79*	0.62*	0.43*	0.04	0.37*	0.65*	−0.05	0.14*	−0.01	0.01
S_{frac}	0.93	0.90	0.79	0.69	0.88	0.82	0.69	0.78	0.69	0.67
(z-score)	(19.5†)	(17.4†)	(8.2†)	(1.1)	(15.7†)	(10.2†)	(1.1)	(7.5†)	(0.7)	(−0.9)

	$q,\mathscr{LS}$	$\mathscr{C},\mathscr{LS}$	$\mathscr{P},\mathscr{LS}$	$\mathscr{R},\mathscr{LS}$	$\mathscr{L},\mathscr{LS}$
S_{frac}	0.50	0.47	0.47	0.44	0.70
(z-score)	(0.1)	(−1.5)	(−1.9)	(−3.4†)	(11.9†)

NNS2010	$q,\mathscr{C}$	$q,\mathscr{P}$	$q,\mathscr{R}$	$q,\mathscr{L}$	$\mathscr{C},\mathscr{P}$	$\mathscr{C},\mathscr{R}$	$\mathscr{C},\mathscr{L}$	$\mathscr{P},\mathscr{R}$	$\mathscr{P},\mathscr{L}$	$\mathscr{R},\mathscr{L}$
Pearson	0.78*	0.84*	0.71*	0.01	0.62*	0.46*	0.01	0.56*	0.02	0.01
Spearman	0.84*	0.56*	0.39*	0.10	0.38*	0.56*	0.04	0.17*	0.08	0.03
S_{frac}	0.96	0.88	0.80	0.71	0.87	0.82	0.72	0.76	0.70	0.75
(z-score)	(29.1†)	(20.2†)	(12.4†)	(3.7†)	(19.8†)	(14.0†)	(4.1†)	(8.7†)	(1.9†)	(7.8†)

	$q,\mathscr{LS}$	$\mathscr{C},\mathscr{LS}$	$\mathscr{P},\mathscr{LS}$	$\mathscr{R},\mathscr{LS}$	$\mathscr{L},\mathscr{LS}$
S_{frac}	0.54	0.52	0.50	0.51	0.71
(z-score)	(2.7†)	(1.5)	(−0.1)	(0.9)	(16.9†)

FB-Caltech	$q,\mathscr{C}$	$q,\mathscr{P}$	$q,\mathscr{R}$	$q,\mathscr{L}$	$\mathscr{C},\mathscr{P}$	$\mathscr{C},\mathscr{R}$	$\mathscr{C},\mathscr{L}$	$\mathscr{P},\mathscr{R}$	$\mathscr{P},\mathscr{L}$	$\mathscr{R},\mathscr{L}$
Pearson	0.96*	0.97*	0.98*	0.02	0.86*	0.97*	0.01	0.93*	0.01	0.02
Spearman	1.00*	0.99*	0.99*	0.09	0.98*	1.00*	0.08	0.97*	0.09*	0.07
S_{frac}	0.98	0.97	0.97	0.43	0.97	0.99	0.42	0.96	0.42	0.42
(z-score)	(38.4†)	(36.9†)	(36.7†)	(5.7†)	(35.5†)	(38.4†)	(5.0†)	(34.4†)	(5.5†)	(4.7†)

	$q,\mathscr{LS}$	$\mathscr{C},\mathscr{LS}$	$\mathscr{P},\mathscr{LS}$	$\mathscr{R},\mathscr{LS}$	$\mathscr{L},\mathscr{LS}$
S_{frac}	0.53	0.53	0.53	0.53	0.81
(z-score)	(4.7†)	(4.2†)	(4.4†)	(4.1†)	(25.7†)

FB-Reed	$q,\mathscr{C}$	$q,\mathscr{P}$	$q,\mathscr{R}$	$q,\mathscr{L}$	$\mathscr{C},\mathscr{P}$	$\mathscr{C},\mathscr{R}$	$\mathscr{C},\mathscr{L}$	$\mathscr{P},\mathscr{R}$	$\mathscr{P},\mathscr{L}$	$\mathscr{R},\mathscr{L}$
Pearson	0.92*	0.95*	0.98*	−0.01	0.77*	0.94*	−0.02	0.90*	−0.01	−0.01
Spearman	0.99*	0.98*	0.96*	0.07	0.96*	0.98*	0.07	0.90*	0.09*	0.05
S_{frac}	0.99	0.98	0.98	0.51	0.97	0.99	0.51	0.96	0.51	0.51
(z-score)	(41.8†)	(39.7†)	(40.2†)	(6.3†)	(39.1†)	(42.0†)	(6.5†)	(38.2†)	(6.5†)	(6.3†)

	$q,\mathscr{LS}$	$\mathscr{C},\mathscr{LS}$	$\mathscr{P},\mathscr{LS}$	$\mathscr{R},\mathscr{LS}$	$\mathscr{L},\mathscr{LS}$
S_{frac}	0.51	0.52	0.52	0.52	0.98
(z-score)	(5.4†)	(5.9†)	(5.6†)	(5.7†)	(42.5†)